Minimizing the Maximum Starting Time On-line[*]

Leah Epstein[†] Rob van Stee[‡]

Abstract

We study the scheduling problem of minimizing the maximum starting time on-line. The goal is to minimize the last time that a job starts. We show that while the greedy algorithm has a competitive ratio of $\Theta(\log m)$, we can give a constant competitive algorithm for this problem. We also show that the greedy algorithm is optimal for resource augmentation in the sense that it requires $2m-1$ machines to have a competitive ratio of 1, whereas no algorithm can achieve this with $2m-2$ machines.

1 Introduction

In this paper, we study on-line multiprocessor scheduling with a new objective function: the *maximum starting time*. Jobs arrive on-line to be scheduled on m parallel machines. These machines can be either identical or *related*, in which case each machine has a speed that determines how long it takes to run one unit of work.

We study the on-line paradigm where jobs arrive one by one. A job J_j is defined by its size and by its order in the input sequence. Denote the starting time of job J_j by S_j. We denote the cost of an algorithm $\mathcal{A}$ on a job sequence $\sigma = \{J_1, \ldots, J_n\}$ by $\mathcal{A}(\sigma) = \max_j S_j$. An algorithm is required to run the jobs on each machine in the order of arrival.

An example of this situation is the following. There is a loading station where trucks are loaded with goods. These goods need to be delivered to different places, after which the trucks return to the loading station to pick up a new load. At the end of a work day, the station can close as soon as the truck carrying the last load has left, and does not need to wait for the trucks to return. The time it takes to deliver the goods in one truck is the size of the job. (Here we consider a truck load to be "one job", e. g. each truck contains only one item, or items for only one destination (client).)

[*]A preliminary version of this paper appeared in Proceedings of the 10th European Symposium on Algorithms, 2002.

[†]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. `lea@idc.ac.il`. Research supported by Israel Science Foundation (grant no. 250/01).

[‡]Institut für Informatik, Albert-Ludwigs-Universität, Georges-Köhler-Allee, 79110 Freiburg, Germany. `vanstee@informatik.uni-freiburg.de`. This work done while the author was at the CWI, The Netherlands. Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002.

We use two measures to study the performance of on-line algorithms. The *competitive ratio* compares an on-line algorithm to an optimal off-line algorithm OPT that knows the job sequence in advance (but can not change the order in which jobs run on a machine, i.e. it also has to run jobs on a machine in the order of their arrival). The competitive ratio $\mathcal{R}(\mathcal{A})$ of an on-line algorithm $\mathcal{A}$ is the infimum value of $\mathcal{R}$ such that for every sequence σ,

$$\mathcal{A}(\sigma) \leq \mathcal{R} \cdot \text{OPT}(\sigma) \ . \tag{1}$$

The second measure involves *resource augmentation.* Assume the on-line algorithm uses $\tilde{m}$ machines, where $\tilde{m} > m$. What is the minimum value of $\tilde{m}$ such that the cost of the on-line algorithm is bounded by the cost of the optimal off-line algorithm (i.e. the competitive ratio is at most 1)? Resource augmentation was originally introduced by [11], and further widely studied for various scheduling and load balancing problems [5, 7, 11, 12, 14].

Note that if a sequence σ contains at most m jobs, then $\text{OPT}(\sigma) = 0$. By (1), any algorithm with finite competitive ratio needs to have zero cost and run all jobs on different machines in that case.

All previous work assumed that the output needs to be collected by the same system, and hence the last *completion* time was considered in numerous papers [10, 4, 13, 1, 9, 8]. Other papers also considered different functions of the completion times [2] but never the starting times. Resource augmentation for scheduling of jobs one by one was also considered with the maximum completion time goal function [6, 3]. However, to the best of our knowledge, no previous work on the above goal function exists.

We show the following results for the competitive ratio on identical machines:

- The greedy algorithm, which assigns each job to the least loaded machine, has competitive ratio $\Theta(\log m)$.
- The greedy algorithm has optimal competitive ratios for 2 and 3 machines, which are 2 and 5/2 respectively.
- There exists a constant competitive algorithm BALANCE which has competitive ratio 12 for any m (hence the greedy algorithm is far from having optimal competitive ratio for general m).
- No deterministic algorithm for general m has competitive ratio smaller than 4.

For two related machines, we give a tight bound of $q + 1$ for the competitive ratio, where q is the speed of the fastest machine relative to the slowest.

We show the following results for resource augmentation on identical machines:

- The greedy algorithm has competitive ratio 1 if it uses $2m - 1$ machines (and is compared to an optimal off-line algorithm with m machines).

- Any on-line algorithm which uses $2m-2$ machines has competitive ratio larger than 1, and any on-line algorithm which uses $2m-1$ machines has competitive ratio of at least 1. Hence the greedy algorithm is optimal in this measure.

Note that the off-line version of minimizing the maximum starting time is strongly NP-hard. The off-line problem of minimizing the maximum completion time (minimizing the makespan) is a special case of our problem. A simple reduction from the makespan problem to our problem can be given by adding m very large jobs (larger than the sum of all other jobs) in the end of the sequence. Each machine is forced to have one such job, and the maximum starting time of the large jobs, is the makespan of the original sequence.

We present results on the greedy algorithm in Section 2, the constant competitive algorithm BALANCE in Section 3, lower bounds in Section 4 and results for resource augmentation in Section 6.

2 The Greedy Algorithm

GREEDY always assigns an arriving job on the machine where it can start the earliest (see [10]). In some upper bound proofs we use the following definition: a *final* job is a job that starts as the last job on some machine in OPT's schedule.

Theorem 1 $\mathcal{R}(\text{GREEDY}) = \Theta(\log m)$ *on identical machines.*

Proof. Let $\lambda = \text{OPT}(\sigma)$. Note that all on-line machines are occupied until time $\text{GREEDY}(\sigma)$. We cut the schedule of GREEDY into pieces of time length 2λ starting from the bottom.

If there are less than m final jobs, there are less than m jobs, hence GREEDY is optimal. Suppose there are m final jobs.

Claim: At time $2i\lambda$, at most $m/