# Preemptive Online Scheduling: Optimal Algorithms for All Speeds

Tomáš Ebenlendr* Wojciech Jawor† Jiří Sgall*

**Abstract**

Our main result is an optimal online algorithm for preemptive scheduling on uniformly related machines with the objective to minimize makespan. The algorithm is deterministic, yet it is optimal even among all randomized algorithms. In addition, it is optimal for any fixed combination of speeds of the machines, and thus our results subsume all the previous work on various special cases. Together with a new lower bound it follows that the overall competitive ratio of this optimal algorithm is between 2.054 and $e \approx 2.718$. We also give a complete analysis of the competitive ratio for three machines.

## 1 Introduction

We study an online version of the classical problem of preemptive scheduling on uniformly related machines.

We are given $m$ machines with speeds $s_1 \geq s_2 \geq \ldots \geq s_m$ and a sequence of jobs, each described by its processing time (length). The time needed to process a job with length $p$ on a machine with speed $s$ is $p/s$. In the preemptive version, each job may be divided into several pieces, which can be assigned to different machines in disjoint time slots. (A job may be scheduled in several time slots on the same machine, and there may be times when a partially processed job is not running at all.) The objective is to find a schedule of all jobs in which the maximal completion time (makespan) is minimized.

In the online problem, jobs arrive one-by-one and we need to assign each incoming job to some time slots on some machines, without any knowledge of the jobs that arrive later. This problem, also known as list scheduling, was first studied in Graham's seminal paper [12] for identical machines (i.e., $s_1 = \ldots = s_m = 1$), without preemption. In the preemptive version, upon arrival of a job its complete assignment at all times must be given and we are not allowed to change this assignment later. In other words, the online nature of the problem is in the order in the input sequence and it is not related to possible preemptions and the time in the schedule.

The offline scheduling with makespan objective is well understood, and results for uniformly related machines were usually obtained using similar methods as for identical machines. Exact solutions can be computed with preemptions [16, 15, 10] and approximation schemes exist for the non-preemptive version [14, 13], which is NP-hard to solve exactly.

*Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic. Partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), grant 201/05/0124 of GA ČR, and grant IAA1019401 of GA AV ČR. Email: {ebik,sgall}@math.cas.cz.

†Department of Computer Science, University of California, Riverside, CA 92521. Supported by NSF grants CCF-0208856 and OISE-0340752. Email: wojtek@cs.ucr.edu.

In the online version, the situation is quite different. For non-preemptive scheduling, tight results are known only for two related or three identical machines. Even for deterministic algorithms on $m$ identical machines, there still remains a small gap between the lower bound of 1.880 [17] and the upper bound of 1.923 [9], and a much larger gap for uniformly related machines, where the current bounds are 2.438 and 5.828 [2].

For randomized non-preemptive scheduling even less is known, the bounds are 1.581 and 1.916 for identical machines [3, 18, 1] and 2 and 4.311 for related machines [8, 2]. It is still open whether randomized algorithms are better than deterministic.

The study of preemptive scheduling was partially motivated by studying the power of randomization in the non-preemptive setting. All previous lower bounds for randomized non-preemptive scheduling, except for [19], use speed sequences where tight bounds for preemptive scheduling are known and the same lower bound (but not the algorithm) works also in randomized non-preemptive setting.

Preemptive scheduling appears to be easier compared to non-preemptive, as we can compute an optimal solution exactly, yet up to now the tight results have been limited as well (see below). Thus our optimal algorithm for preemptive online scheduling on related machines is a significant progress. It involves new technical ideas compared to both the previous optimal algorithms for special cases of speeds and the non-optimal algorithms for general speeds.

All previous online algorithms that work for arbitrary speeds, preemptive or not, were obtained by a doubling approach. This means that a competitive algorithm is designed for the case when the optimum is approximately known in advance, and then, without this knowledge, it is used in phases with geometrically increasing guesses of the optimum. Such an approach probably cannot lead to an optimal algorithm for this type of scheduling problems. Instead, our algorithm computes exactly the current optimum at each step of the sequence and takes full advantage of this knowledge.

In all the previously known optimal algorithms for special cases, the optimal algorithms try to maintain certain fixed ratio of loads on the machines, generally with largest part of each job scheduled on the fast machine. These algorithms create no "holes" in the schedules, i.e., each machine is always busy from time 0 until some time $t$ and idle afterwards. In contrast, our algorithm attempts to schedule the whole job on as slow machine as possible without violating the desired competitive ratio. This is done even at the cost of creating "holes" in the schedule, and using these holes efficiently is the key issue.

**Previous results for preemptive online scheduling.** The study of online preemptive scheduling was started by Chen *et al.* [4], who studied the case of $m$ identical machines. They gave an optimal algorithm with the competitive ratio $1/(1-(1-1/m)^m)$, which is 4/3 for $m=2$ and approaches $e/(e-1)\approx 1.582$ when $m\to\infty$.

An optimal online algorithm for the special case of two related machines was given by Wen and Du [20], and by Epstein *et al.* [7]. The optimal competitive ratio in this case is $1+s_1s_2/(s_1^2+s_1s_2+s_2^2)$.

A special case with non-decreasing speed ratios, i.e., $s_{i-1}/s_i \le s_i/s_{i+1}$ for $i=2,\dots,m-1$, was studied by Epstein [6]; note that this subsumes both identical and two related machines. Epstein gave an optimal competitive ratio for each sequence of speeds in this class; this ratio is equal to

$$R = \left(\sum_{i=1}^{m} \frac{s_i}{S}\left(1-\frac{s_1}{S}\right)^{i-1}\right)^{-1}, \qquad \text{where } S=\sum_{i=1}^{m} s_i.$$

All these algorithms are deterministic and matching lower bounds are known to hold also for randomized algorithms.

For the general case, Ebenlendr and Sgall [5] obtained a 4-competitive deterministic algorithm and an $e$-competitive randomized algorithm, where $e \approx 2.718$.

Epstein and Sgall [8] gave lower bounds on the competitive ratio for the worst case combination of speeds for any fixed $m$. These bounds approach 2 when $m \to \infty$ and all hold for randomized algorithms.

**Our results.** Our main result is an optimal online algorithm for preemptive scheduling on uniformly related machines. The algorithm achieves the best possible competitive ratio not only in the general case, but also for any number of machines and any particular combination of machine speeds. We show that although our algorithm is deterministic, its competitive ratio matches the best competitive ratio of any randomized algorithm. This proves that, similarly to the case of identical machines and other special cases studied before, randomization does not help for preemptive scheduling.

For any fixed set of speeds the competitive ratio of our algorithm can be computed by solving a linear program. We do not know, however, what is its worst case value over all speed combinations. Nevertheless, using the fact that there exists an $e$-competitive randomized algorithm [5], we conclude that our (deterministic) algorithm is also $e$-competitive.

In Section 3 we present the linear program that gives the optimal competitive ratio and the lower bound, in Section 4 we describe the algorithm and its analysis.

In Section 5 we prove that no algorithm can be better than 2.054-competitive, by providing an explicit numerical instance on 100 machines. This improves the lower bound of 2 of Epstein and Sgall [8]. Numerical calculations also give sequences that show that for $m \geq 8$ the lower bounds given by Epstein and Sgall [8] are not tight.

In Section 6 we analyze the linear program for computing the optimal competitive ratio for certain cases of speed sequences.

We show that the formula for non-decreasing speed ratios, given by Epstein [6] and mentioned above, gives an upper bound on the competitive ratio for all possible speed combinations, and we extend the region where it is proven to be optimal.

For $m = 3$ and $m = 4$ we give an exact formula for the competitive ratio for any speed combination. The global upper bound in these cases matches the lower bound in Epstein and Sgall [8].

## 2 Preliminaries

Let $M_i$, $i = 1, 2, \ldots, m$ denote the $m$ machines, and let $s_i$ be the speed of $M_i$. Without loss of generality we assume that the machines are sorted by decreasing speeds, i.e., $s_1 \geq s_2 \geq \ldots \geq s_m$. To avoid degenerate cases, we assume that $s_1 > 0$.

Let $\mathcal{J} = (p_j)_{j=1}^{n}$ denote the input sequence of jobs, where $n$ is the number of jobs and $p_j$ is the length, or processing time, of $j$th job. Given $\mathcal{J}$, let $\mathcal{J}_j$ denote a sequence that is obtained from $\mathcal{J}$ by removing the last $j - 1$ jobs.

The time needed to process a job with length $p$ on machine with speed $s$ is equal to $p/s$; each machine can process at most one job at any time. Preemption is allowed, which means that each job may be divided into several pieces, which can be assigned to different machines, but any two time slots to which a single job is assigned must be disjoint (no parallel processing of a job); there is no additional cost for preemptions. Formally, if $t_i$ denotes the total length of the time intervals when the job is assigned to machine $M_i$, it is required that $t_1 s_1 + t_2 s_3 + \cdots + t_m s_m = p$.

The objective is to find a schedule of all jobs in which the maximal completion time (makespan) is minimized. In Graham's three-field notation [11] the problem is denoted $Q|\text{pmtn}|C_{\max}$.

For an algorithm $\mathsf{A}$, let $C^{\mathsf{A}}_{\max}[\mathcal{J}]$ denote the makespan of the schedule of $\mathcal{J}$, produced by $\mathsf{A}$. By $C^*_{\max}[\mathcal{J}]$ we denote the makespan of the optimal offline schedule of $\mathcal{J}$.

In the online version of this problem, denoted $Q|\text{online-list,pmtn}|C_{\max}$, jobs arrive one-by-one and we need to assign each incoming job to some time slots on some machines, without the knowledge of the jobs that arrive later. Upon release of each job a complete assignment of this job at all times must be given.

Online algorithms are evaluated using competitive analysis. An online algorithm $\mathsf{A}$ is called $R$-competitive if for every input $\mathcal{J}$, the makespan is at most $R$ times the optimal makespan, i.e., $C^{\mathsf{A}}_{\max}[\mathcal{J}] \leq R \cdot C^*_{\max}[\mathcal{J}]$. In case of a randomized algorithm, the same must hold for every input for the expected makespan of the online algorithm, $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}]] \leq R \cdot C^*_{\max}[\mathcal{J}]$, where the expectation is taken over the random choices of the algorithm.

There are two easy lower bounds on $C^*_{\max}[\mathcal{J}]$. First, $C^*_{\max}[\mathcal{J}]$ can be bounded by the total work done on all machines. i.e.,

$$C^*_{\max}[\mathcal{J}] \quad \geq \quad \frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{m} s_i}. \tag{1}$$

Second, the makespan of the optimal schedule is at least the makespan of the optimal schedule of any $\ell$ jobs. For $\ell < m$ this latter schedule uses only $\ell$ fastest machines, so the work of any $\ell$ jobs must fit on these machines. So,

$$C^*_{\max}[\mathcal{J}] \quad \geq \quad \frac{P_\ell}{\sum_{i=1}^{\ell} s_i} \qquad \text{for } \ell = 1, \ldots, m-1, \tag{2}$$

where $P_\ell$ denotes the sum of $\ell$ largest processing times in $\mathcal{J}$. The following is a known fact [15, 10, 5].

**Fact 2.1** *the value of $C^*_{\max}[\mathcal{J}]$ is the minimal value that satisfies (1) and (2).*

The following lemma is due to Epstein and Sgall [8]. We include the proof, as it is very helpful in understanding our results.

**Lemma 2.2 ([8])** *For any randomized $R$-competitive on-line algorithm $\mathsf{A}$ for preemptive scheduling on $m$ machines, and for any input sequence $\mathcal{J}$ we have*

$$\sum_{j=1}^{n} p_j \quad \leq \quad R \cdot \sum_{i=1}^{m} s_i C^*_{\max}[\mathcal{J}_i].$$

*For non-preemptive scheduling, the same holds if $C^*_{\max}$ refers to the non-preemptive optimal makespan.*

*Proof:* Fix a sequence of random bits used by $\mathsf{A}$. Let $T_i$ denote the last time when at most $i$ machines are running and set $T_{m+1} = 0$. First observe that

$$\sum_{j=1}^{n} p_j \;\leq\; \sum_{i=1}^{m} s_i T_i. \tag{3}$$

During the time interval $(T_{i+1}, T_i]$ at most $i$ machines are busy, and their total speed is at most $s_1 + s_2 + \ldots + s_i$. Thus the maximum possible work done in this interval is $(T_i - T_{i+1})(s_1 + s_2 + \ldots + s_i)$. Summing over all $i$, we obtain $\sum_{i=1}^{m} s_i T_i$. In any valid schedule all the jobs are completed, so (3) follows.

Since the algorithm is online, the schedule for $\mathcal{J}_i$ is obtained from the schedule for $\mathcal{J}$ by removing the last $i-1$ jobs. At time $T_i$ there are at least $i$ jobs running, thus after removing $i-1$ jobs at least one machine is busy at $T_i$. So we have $T_i \leq C^{\mathsf{A}}_{\max}[\mathcal{J}_i]$ for any fixed random bits. Averaging over random bits of the algorithm and using (3), we have

$$\sum_{j=1}^{n} p_j \;\leq\; \mathbb{E}\left[\sum_{i=1}^{m} s_i C^{\mathsf{A}}_{\max}[\mathcal{J}_i]\right] \;=\; \sum_{i=1}^{m} s_i \mathbb{E}\left[C^{\mathsf{A}}_{\max}[\mathcal{J}_i]\right].$$

Since $\mathsf{A}$ is $R$-competitive, i.e., $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}_i]] \leq R \cdot C^{*}_{\max}[\mathcal{J}_i]$, the lemma follows. $\Box$

# 3 The optimal competitive ratio and the lower bound

The optimal competitive ratio for given speeds $s_1, \ldots, s_m$ turns out to be equal to the best lower bound obtained by Lemma 2.2. In this section we formalize this bound using a linear program and prove the lower bound.

For each input sequence $\mathcal{J} = (p_j)_{j=1}^{n}$, Lemma 2.2 shows that the competitive ratio is at least $\sum_{j=1}^{n} p_j / (\sum_{i=1}^{m} s_i C^{*}_{\max}[\mathcal{J}_i])$. The set of input sequences can be restricted in two ways. First, we may assume that the processing times of jobs are non-decreasing: Sorting the jobs can only decrease the values of $C^{*}_{\max}[\mathcal{J}_i]$, as in each of these partial instances some jobs are possibly replaced by smaller jobs; thus the bound on $R$ can only increase. Second, the bound is invariant under scaling, so we may assume that $\sum_{i=1}^{m} s_i C^{*}_{\max}[\mathcal{J}_i] = 1$. The lower bound is then simply the sum of all processing times.

Now it is easy to give a linear program to compute the optimal lower bound, given the parameters $s_1 \geq \cdots \geq s_m$. The linear program has variables $q_1, q_2, \ldots, q_m, O_1, O_2, \ldots, O_m$. Variable $q_1$ corresponds to the sum of all processing times in $\mathcal{J}_m$, variables $q_2, \ldots, q_m$ to the processing times of the last $m-1$ jobs, and variables $O_k$ correspond to $C^{*}_{\max}[\mathcal{J}_{m-k+1}]$.

**Definition 3.1** *Let $r(s_1, \ldots, s_m)$ denote the value of the objective function of the optimal solution of the following linear program:*

$$\begin{array}{rcll}
\textbf{maximize} & & r(s_1, \ldots, s_m) = q_1 + q_2 + q_3 + \ldots + q_m & \\
\textbf{subject to} & & & \\
q_1 + \ldots + q_k & \leq & (s_1 + s_2 + \ldots + s_m) O_k & \text{for } k = 1, \ldots, m \\
q_j + q_{j+1} + \ldots + q_k & \leq & (s_1 + s_2 + \ldots + s_{k-j+1}) O_k & \text{for } 2 \leq j \leq k \leq m \\
1 & = & s_1 O_m + s_2 O_{m-1} + \ldots + s_m O_1 & \\
q_j & \leq & q_{j+1} & \text{for } j = 2, \ldots, m-1 \\
0 & \leq & q_1 & \\
0 & \leq & q_2 &
\end{array} \tag{4}$$

The linear program has a feasible solution with the only non-zero variable $O_m = 1/s_1$. It is also easy to see that the objective function is bounded, the constraints imply that $q_1 + q_2 + \ldots + q_m \leq (s_1 + s_2 + \ldots + s_m)O_m \leq m \cdot s_1 O_m \leq m$. Thus the value $r(s_1, \ldots, s_m)$ is well-defined. Finally, note that the linear program has a quadratic number of constraints, and thus it can be solved efficiently.

**Theorem 3.2** *Any randomized online algorithm for $m$ machines with speeds $s_1 \geq s_2 \geq \cdots \geq s_m$ has competitive ratio at least $r(s_1, \ldots, s_m)$.*

*Proof:* There exist values $q_1^*$, $q_2^*$, …, $q_m^*$, $O_1^*$, $O_2^*$, …, $O_m^*$ of variables $q_1$, $q_2$, …, $q_m$, $O_1$, $O_2$, …, $O_m$, which satisfy all the constraints of the linear program (4) and $r(s_1, \ldots, s_m) = q_1^* + q_2^* + \ldots + q_m^*$. Create instance $\mathcal{I}$ as follows: The first $m$ jobs have processing times $p_1 = \cdots = p_m = q_1^*/m$. The remaining $m - 1$ jobs have processing times $p_{m+1} = q_2^*$, $p_{m+2} = q_3^*$, …, $p_{2m-1} = q_m^*$.

We claim that the first two families of constraints of (4) guarantee that the values $O_k^*$ satisfy (1) and (2) for $C^*_{\max}[\mathcal{I}_{m-k+1}]$. This is obvious for (1) and also for the bound (2) if the set of $\ell$ largest jobs contains only jobs larger than $p_1$. However, if $\ell$ largest jobs include a job with processing time $p_1$ then we claim that the right-hand side of (2) is upper-bounded either by (2) for $\ell' < \ell$ such that $\ell'$ largest jobs do not contain any job of processing time $p_1$, or by (1) which includes all $m$ jobs with processing time $p_1$: Suppose that the bound including some job of size $p_1$ and machines up to speed $s_\ell$ for $\ell < m$ is maximal and strictly larger than (1) and choose the largest such $\ell$. Then we have $(p_1 + X)/(s_\ell + Y) \geq X/Y$ where $X$ and $Y$ denote the remaining sums of processing times and speeds, so that $X/Y$ is the previous bound from (2). In this case we have $p_1/s_\ell \geq X/Y$, thus also $p_1/s_{\ell+1} \geq p_1/s_\ell \geq X/Y$ and $(p_1 + p_1 + X)/(s_{\ell+1} + s_\ell + Y) \geq (p_1 + X)/(s_\ell + Y)$. Thus the bound for a larger $\ell$ (or the bound in (1)) is at least as large, contradicting the choice of $\ell$ and completing the proof of the claim. Fact 2.1 now implies that $C^*_{\max}[\mathcal{I}_{m-k+1}] \leq O_k^*$ for $k = 1, 2, \ldots, m$.

Finally, Lemma 2.2 implies that the competitive ratio of any algorithm is at least

$$\frac{q_1^* + q_2^* + \ldots + q_m^*}{\sum_{i=1}^m s_i C^*_{\max}[\mathcal{J}_i]} \geq \frac{q_1^* + q_2^* + \ldots + q_m^*}{s_1 O_m^* + \ldots + s_m O_1^*} = r(s_1, \ldots, s_m),$$

using the constraint $s_1 O_m^* + \ldots + s_m O_1^* = 1$ in the last step. □

# 4 The optimal algorithm

In this section we present the $r(s_1, \ldots, s_m)$-competitive algorithm RatioStretch for all combinations of speeds.

The idea of the algorithm is fairly natural. First we compute the desired competitive ratio for the given speeds, $r = r(s_1, \ldots, s_m)$. Next, for each arriving job, we compute the optimal makespan for jobs that have arrived so far and run the incoming job as slow as possible so that it finishes at $r$ times the computed optimal makespan. There are many ways of creating such a schedule given the flexibility of preemptions. We choose a particular one based on the notion of a *virtual machine* from [5]. Given a schedule, a virtual machine at each time corresponds to one of the real machines that are idle. This assignment can vary at different times in the schedule. Due to preemption, a virtual machine can be thought and used as a single machine with changing speed. Initially virtual machines are the real machines. The idea of the algorithm is to schedule each job on two adjacent virtual machines, and update the virtual machines as necessary.

We define the $i$th *virtual machine*, denoted $V_i$, so that at each time $\tau$ contains the $i$th fastest machine among those real machines $M_1$, $M_2$, …, $M_m$ that are idle at time $\tau$. When we schedule (a part of) a job on a virtual machine during some interval, we actually schedule it on the corresponding real machines that are uniquely defined at each time; this is always possible to achieve using preemptions. To simplify the description of the algorithm, we assume that there are infinitely many real machines of speed zero, i.e., $s_i = 0$ for any $i > m$. Scheduling a job on one of these zero-speed machines means that we do not schedule the job at the given time at all. Initially, each virtual machine $V_i$ corresponds to the real machine $M_i$; as the incoming jobs are scheduled, the assignment of the real machines to the virtual machines changes.

In our algorithm, upon arrival of a job $j$ we compute a value $T_j$ defined as $r$ times the current optimal makespan. Then we find two adjacent virtual machines $V_k$ and $V_{k+1}$, and time $t_j$, such that if we schedule $j$ on $V_{k+1}$ in the time interval $(0, t_j]$ and on $V_k$ from $t_j$ on, then $j$ finishes exactly at time $T_j$. It is essential that each job is stretched over the whole interval $(0, T_j]$, which is the maximal time interval which it can use without violating the desired competitive ratio. Next we update the virtual machines, which means that in the interval $(0, T_j]$ we merge $V_k$ and $V_{k+1}$ into $V_k$ and shift machines $V_{i+1}$, $i > k$, to $V_i$. Then we continue with the next job. This gives a complete informal description of the algorithm sufficient for its implementation.

To prove that our algorithm works, it is sufficient to show that each job $j$ scheduled on $V_1$, completes by time $T_j$; this is equivalent to the fact that we can schedule $j$ as described above. We show that this is true due to our choice of $r$.

To facilitate the proof, we maintain an assignment of scheduled jobs (and consequently busy machines at each time) to the set of virtual machines, i.e., for each virtual machine $V_i$ we compute a set $\mathcal{S}_i$ of jobs assigned to $V_i$. Although the incoming job $j$ is split between two different virtual machines, at the end of each iteration each scheduled job belongs to exactly one set $\mathcal{S}_i$, since right after $j$ is scheduled the virtual machines executing this job are merged (during the execution of $j$). We stress that the sets $\mathcal{S}_i$ serve only as means of bookkeeping for the purpose of the proof, and their computation is not an integral part of the algorithm.

See Figure 1 for an example.

At each time $\tau$, machine $M_{i'}$ belongs to $V_i$ if it is the $i$th fastest idle machine at time $\tau$, or if it is running a job $j \in \mathcal{S}_i$ at time $\tau$. At each time $\tau$ the real machines belonging to $V_i$ form a set of adjacent real machines, i.e., all machines $M_{i'}, M_{i'+1}, \ldots, M_{i''}$ for some $i' \leq i''$. This relies on the fact that we always schedule a job on two adjacent virtual machines which are then merged into a single virtual machine during the times when the job is running, and on the fact that these time intervals $(0, T_j]$ increase with $j$, as adding new jobs cannot decrease the optimal makespan.

Let $v_i(t)$ denote the speed of the virtual machine $V_i$ at time $t$, which is the speed of the unique idle real machine that belongs to $V_i$. Let $W_i(t) = \int_0^t v_i(\tau)d\tau$ be the total work which can be done on machine $V_i$ in the time interval $(0, t]$. By definition we have $v_i(t) \geq v_{i+1}(t)$ and thus also $W_i(t) \geq W_{i+1}(t)$ for all $i$ and $t$. Note also that $W_{m+1}(t) = v_{m+1}(t) = 0$ for all $t$.

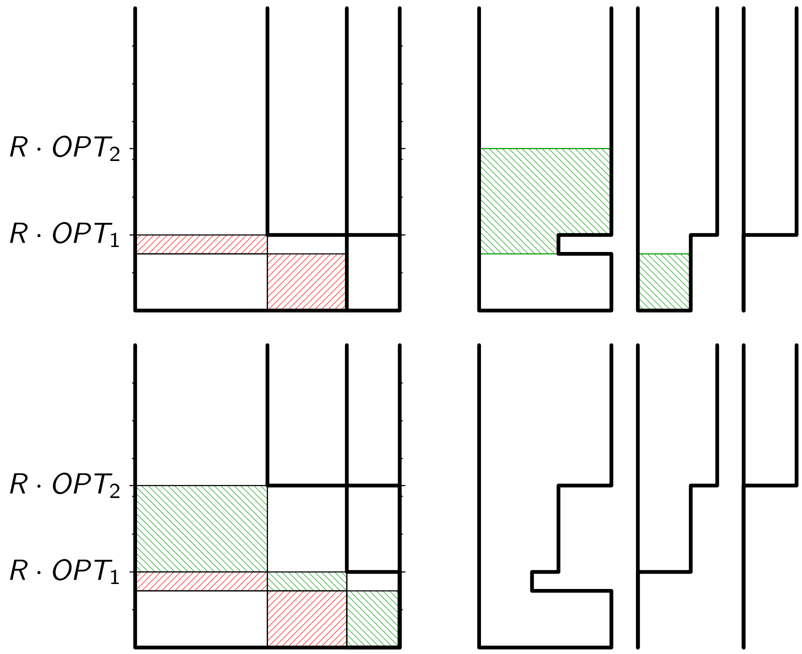



Figure 1: An illustration of a schedule of two jobs on three machines produced by RatioStretch. Vertical axis denotes the time, horizontal axis corresponds to the speed of the machines. The pictures on the left depict the schedule on the real machines, with bold lines separating the virtual machines. The pictures on the right show only the idle time on the virtual machines. The top pictures show the situation after the first job, with the second job being scheduled on the first two virtual machines. The bottom pictures show the situation with after the second job is scheduled and virtual machines updated.

**Algorithm** RatioStretch. First solve the linear program (4) for a fixed sequence of speeds $s_1 \geq s_2 \geq \cdots \geq s_m$ given on input. Let $r = r(s_1, \ldots, s_m)$ be the optimal objective value. Also initialize $T_0 := 0$, $\mathcal{S}_i := \emptyset$, $v_i(\tau) := s_i$, and $v_{m+1}(\tau) := 0$ for all $i = 1, 2, \ldots, m$ and $\tau \geq 0$. For each arriving job $j$, compute the output schedule as follows:

(1) Let $T_j := r \cdot C^*_{\max}[(p_i)_{i=1}^j]$.

(2) Find the smallest $k$ such that $W_k(T_j) \geq p_j \geq W_{k+1}(T_j)$. If such $k$ does not exist, then output "failed" and stop. Otherwise find time $t_j \in [0, T_j]$ such that $W_{k+1}(t_j) + W_k(T_j) - W_k(t_j) = p_j$.

(3) Schedule job $j$ on $V_{k+1}$ in time interval $(0, t_j]$ and on $V_k$ in time interval $(t_j, T_j]$.

(4) Set $v_k(\tau) := v_{k+1}(\tau)$ for $\tau \in (t_j, T_j]$, and $v_i(\tau) := v_{i+1}(\tau)$ for $i = k+1, \ldots, m$ and $\tau \in (0, T_j]$. Also set $\mathcal{S}_k := \mathcal{S}_k \cup \mathcal{S}_{k+1} \cup \{j\}$, and $\mathcal{S}_i := \mathcal{S}_{i+1}$ for $i = k+1, \ldots, m$.

We leave out implementation details. We only note that job $j$ can be preempted only at times $T_{j'}$ for $j' < j$ or at times $t_{j'}$ for $j' \leq j$, i.e., at most $2j - 1$ times. The total number of preemptions is at most $n(m+1)$, since at most $m-1$ jobs are preempted at each time $T_j$ and at most two jobs are preempted at each time $t_j$. This also implies that the functions $v_i$ and $W_i$ are piecewise linear with at most $2n$ parts. Thus it is possible to represent and process them efficiently. The computation of $r$ and $T_j$ is efficient as well.

We also note that if we use any $r' \geq r(s_1, \ldots, s_m)$ in place of $r$ in the algorithm, we obtain an $r'$-competitive algorithm. Thus if for some sets of speeds we know a good upper bound on $r(s_1, \ldots, s_m)$, we may use it instead of solving the linear program exactly.

**Theorem 4.1** *Algorithm* RatioStretch *is $r = r(s_1, \ldots, s_m)$ competitive for online preemptive scheduling on $m$ uniformly related machines with speeds $s_1 \geq s_2 \geq \cdots \geq s_m$.*

*Proof:* If RatioStretch schedules a job, it is always completed at time $T_j \leq r \cdot C^*_{\max}[(p_i)_{i=1}^n]$. Thus to prove the theorem, it is sufficient to guarantee that the algorithm does not fail to find machines $V_k$ and $V_{k+1}$ for the incoming job $j$. This is equivalent to the statement that there is always enough space on $V_1$, i.e., that $p_j \leq W_1(T_j)$ in the iteration when $j$ is to be scheduled. Since $W_{m+1} \equiv 0$, this is sufficient to guarantee that required $k$ exists. Given the choice of $k$, it is always possible to find time $t_j$ as the expression $W_{k+1}(t_j) + W_k(T_j) - W_k(t_j)$ is continuous in $t_j$, for $t_j = 0$ it is equal to $W_k(T_j) \geq p_j$, and for $t_j = T_j$ it is equal to $W_{k+1}(T_j) \leq p_j$.

To avoid cases, we assume that the input sequence starts by $m-1$ jobs with processing time 0. Adding these jobs does not affect any of the optimal makespans computed during the algorithm. In RatioStretch, they are assigned to $V_1$, but they are actually never running (they complete at time 0) and thus do not affect the schedule produced by RatioStretch.

Consider now all the jobs scheduled on the first virtual machine, i.e., the set $\mathcal{S}_1$. Let $j_1, j_2, \ldots, j_{m-1}$ denote the last $m-1$ jobs in $\mathcal{S}_1$, ordered as they appear on input. Let $\mathcal{I}$ be the sequence of the remaining jobs in $\mathcal{S}_1$, and let $P$ be the total processing time of jobs in $\mathcal{I}$. Finally, let $j_m = j$ be the incoming job.

Consider any $i = 1, \ldots, m$ and any time $\tau \in (0, T_{j_i}]$. Using the fact that the times $T_j$ are non-decreasing in $j$ and that the algorithm stretches each job $j$ over the whole interval $(0, T_j]$, there are at least $m - i$ jobs from $\mathcal{S}_1$ running at $\tau$, namely jobs $j_i, j_{i+1}, \ldots, j_{m-1}$. Including the idle machine, there are at least $m + 1 - i$ real machines belonging to $V_1$. Since $V_1$ is the first virtual machine and the real machines are adjacent, they must include the fastest real machines $M_1, \ldots, M_{m+1-i}$. It follows that the total work that can be processed on the real machines belonging to $V_1$ during the interval $(0, T_{j_m}]$ is at least $s_1 T_{j_m} + s_2 T_{j_{m-1}} + \cdots + s_m T_{j_1}$. The total processing time of jobs in $\mathcal{S}_1$ is $P + p_{j_1} + p_{j_2} + \cdots + p_{j_{m-1}}$. Thus to prove that $j_m$ can be scheduled on $V_1$ we need to verify that

$$p_{j_m} \leq s_1 T_{j_m} + s_2 T_{j_{m-1}} + \cdots + s_m T_{j_1} - (P + p_{j_1} + p_{j_2} + \cdots + p_{j_{m-1}}). \tag{5}$$

Let $\nu_1, \nu_2, \ldots, \nu_m$ be the sequence of jobs $j_1, j_2, \ldots j_m$ ordered so that the processing times $p_{\nu_i}$ are non-decreasing, i.e., $p_{\nu_i} \leq p_{\nu_{i+1}}$ for $i = 1, \ldots, m-1$. We claim that for each $i = 1, \ldots, m$,

$$\begin{aligned} T_{j_i} = r \cdot C^*_{\max}[(p_1, p_2, \ldots, p_{j_i})] &\geq r \cdot C^*_{\max}[(\mathcal{I}, p_{j_1}, p_{j_2}, \ldots, p_{j_i})] \\ &\geq r \cdot C^*_{\max}[(\mathcal{I}, p_{\nu_1}, p_{\nu_2}, \ldots, p_{\nu_i})]. \end{aligned} \tag{6}$$

The first equality is the definition of $T_{j_i}$. The next inequality follows since removing some jobs from the input sequence cannot increase $C^*_{\max}$. Finally, replacing the jobs $p_{j_1}, p_{j_2}, \ldots, p_{j_i}$ by jobs $p_{\nu_1}, p_{\nu_2}, \ldots, p_{\nu_i}$ can be thought as replacing some of the jobs by smaller ones and then permuting them; this also cannot increase $C^*_{\max}$.

The inequality (6) and the fact that $p_{j_1} + p_{j_2} + \cdots + p_{j_m} = p_{\nu_1} + p_{\nu_2} + \cdots + p_{\nu_m}$ together imply that to prove (5), it is sufficient to prove

$$P + p_{\nu_1} + p_{\nu_2} + \cdots + p_{\nu_m} \quad \leq \quad r \cdot \sum_{i=1}^{m} s_i C^*_{\max}[(\mathcal{I}, p_{\nu_1}, p_{\nu_2}, \ldots, p_{\nu_{m-i+1}})]. \tag{7}$$

Let $\sigma = \sum_{i=1}^{m} s_i C^*_{\max}[(\mathcal{I}, p_{\nu_1}, p_{\nu_2}, \ldots, p_{\nu_i})]$. Let $q_1 = (P + p_{\nu_1})/\sigma$, $q_j = p_{\nu_j}/\sigma$, for $j = 2, 3, \ldots, m$, and let $O_k = C^*_{\max}[(\mathcal{I}, p_{\nu_1}, p_{\nu_2}, \ldots, p_{\nu_k})]/\sigma$ for $k = 1, \ldots, m$. These values satisfy all constraints of (4), as follows by using inequalities (1) and (2) for instances $(\mathcal{I}, p_{\nu_1}, p_{\nu_2}, \ldots, p_{\nu_k})$. Thus $(P + p_{\nu_1} + p_{\nu_2} + \cdots + p_{\nu_m})/\sigma = q_1 + q_2 + \cdots + q_m \leq r$, as $r$ is defined as the maximum of the objective of the linear program (4). This proves (7) and thus also (5) and correctness of the algorithm. □

Theorems 3.2 and 4.1 show that Algorithm RatioStretch is as good as any randomized algorithm. Together with the $e$-competitive randomized algorithm from [5] we obtain the following:

**Corollary 4.2** *Algorithm* RatioStretch *is $e$-competitive for online preemptive scheduling on uniformly related machines with arbitrary speeds, where $e \approx 2.718$.*

# 5 Numerical lower bounds

We have the optimal algorithm for arbitrary speeds, but we do not know the numerical value of its competitive ratio. The competitive ratio for $m$ machines is equal to the solution of a quadratic program obtained from (4) by considering $s_i$ to be variables (in addition to $q_j$ and $O_k$). However, this quadratic program is not convex and we do not know how to solve it. We have obtained some lower bounds numerically using mathematical software Maple.

A lower bound is simply a feasible solution of (4). Once the values of $s_i$ are given, verification only involves solving a linear program. Once also the optimal values $q_j^*$ are given, it is trivial to compute values $O_k^*$ and verify the constraints of (4). A file with our solutions for $m = 3, \ldots, 70$ and $m = 100$ in a format suitable for computer verification is available at `http://math.cas.cz/sgall/ps/optrel/`.

We obtain lower bounds improving the bounds from [8] for $m \geq 8$. For $m \geq 58$ the bounds are above 2, and for $m = 100$ we obtain a speed combination with optimal competitive ratio above 2.054, see appendix A. Thus we have:

**Theorem 5.1** *For any online algorithm for preemptive scheduling on uniformly related machines, the competitive ratio is at least* 2.054*.*

# 6 Special cases

One approach to analyze the optimal competitive ratio is to give a symbolic solution to the linear program (4). A feasible primal solution gives a lower bound (which can be easily turned into an sequence of jobs, as we have seen before). A feasible solution of a dual linear program gives an upper bound on the competitive ratio. A dual solution actually means that we form a positive linear combination of some of the linear constraints so that the resulting inequality bounds the objective function by the desired competitive ratio.

A basic solution of the linear program is described by giving a subset of constraints where equality holds. If for some range of speeds this subset does not change in the optimal solution, the optimal competitive ratio is given by a solution of this system of equation with $m$ variables and coefficients that are linear functions of speeds. Thus it is a rational function of the speeds of degree $m$. However, in general, for different speed sequences we need to use different subsets of the constraints. In general, therefore, the competitive ratio is a piecewise rational function of degree $m$ of speeds.

We give two cases where we can provide analysis along these lines. First we generalize the case of non-decreasing speed ratios from [6]. We prove that the same formula is a general upper bound on the competitive ratio and that it is actually optimal for a slightly larger region of speed sequences. Then we give a complete analysis of the case $m = 3$ and the resulting formula for $m = 4$. The proof for $m = 4$ involves more of similar case analysis, which will appear in a separate publication. It turns out that the result involves two (for $m = 3$) and four (for $m = 4$) regions with different formulas for the optimal competitive ratio, always one coming from the (extension of) the case of non-decreasing speed ratios. Our analysis of the cases $m = 3, 4$ also implies that the lower bound for $m = 3, 4$ from [8] is tight. Finally, let us remark that in the table in [8] describing these bounds, there is an error, namely the column of $k$ shows wrong values; however the actual bounds are computed correctly.

We denote $\quad S = \sum_{i=1}^{m} s_i, \quad \alpha = 1 - \frac{s_1}{S} = \frac{s_2 + \cdots + s_m}{s_1 + s_2 + \cdots + s_m}.$

### 6.1 Beyond non-decreasing speed ratios

**Theorem 6.1** *Let*

$$R = \left( \sum_{i=1}^{m} \frac{s_i}{S} \alpha^{i-1} \right)^{-1}.$$

*Then $r(s_1, \ldots, s_m) \leq R$ for any speeds $s_1 \geq \cdots \geq s_m$ and thus* RatioStretch *is $R$-competitive. Furthermore $r(s_1, \ldots, s_m) = R$ whenever*

$$\left(1 + \alpha + \ldots + \alpha^{i-1}\right) s_1 \leq s_1 + \ldots + s_i \qquad \text{for all } i = 2, \ldots, m-1. \tag{8}$$

*Proof:* **The upper bound.** As described in the outline above, we form a positive linear combination of some constraints of the linear program (4). Here we use the constraints corresponding to (2) with only one largest job and two constraints corresponding to (1). We put $x_1 = 0$ and add up

$$\begin{aligned} q_k &\leq s_1 O_k \quad \text{for } 2 \leq k \leq m, \quad \text{times } x_k = \sum_{i=2}^{k} s_{m-k+i} \alpha^{i-2} \qquad (9) \\ q_1 + \ldots + q_k &\leq S O_k \quad \text{for } 1 \leq k \leq m, \quad \text{times } y_k = s_{m-k+1} - \frac{s_1}{S} x_k \end{aligned}$$

We need to show that $y_k \geq 0$ and simplify the resulting inequality. For all $k = 1, \ldots, m$ we have

$$\begin{aligned} x_k + y_k &= s_{m-k+1} + \left(1 - \frac{s_1}{S}\right) \sum_{i=2}^{k} s_{m-k+i} \alpha^{i-2} \\ &= \sum_{i=1}^{k} s_{m-k+i} \alpha^{i-1} = \sum_{i=2}^{k+1} s_{m-k+i-1} \alpha^{i-2}, \qquad (10) \\ \text{and thus} \quad y_k &= \sum_{i=2}^{k+1} s_{m-k+i-1} \alpha^{i-2} - x_k \geq \sum_{i=2}^{k} (s_{m-k+i-1} - s_{m-k+i}) \alpha^{i-2} \geq 0. \end{aligned}$$

In addition, (10) implies that $x_k + y_k = x_{k+1}$ for $k < m$, and $x_m + y_m = \sum_{i=1}^{m} s_i \alpha^{i-1} = S/R$. Using also the fact that $y_1 = s_m = x_2$, the left-hand side of the linear combination given by (9) is equal to

$$\begin{aligned} &q_1(y_1 + y_2 \ldots + y_m) + q_2(x_2 + y_2 + \ldots + y_m) + q_3(x_3 + y_3 + \ldots + y_m) + \cdots + q_m(x_m + y_m) \\ &\quad = (q_1 + \ldots + q_m)\frac{S}{R}. \end{aligned}$$

The right-hand side of the linear combination given by (9) is equal to

$$y_1 S O_1 + (x_2 s_1 + y_2 S) O_2 + \ldots + (x_m s_1 + y_m S) O_m \;=\; S(s_1 O_m + \ldots + s_m O_1) \;=\; S,$$

using the definitions of $x_k$ and $y_k$ and the third constraint of (4). We conclude that any feasible solution of (4) satisfies the linear combination given by (9), which simplifies to $(q_1 + \ldots + q_m)S/R \le S$, and thus $r(s_1, \ldots, s_m) \le R$ for any speeds $s_i$.

**The lower bound.** Now we give a primal solution of (4) with the value of objective $R$, assuming (8). Naturally, this exactly corresponds to the lower bound from [6]. The solution is:

$$\begin{aligned} O_k &= R\frac{\alpha^{m-k}}{S} && \text{for } k = 1, \ldots, m \\ q_1 &= R\alpha^{m-1} \\ q_k &= s_1 O_k \;=\; R\frac{s_1}{S}\alpha^{m-k} && \text{for } k = 2, \ldots, m. \end{aligned}$$

We verify all the constraints of (4). The verification is straightforward; it uses the assumption (8) for the second constraint of (4).

Trivially $q_1 \ge 0$, $q_2 \ge 0$. From $\alpha \le 1$ we get $q_k \le q_{k+1}$ for $i = 2, \ldots, m-1$. For the third constraint of (4) we have

$$s_1 O_m + \ldots + s_m O_1 \;=\; R\left(\frac{s_1}{S}\alpha^0 + \frac{s_2}{S}\alpha^1 + \ldots + \frac{s_m}{S}\alpha^{m-1}\right) \;=\; 1$$

For the first constraint of (4), summing a geometric sequence and using the definition of $\alpha$ we have

$$\begin{aligned} q_1 + \ldots + q_k &= R\left(\alpha^{m-1} + \frac{s_1}{S}\left(\alpha^{m-2} + \ldots + \alpha^{m-k}\right)\right) \\ &= R\left(\alpha^{m-1} + \frac{s_1}{S} \cdot \frac{\alpha^{m-k} - \alpha^{m-1}}{1-\alpha}\right) \\ &= R\left(\alpha^{m-1} + \alpha^{m-k} - \alpha^{m-1}\right) \;=\; S O_k. \end{aligned}$$

This also shows that the objective function is $q_1 + \ldots + q_m = S O_m = R$. Finally, we check the second constraint of (4). We have

$$\begin{aligned} q_i + \ldots + q_k &= \frac{R s_1}{S}(\alpha^{m-i} + \ldots + \alpha^{m-k}) \\ &= O_k s_1 (1 + \alpha + \ldots + \alpha^{k-i}) \;\le\; O_k(s_1 + \ldots s_{k-i+1}). \end{aligned}$$

The last inequality is trivial for $k = i$ and follows from the assumption (8) for $2 \le i < k \le m$.
□

We remark that the upper bound in the previous theorem is not bounded by any constant in the region where the condition (8) does not hold.

Epstein [6] in Claim 1 exactly proves that the condition (8) is satisfied by speed sequences with non-decreasing speed ratios. In her notation, speeds are listed in reversed order and normalized so that the maximal speed is $s_m = 1$; her $x$ is our $1/\alpha$. However, (8) is satisfied for a slightly wider range of speeds. One example is $s_1 = 2$, $s_2 = s_3 = 1$.

### 6.2 Three machines

For $m = 3$, the condition (8) has only one inequality $s_1\alpha \le s_2$ and this is equivalent to $\frac{s_1}{s_2} \le \frac{s_2}{s_3} + 1$. (Substituting for $\alpha$ in the first formula, it is equivalent to $s_1(s_2 + s_3) \le s_2(s_1 + s_2 + s_3)$, which after canceling equal terms and dividing by $s_2 s_3$ gives the second formula.) This inequality turns out to be the condition that distinguishes two cases of the formula computing the optimal competitive ratio.

**Theorem 6.2** *For $m = 3$ and any speeds $s_1 \ge s_2 \ge s_3$,*

$$r(s_1, s_2, s_3) = \begin{cases} \left(\frac{s_1}{S} + (1 - \frac{s_1}{S})\frac{s_2}{S} + (1 - \frac{s_1}{S})^2 \frac{s_3}{S}\right)^{-1} & \text{if } \frac{s_1}{s_2} \le \frac{s_2}{s_3} + 1 \\ \dfrac{S^2}{s_1^2 + s_2^2 + s_3^2 + s_1 s_2 + s_1 s_3 + s_2 s_3} & \text{if } \frac{s_1}{s_2} \ge \frac{s_2}{s_3} + 1 \end{cases}$$

*The function $r(s_1, s_2, s_3)$ has maximal value $\frac{37+7\sqrt{7}}{38} \approx 1.461$, and thus this is also the optimal competitive ratio for $m = 3$ over all speeds.*

*Proof:* The first case follows directly from Theorem 6.1 and the observation that the case condition is equivalent to (8).

For the second case, let $R = S^2/(s_1^2 + s_2^2 + s_3^2 + s_1 s_2 + s_1 s_3 + s_2 s_3)$.

**The upper bound.** We again form a positive linear combination of some constraints of the linear program (4). We add up

$$\begin{array}{lllllllll} q_1 & & & & \le & SO_1 & & & \text{times} \quad s_3 \\ q_1 & + & q_2 & & \le & & SO_2 & & \text{times} \quad s_2 \\ q_1 & + & q_2 & + \; q_3 & \le & & & SO_3 & \text{times} \quad (s_1^2 - s_2 s_3)/S \\ & & q_2 & + \; q_3 & \le & & & (s_1 + s_2)O_3 & \text{times} \quad s_3 \\ & & & q_3 & \le & & & s_1 O_3 & \text{times} \quad s_2. \end{array}$$

Note that $s_1^2 \ge s_2 s_3$ by the ordering of the speeds, so we have a non-negative linear combination. In the resulting inequality, the coefficients of all $q_k$ are equal to

$$s_3 + s_2 + \frac{s_1^2 - s_2 s_3}{S} = \frac{s_1 s_3 + s_2 s_3 + s_3^2 + s_1 s_2 + s_2^2 + s_1^2}{S} = \frac{S}{R}.$$

The coefficient of $O_3$ equals $s_1^2 - s_2 s_3 + (s_1 + s_2)s_3 + s_1 s_2 = s_1 S$. Thus the resulting inequality is $(q_1 + q_2 + q_3)S/R \le S(s_1 O_3 + s_2 O_2 + s_3 O_1) = S$, using the third constraint of (4). We obtain $r(s_1, s_2, s_3) \le R$.

**The lower bound.** We put $q_k = s_{4-k}R/S$ and $O_k = (s_{4-k} + \cdots + s_3)R/S^2$, for $k = 1, 2, 3$. We have $q_1 + q_2 + q_3 = R$ which gives the desired value of the objective. We check that this is a feasible solution of (4). For the inequality bounding $q_2$ we use the case condition $s_1/s_2 \geq s_2/s_3 + 1$.

$$\begin{aligned}
q_1 + \cdots + q_k &= (s_1 + \cdots + s_{4-k})R/S \;=\; SO_k, \\
q_2 &= (s_1 s_2 + s_2(s_2 + s_3))R/S^2 \;\leq\; (s_1 s_2 + s_1 s_3)R/S^2 \;=\; s_1 O_2, \\
q_3 &= s_1 R/S \;=\; s_1 O_3, \\
q_2 + q_3 &= (s_1 + s_2)R/S \;=\; (s_1 + s_2)O_3, \\
s_1 O_3 + s_2 O_2 + s_3 O_1 &= (s_1(s_1 + s_2 + s_3) + s_2(s_2 + s_3) + s_3^2)R/S^2 \;=\; 1 \\
q_2 &= s_2 R/S \;\leq\; s_1 R/S \;=\; q_3
\end{aligned}$$

**The overall ratio.** Given the explicit formula, it is straightforward to calculate the overall competitive ratio for $m = 3$. First we note that if $s_2 > s_3$ then $r(s_1, s_2-\varepsilon, s_3+\varepsilon) > r(s_1, s_2, s_3)$ for a sufficiently small $\varepsilon > 0$ such that both points are covered by the same case of the formula. Thus the maximum is achieved on a point with $s_2 = s_3$. We can scale the speeds so that $S = 1$, and then we are maximizing a polynomial function of a single variable. The case $s_1/s_2 \leq s_2/s_3 + 1$ is maximized by $s_1 = 6$, $s_2 = s_3 = 1 + \sqrt{7}$. That gives worst case ratio $R = \frac{37+7\sqrt{7}}{38} \approx 1.461$. The other case is maximized by $s_1 = 2$, $s_2 = s_3 = 1$, with $R = \frac{16}{11} \approx 1.454$. $\square$

### 6.3 Four machines

**Theorem 6.3** *For $m = 4$ and any speeds $s_1 \geq s_2 \geq s_3 \geq s_4$,*

$$r(s_1, s_2, s_3, s_4) = \begin{cases}
\dfrac{S}{s_1 + \alpha s_2 + \alpha^2 s_3 + \alpha^3 s_4} & \text{if } \alpha s_1 \leq s_2 \text{ and } (\alpha+\alpha^2)s_1 \leq s_2 + s_3 \\[2ex]
\dfrac{S^2}{\sum_{i=1}^4 \sum_{j=i}^4 s_i s_j + \alpha(s_3 + s_4)s_4 - s_4^2} & \text{if } \alpha s_1 \geq s_2 \text{ and } s_1(s_3 + s_4) \leq s_3 S \\[2ex]
\dfrac{S^2}{\sum_{i=1}^4 \sum_{j=i}^4 s_i s_j} & \text{if } \alpha s_1 \geq s_2 \text{ and } s_1(s_3 + s_4) \geq s_3 S \\[2ex]
\dfrac{S^2}{(s_1 + \alpha s_2 + \alpha^2 s_3)S + s_4^2} & \text{if } \alpha s_1 \leq s_2 \text{ and } (\alpha+\alpha^2)s_1 \geq s_2 + s_3
\end{cases}$$

*The function $r(s_1, s_2, s_3, s_4)$ has maximal value approximately $1.550$, and thus this is also the optimal competitive ratio for $m = 4$ over all speeds. The maximum is achieved for speed vector approximately $(0.3584, 0.2300, 0.2058, 0.2058)$, and this vector is unique up to scaling.*

The first case is already analyzed in Section 6.1. For the sketch of the proof of the remaining cases, see Appendix B.

The cases are exhaustive, as the first and the last case together cover $\alpha s_1 \geq s_2$ and the middle cases cover $\alpha s_1 \leq s_2$. Also, when $s_1 = \alpha s_2$, then the second conditions in the first two cases turn out to be equivalent.

The maximal value of $r(s_1, s_2, s_3, s_4)$ is achieved at the border between the first two cases, see also Appendix B. Note also that the third case is a natural extension of the second case for three machines.

**Conclusions.** The main open problem is to find better bounds on the overall competitive ratio, and perhaps to find an explicit formula for further special cases. With the knowledge of the optimal algorithm, this "only" involves analyzing the linear program better. In general, the formula for computing the optimal competitive ratio may need to have many cases. Still, it is plausible that a good overall bound can be proved with only a few upper bounds similar to the ones in Section 6.

Another question concerns the idle times in the schedule. Our algorithm relies on creating idle periods on some machines during the schedule. In contrast, all the previously known optimal algorithms for special cases create no "holes". However, we are not able to show whether these idle periods are really necessary to achieve the optimal competitive ratio or not, even for any particular combination of speeds. In particular, we do not know if idle times are necessary for three machines, even in the case which extends Epstein's algorithm for non-decreasing speed ratios where no idle times are needed.

**Acknowledgments.** We are grateful to anonymous referees for many useful comments.

## References

[1] S. Albers. On randomized online scheduling. In *Proc. 34th Symp. Theory of Computing (STOC)*, pages 134–143. ACM, 2002.

[2] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.

[3] B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Inform. Process. Lett.*, 51:219–222, 1994.

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

[5] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 199–210. Springer, 2004.

[6] L. Epstein. Optimal preemptive scheduling on uniform processors with non-decreasing speed ratios. *Oper. Res. Lett.*, 29:93–98, 2001.

[7] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized on-line scheduling for two uniform machines. *J. Sched.*, 4:71–92, 2001.

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

[9] R. Fleischer and M. Wahl. On-line scheduling revisited. *J. Sched.*, 3:343–353, 2000.

[10] T. F. Gonzales and S. Sahni. Preemptive scheduling of uniform processor systems. *J. ACM*, 25:92–101, 1978.

[11] R. Graham, E. Lawler, J. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math*, 5:287–326, 1979.

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

[13] D. S. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17, 1988.

[14] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM*, 34:144–162, 1987.

[15] E. Horwath, E. C. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *J. ACM*, 24:32–43, 1977.

[16] R. McNaughton. Scheduling with deadlines and loss functions. *Management Sci.*, 6:1–12, 1959.

[17] J. F. Rudin III. *Improved Bound for the Online Scheduling Problem.* PhD thesis, The University of Texas at Dallas, 2001.

[18] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Inform. Process. Lett.*, 63:51–55, 1997.

[19] T. Tichý. Randomized on-line scheduling on 3 processors. *Oper. Res. Lett.*, 32:152–158, 2004.

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

## A Numerical lower bounds

In the table below we list the set of speeds $s_i$ and the sequence of values $q_j$ that witness a lower bound on competitive ratio of 2.054, using the linear program (4). The values $O_k$ can be computed directly from $q_j$ and $s_i$. A file with the solutions for $m = 3, \dots, 70$ and $m = 100$ in a format suitable for computer verification is available at `http://math.cas.cz/sgall/ps/optrel/`.

$$
\begin{aligned}
m &= 100 \\
R &= 2.0546772231 \\
\\
s_1 &= 1.0000000000 \\
s_2 &= 0.7659834007 \\
s_3 &= 0.6666239047 \\
s_4 &= 0.5899144526 \\
s_5 &= 0.4904674968 \\
s_6 &= 0.4077851701 \\
s_7 &= 0.3390413148 \\
s_8 = s_9 &= 0.2818862027 \\
s_{10} = s_{11} &= 0.2588311627 \\
s_{12} &= 0.2250940836 \\
s_{13} &= 0.1957544290 \\
s_{14} = s_{15} = s_{16} &= 0.1823338592 \\
s_{17} = s_{18} = s_{19} &= 0.1759567132 \\
s_{20} &= 0.1562503561 \\
s_{21} = s_{22} = s_{23} = s_{24} &= 0.1489918576 \\
s_{25} &= 0.1489918632 \\
s_{26} &= 0.1411121793 \\
s_{27} = \ldots = s_{31} &= 0.1269310036 \\
s_{32} &= 0.1234155094 \\
s_{33} &= 0.1128943928 \\
s_{34} = s_{35} = s_{36} = s_{37} &= 0.1112466212 \\
s_{38} &= 0.1031563909 \\
s_{39} = \ldots = s_{47} &= 0.0977740900 \\
s_{48} &= 0.0958859454 \\
s_{49} &= 0.0917230259 \\
s_{50} = \ldots = s_{100} &= 0.0840900444 \\
\\
q_1 &= 0.1127872610 \\
q_2 &= 0.0026172659 \\
q_3 = q_4 &= 0.0027483680 \\
q_5 &= 0.0028129098 \\
q_6 = q_7 &= 0.0031721583 \\
q_8 &= 0.0032527701 \\
q_9 &= 0.0033327815 \\
q_{10} &= 0.0034657286 \\
q_{11} &= 0.0038636996 \\
q_{12} &= 0.0038916544 \\
q_{13} = q_{14} &= 0.0039963547 \\
q_{15} &= 0.0045675782 \\
q_{16} &= 0.0047388936 \\
q_{17} &= 0.0049106439 \\
q_{18} &= 0.0050022044
\end{aligned}
$$

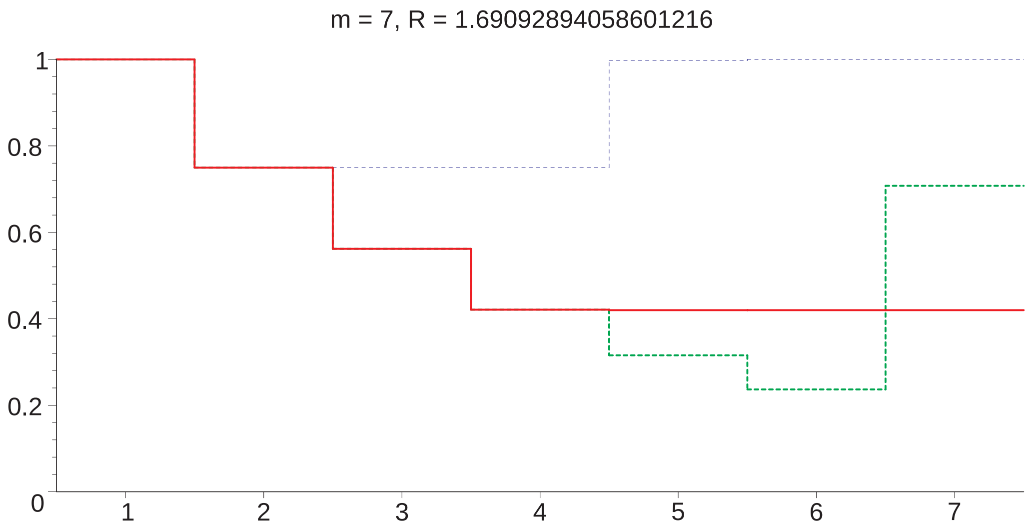
$$
\begin{aligned}
q_{19} &= 0.0050550746 \\
q_{20} &= 0.0051466351 \\
q_{21} &= 0.0051995053 \\
q_{22} &= 0.0052910658 \\
q_{23} = q_{24} = q_{25} &= 0.0054482328 \\
q_{26} = q_{27} &= 0.0056362745 \\
q_{28} = q_{29} = q_{30} &= 0.0058587447 \\
q_{31} = q_{32} = q_{33} = q_{34} &= 0.0061230671 \\
q_{35} = q_{36} = q_{37} = q_{38} &= 0.0063914475 \\
q_{39} = q_{40} = q_{41} &= 0.0066927997 \\
q_{42} = q_{43} &= 0.0069965575 \\
q_{44} &= 0.0076316457 \\
q_{45} &= 0.0079780138 \\
q_{46} = q_{47} &= 0.0081351134 \\
q_{48} = \ldots = q_{52} &= 0.0085782005 \\
q_{53} = q_{54} = q_{55} = q_{56} &= 0.0091448116 \\
q_{57} &= 0.0096482182 \\
q_{58} &= 0.0104048975 \\
q_{59} = q_{60} = q_{61} = q_{62} &= 0.0112209208 \\
q_{63} &= 0.0113871238 \\
q_{64} &= 0.0124483391 \\
q_{65} &= 0.0136084536 \\
q_{66} = \ldots = q_{70} &= 0.0139960908 \\
q_{71} &= 0.0155597830 \\
q_{72} &= 0.0172981764 \\
q_{73} &= 0.0182641041 \\
q_{74} = q_{75} = q_{76} = q_{77} &= 0.0182641034 \\
q_{78} &= 0.0191538833 \\
q_{79} &= 0.0215695787 \\
q_{80} = q_{81} = q_{82} &= 0.0242899426 \\
q_{83} = q_{84} = q_{85} &= 0.0251702756 \\
q_{86} &= 0.0270229180 \\
q_{87} &= 0.0310731103 \\
q_{88} &= 0.0357303451 \\
q_{89} = q_{90} &= 0.0410856057 \\
q_{91} = q_{92} &= 0.0447452511 \\
q_{93} &= 0.0538177769 \\
q_{94} &= 0.0647298436 \\
q_{95} &= 0.0778544359 \\
q_{96} &= 0.0936401642 \\
q_{97} &= 0.1126265992 \\
q_{98} &= 0.1272719849 \\
q_{99} &= 0.1462417221 \\
q_{100} &= 0.1909202236
\end{aligned}
$$

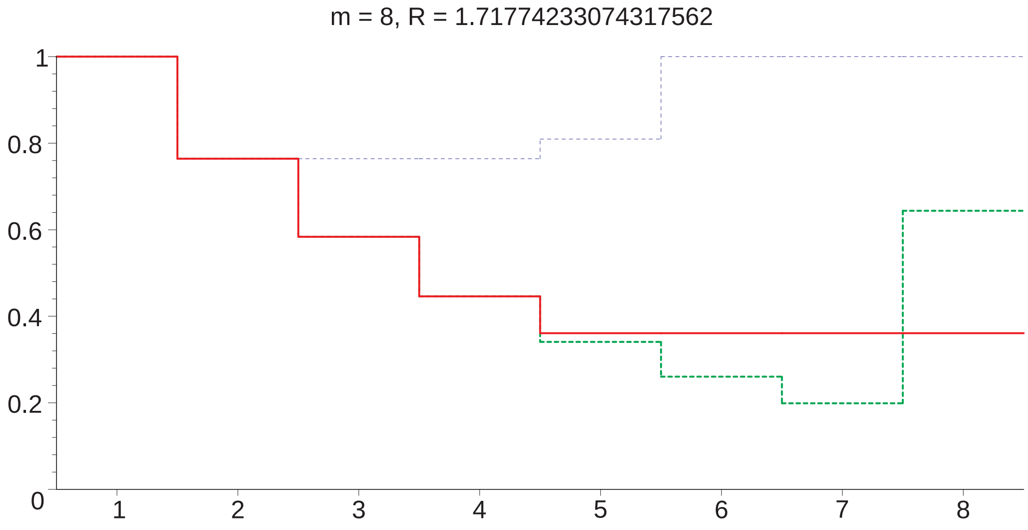
In the figure below we show graphical representation of the set of speeds $s_i$ and of the sequence of values $q_j$ that witness a lower bound on competitive ratio of 2.054, using the linear program (4). The values $O_k$ can be computed directly from $q_j$ and $s_i$. A file with the solutions for $m = 3, \ldots, 70$ and $m = 100$ which is in a format suitable for computer verification is available at `http://math.cas.cz/sgall/ps/optrel/`.

The following figures show some of the sets of speeds with corresponding sets of jobs that witness our currently best lower bounds for each value of $m$. For each $m$, the solution was found by Maple as a local optimum of the quadratic program.

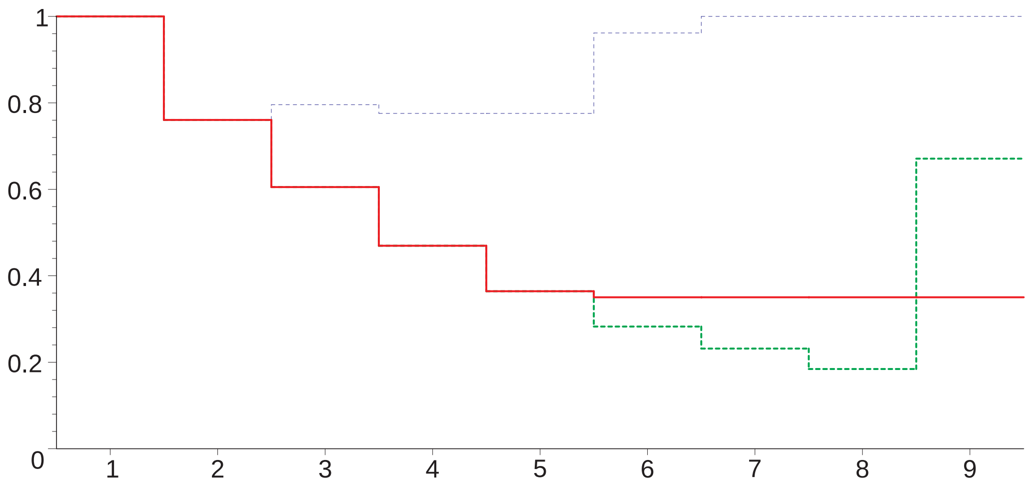
The red bold curve shows speeds $s_i$, green dashed bold curve (bottom one) shows scaled job variables, and the thin dashed blue curve (top one) shows inverses of the speed ratios, i.e., $s_{i-1}/s_i$.

The values of $q_j$ are printed in reverse order, i.e., column $i$ shows $q_{m-i+1}$. The values are scaled so that $q_m = 1 = s_1$. This ordering and scaling shows best the relations between job sizes and speeds.

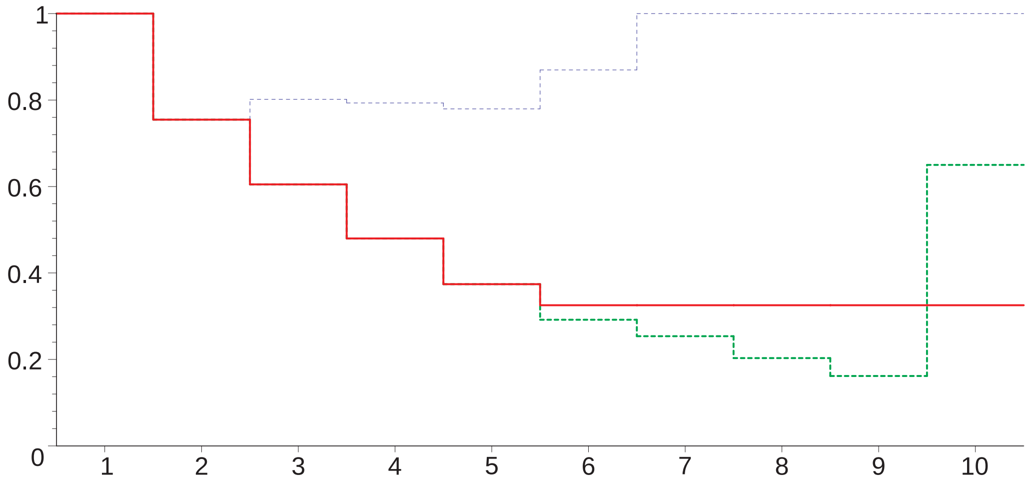


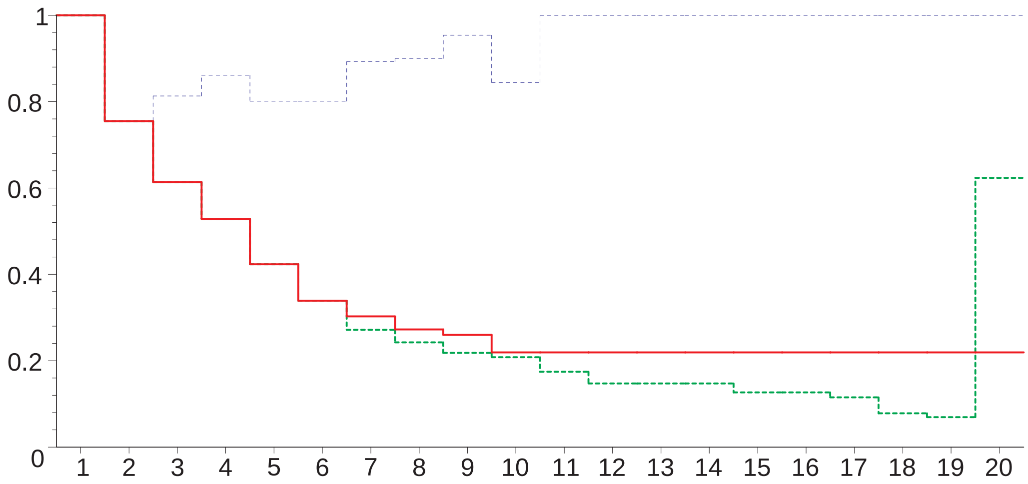
m = 9, R = 1.74070209102756923

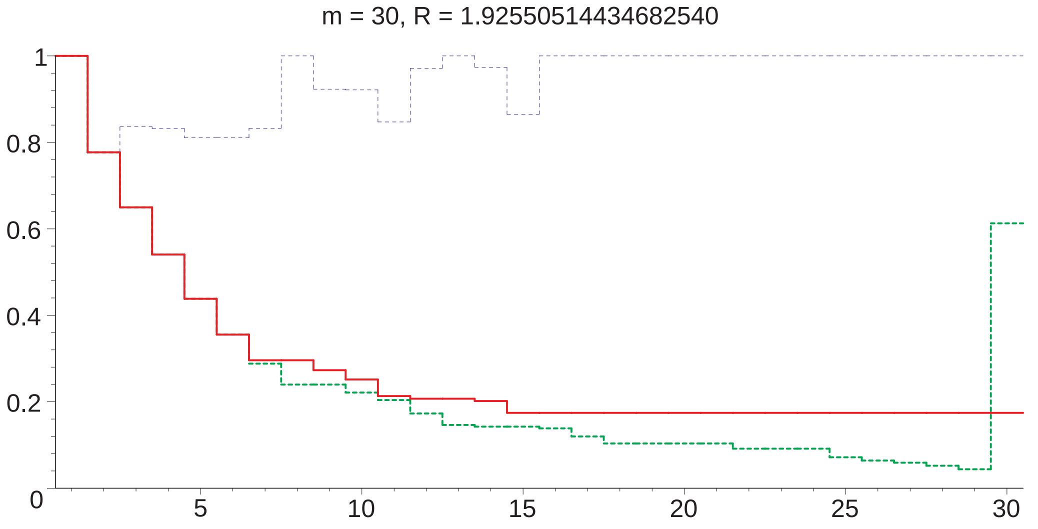


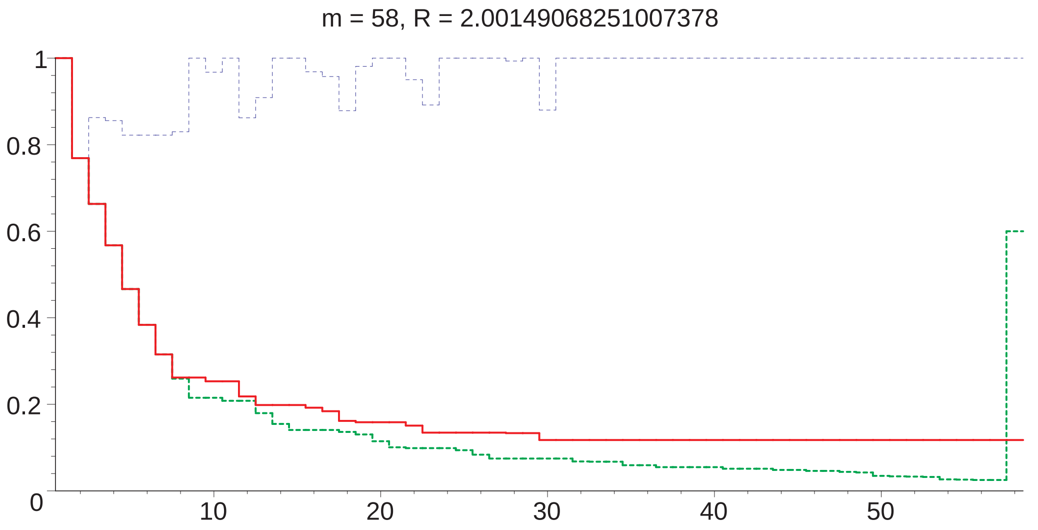

m = 10, R = 1.76082012794551313

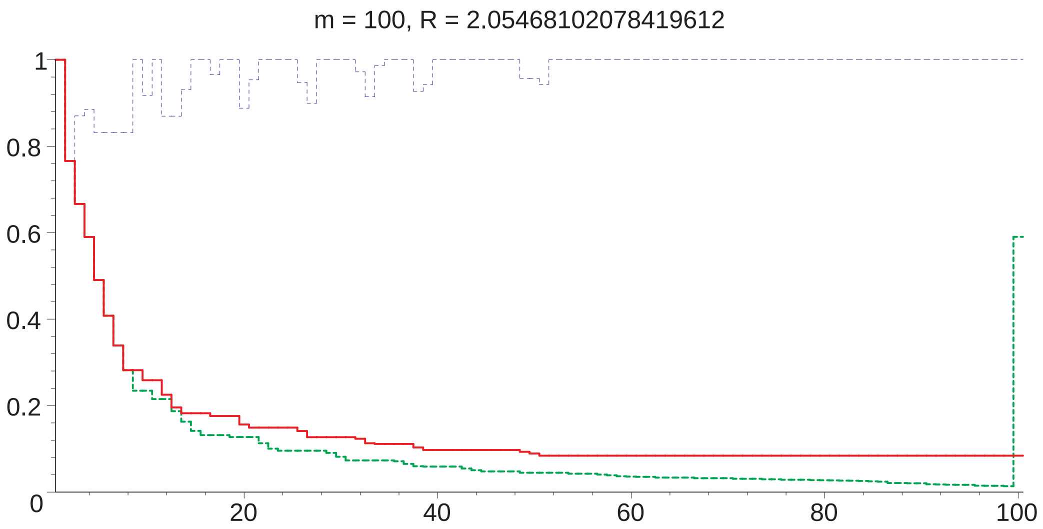



m = 20, R = 1.87132273399129545




m = 30, R = 1.92550514434682540
1
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30


m = 58, R = 2.00149068251007378
1
0.8
0.6
0.4
0.2
0
10
20
30
40
50


m = 100, R = 2.05468102078419612
1
0.8
0.6
0.4
0.2
0
20
40
60
80
100

## B Four machines

We prove the remaining three cases now by giving an explicit solution of the primal linear program for the lower bounds and a dual solution in the form of a non-negative linear combination of the constraints for the upper bounds on the competitive ratio.

**Case II.** We have

$$\begin{aligned} \alpha s_1 &\geq s_2 && (11)\\ s_1(s_3+s_4) &\leq s_3 S\,, && (12) \end{aligned}$$

and we put

$$R = \frac{S^2}{\left(\sum_{i=1}^4 \sum_{j=i}^4 s_i s_j\right) + \alpha(s_3+s_4)s_4 - s_4^2}$$

To prove that $R$ is a lower bound on the competitive ratio, we give a solution of the primal program:

$$\begin{aligned} q_1 &= \alpha(s_3+s_4)R/S & O_1 &= \alpha(s_3+s_4)R/S^2\\ q_2 &= s_1(s_3+s_4)R/S^2 & O_2 &= (s_3+s_4)R/S^2\\ q_3 &= s_2 R/S & O_3 &= \alpha R/S\\ q_4 &= s_1 R/S & O_4 &= R/S \end{aligned}$$

It is easy to check that in the following constraints equality holds:

$$\begin{aligned} q_1+\ldots+q_k &\leq SO_k && \text{for } k=1,2,3,4\\ q_2 &\leq s_1 O_2\\ q_2+q_3 &\leq (s_1+s_2)O_3\\ q_3+q_4 &\leq (s_1+s_2)O_4\\ q_4 &\leq s_1 O_4\\ 1 &= s_4 O_1 + s_3 O_2 + s_2 O_3 + s_1 O_4 \end{aligned}$$

Thus we also have the objective equal to $q_1+q_2+q_3+q_4 = SO_4 = R$. The remaining two constraints are actually equivalent to the case conditions:

$$\begin{aligned} q_2+q_3+q_4 &\leq (s_1+s_2+s_3)O_4 && \text{This is equivalent to (12).}\\ q_3 &\leq s_1 O_3 && \text{This is equivalent to (11).} \end{aligned}$$

To prove that $R$ is an upper bound on the competitive ratio, we sum

$$\begin{aligned} q_1 &\leq SO_1 && \text{times} && s_4 S\\ q_1+q_2 &\leq SO_2 && \text{times} && s_3 S - s_1 s_4\\ q_1+q_2+q_3 &\leq SO_3 && \text{times} && s_2 S\\ q_1+q_2+q_3+q_4 &\leq SO_4 && \text{times} && s_1^2 + s_1 s_4 - s_2 s_3 - \alpha s_4(s_1+s_2)\\ q_2 &\leq s_1 O_2 && \text{times} && s_4 S\\ q_3+q_4 &\leq (s_1+s_2)O_4 && \text{times} && (s_3+\alpha s_4)S\\ q_4 &\leq s_1 O_4 && \text{times} && s_2 S \end{aligned}$$

We first verify that all the coefficients are non-negative. The second coefficient is non-negative as $S \geq s_1$ and $s_3 \geq s_4$. The middle coefficient is non-negative, since, using (11) in the first inequality, we have

$$\begin{aligned}
s_1^2 + s_1 s_4 - s_2 s_3 - \alpha s_4(s_1 + s_2) &= (1-\alpha)(s_1^2 + s_1 s_4 + s_2 s_4) + \alpha s_1^2 - s_2(s_3 + s_4) \\
&\geq (1-\alpha)(s_2 + s_3 + s_4)s_4 + \alpha s_1^2 - \alpha s_1(s_3 + s_4) \\
&= \alpha s_1(s_1 - s_3) \;\geq\; 0
\end{aligned}$$

After some calculation, it can be verified that resulting inequality is

$$(q_1 + q_2 + q_3 + q_4)S^2/R \;\leq\; (s_1 O_4 + s_2 O_3 + s_3 O_2 + s_4 O_1)S^2 = S^2\,.$$

Thus we obtain $r(s_1, s_2, s_3, s_4) \leq R$, which completes the case.

**Case III.** We have

$$\alpha s_1 \;\geq\; s_2 \tag{13}$$

$$s_1(s_3 + s_4) \;\geq\; s_3 S \tag{14}$$

and put

$$R \;=\; \frac{S^2}{\sum_{i=1}^{4}\sum_{j=i}^{4} s_i s_j}$$

For the lower bound, we use the primal solution

$$\begin{aligned}
q_k + \ldots + q_4 &\leq (s_1 + \ldots + s_{5-k})O_4 && \text{for } k = 1,2,3,4, \\
O_k &= (s_{5-k} + \ldots + s_4)R/S^2 && \text{for } k = 1,2,3,4.
\end{aligned}$$

In the following constraints equality holds:

$$\begin{aligned}
q_1 + \ldots + q_k &\leq SO_k && \text{for } k = 1,2,3,4 \\
q_k + \ldots + q_4 &\leq (s_1 + \ldots + s_{5-k})O_4 && \text{for } k = 2,3,4 \\
1 &= s_4 O_1 + s_3 O_2 + s_2 O_3 + s_1 O_4
\end{aligned}$$

Thus also $q_1 + \ldots + q_4 = SO_4 = R$. The constraints $q_2 \leq s_1 O_2$ and $q_2 + q_3 \leq (s_1 + s_2)O_3$ are both equivalent to (14). The last constraint $q_3 \leq s_1 O_3$ is equivalent to (13).

For the upper bound, we sum

$$\begin{aligned}
q_1 + \ldots + q_k &\leq SO_k && \text{times} \quad s_{5-k} && \text{for } k = 1,2,3 \\
q_1 + \ldots + q_4 &\leq SO_4 && \text{times} \quad s_1 - S + S/R \\
q_k + \ldots + q_4 &\leq (s_1 + \ldots + s_{5-k})O_4 && \text{times} \quad s_{6-k} && \text{for } k = 2,3,4.
\end{aligned}$$

From (14) and $s_3 \geq s_4$ we have $s_1 \geq S/2$. It is easy to see that $R \leq 2$, so the middle coefficient is non-negative. The resulting inequality is

$$(q_1 + \ldots + q_4)S/R \leq (s_m O_1 + \ldots + s_1 O_m)S,$$

which gives the desired bound.

**Case IV.** We have

$$s_1\alpha \leq s_2 \tag{15}$$
$$s_1(\alpha+\alpha^2) \geq s_2+s_3 \tag{16}$$

and we put

$$R = \frac{S^2}{(s_1+\alpha s_2+\alpha^2 s_3)S+s_4^2}.$$

To prove the lower bound, we use the primal solution

$$\begin{array}{lllllll} q_1 & = & s_4R/S & \quad & O_1 & = & s_4R/S^2 \\ q_2 & = & (\alpha(s_2+s_3+s_4)-s_4)R/S & & O_2 & = & \alpha^2R/S \\ q_3 & = & \alpha s_1R/S & & O_3 & = & \alpha R/S \\ q_4 & = & s_1R/S & & O_4 & = & R/S \end{array}$$

We prove that $q_2$ is non-negative. The inequality is due to (16):

$$\begin{aligned} q_2S^2/R &= (s_2+s_3)S - s_1(s_2+s_3+s_4) \\ &\geq s_1(\alpha+\alpha^2)S - s_1\alpha S \geq 0 \end{aligned}$$

In the following constraints equality holds:

$$\begin{array}{rcll} q_1+\ldots+q_k & \leq & SO_i & \text{for } k=1,2,3,4 \\ q_2+q_3+q_4 & \leq & (s_1+s_2+s_3)O_4 & \\ q_3 & \leq & s_1O_3 & \\ q_4 & \leq & s_1O_4 & \\ 1 & = & s_4O_1+s_3O_2+s_2O_3+s_1O_4 & \end{array}$$

Thus we also have $q_1+q_2+q_3+q_4 = SO_4 = R$. The following constraints are equivalent to the case conditions:

$$\begin{array}{rcll} q_2 & \leq & s_1O_2 & \text{This is equivalent to (16).} \\ q_3+q_4 & \leq & (s_1+s_2)O_4 & \text{This is equivalent to (15).} \end{array}$$

It remains to verify the last constraint

$$q_2+q_3 \leq (s_1+s_2)O_3.$$

This constraint is equivalent to

$$s_2+s_3 \leq (s_1+s_2)\alpha\,. \tag{17}$$

This follows from (16) and (15), as we have $s_2+s_3 \leq (\alpha+\alpha^2)s_1 \leq (s_1+s_2)\alpha$.

Later, we will use another form of (17), which can be derived by multiplying by $S$ and subtracting the same terms from both sides:

$$s_3(s_2+s_3+s_4) \leq s_1s_4\,. \tag{18}$$

For the optimality (dual program), we sum:

$$\begin{array}{rcll}
q_1 & \le & SO_1 & \text{times} \quad s_4 S \\
q_1+q_2 & \le & SO_2 & \text{times} \quad s_3 S \\
q_1+q_2+q_3 & \le & SO_3 & \text{times} \quad s_2 S - s_3 s_1 \\
q_1+q_2+q_3+q_4 & \le & SO_4 & \text{times} \quad s_1 S - (s_1+s_2+s_3)s_4 - s_1(s_2+\alpha s_3) \\
q_2+q_3+q_4 & \le & (s_1+s_2+s_3)O_4 & \text{times} \quad s_4 S \\
q_3 & \le & s_1 O_3 & \text{times} \quad s_3 S \\
q_4 & \le & s_1 O_4 & \text{times} \quad S(s_2+\alpha s_3)
\end{array}$$

The third coefficient is non-negative since $s_2 \ge s_3$ and $S \ge s_1$. We prove that the middle coefficient is non-negative; we use (18) in the second inequality below:

$$\begin{aligned}
s_1 S - (s_1+s_2+s_3)s_4 - s_1(s_2+\alpha s_3) &= s_1^2 + (1-\alpha)s_1 s_3 - (s_2+s_3)s_4 \\
&\ge s_1^2 - (s_2+s_3+s_4)s_3 \\
&\ge s_1^2 - s_1 s_4 \ \ge\ 0
\end{aligned}$$

The resulting inequality is

$$(q_1+q_2+q_3+q_4)S^2/R \ \le\ (s_1O_4 + s_2O_3 + s_3O_2 + s_4O_1)S^2\,,$$

and this proves the upper bound and completes all the cases.

**The overall ratio** Given the explicit formula, it is possible to find the global maximum of the competitive ratio. It turns out that for the maximum we have $s_3 = s_4$, $s_1 = \alpha s_2$, and that the maximum lies on the border between Case I and Case II; we sketch the proof below.

After substitution of these equalities we get

$$R^{-1} = \frac{1}{2}\alpha^5 + \frac{1}{2}\alpha^4 - \alpha^3 + \alpha^2 - \alpha + 1.$$

Numerical optimization by Maple gives the unique maximum at $R = 1.5503$, $\alpha = 0.6416$, which corresponds to the speed vector given in the theorem.

To prove the properties of the maximum claimed above, we show that in all the cases but Case II we can move the speed vector towards the case boundary while increasing $r(s_1, s_2, s_3, s_4)$. Then we optimize Case II using standard calculus. As a technicality, note that $r(s_1, s_2, s_3, s_4)$ is continuous and bounded, and the domain is essentially compact (we may assume that $s_i$ are bounded because of scaling), so there exists a maximum on any closed subdomain.

**Case III.** We claim that the maximum of $r(s_1, s_2, s_3, s_4)$ over sets of speeds satisfying conditions of Case III is equal to 1.5.

Suppose that the maximum is achieved at some point $(s_1, s_2, s_3, s_4)$. First we claim that $s_2 = s_3 = s_4$. Suppose not. If $s_3 > s_4$ then for small $\varepsilon > 0$ the point $(s_1, s_2, s_3 - \varepsilon, s_4 + \varepsilon)$ is a valid speed vector still in Case III, as both case conditions are maintained. It is easy to check that $r(s_1, s_2, s_3 - \varepsilon, s_4 + \varepsilon) > r(s_1, s_2, s_3, s_4)$, contradicting the maximality of $r(s_1, s_2, s_3, s_4)$. If $s_2 > s_3 = s_4$ then for small $\varepsilon > 0$, we use the point $(s_1, s_2 - 2\varepsilon, s_3 + \varepsilon, s_4 + \varepsilon)$. The value of $r$ increases and case conditions are maintained as well.

Having $s_2 = s_3 = s_4$, (14) is equivalent to $s_1 \ge S/2$ and if this is true, then (13) is valid. If $s_1 > S/2$ the we can take the point $(s_1 - 3\varepsilon, s_2 + \varepsilon, s_3 + \varepsilon, s_4 + \varepsilon)$ and the value of $r$ increases. This implies that the maximum in Case III is $r(S/2, S/6, S/6, S/6) = 1.5$ for any $S$.

**Case IV.** We claim that the maximum of $r(s_1, s_2, s_3, s_4)$ over Case IV is achieved on a point for which in (15) equality holds, i.e., on a boundary with Case III.

Suppose for a contradiction that the maximum is achieved at some other point $(s_1, s_2, s_3, s_4)$. The case conditions imply that $s_2 > s_3$, as otherwise $2s_2 = s_2 + s_3 \le (\alpha + \alpha^2)s_1 \le (1+\alpha)s_2$, contradicting $\alpha < 1$. We take the point $(s_1, s_2 - \varepsilon, s_3 + \varepsilon, s_4)$ for small $\varepsilon > 0$. The value of $r$ increases, while (16) is maintained.

**Case I.** We claim that the maximum of $r(s_1, s_2, s_3, s_4)$ over Case I is achieved on a point for which in $\alpha s_1 = s_2$, i.e., on a boundary with Case II.

Suppose for a contradiction that the maximum is achieved at some other point $(s_1, s_2, s_3, s_4)$.

If $s_2 > s_3$ then we take the point $(s_1, s_2 - \varepsilon, s_3 + \varepsilon, s_4)$. The value of $r$ increases and the second case condition is maintained.

If $s_2 = s_3 > s_4$ then we take $(s_1, s_2 - \varepsilon, s_3 - \varepsilon, s_4 + 2\varepsilon)$. The value of $r$ increases and the second case condition is maintained, since it is implied by $\alpha s_1 \le s_2$ and $s_2 = s_3$.

If $s_2 = s_3 = s_4$ then $r(1-\alpha, \alpha/3, \alpha/3, \alpha/3)^{-1} = 1 - \alpha + \alpha^2/3 + \alpha^3/3 + \alpha^4/3$. The case condition $\alpha s_1 \le s_2$ implies that $\alpha \ge 2/3$, and for such $\alpha$ this polynomial is increasing. Thus in this subcase $r$ is maximized for $\alpha = 2/3$, which satisfies $\alpha s_1 = s_2$.

**Case II.** If in (12) equality holds, then we are in the intersection with Case III and the value of $r$ is at most 1.5, which we know is not a global maximum. Thus for the rest of the proof we assume that in (12) we have sharp inequality.

Assume that the maximum of the case is achieved at some point $(s_1, s_2, s_3, s_4)$ with $s_3 > s_4$. We take the point $(s_1, s_2, s_3 - \varepsilon, s_4 + \varepsilon)$. The value of $r$ increases and (11) is maintained. We conclude that any maximum has $s_3 = s_4$.

Suppose now that the maximum is achieved at some point $(s_1, s_2, s_3, s_4)$ not on the boundary of the case. We scale the point so that $S = 1$ and express $r(s_1, s_2, s_3, s_4)^{-1}$ in terms of $s_2$ and $s_3$, using substitutions $\alpha = s_2 + 2s_3$, $s_1 = 1 - s_2 - 2s_3$, and $s_4 = s_3$. We obtain

$$r(s_1, s_2, s_3, s_4)^{-1} = 1 - s_2 - 2s_3 + s_2^2 + 2s_2s_3 + 2s_3^2 + 2s_2s_3^2 + 4s_3^3.$$

In any maximum not on the boundary, both derivatives of this polynomial must be 0. Taking the derivative in $s_2$, we obtain $s_2 = 1/2 - s_3 - s_3^2$. After this substitution, we get a polynomial whose minimum is at $s_3 \approx 0.198$ and the resulting speed vector turns out to be out of the feasible range of Case II. Thus any maximum is on the boundary of Case II. Since we have excluded the boundary with Case III, it must be on the boundary with case I and satisfy $s_1 = \alpha s_2$.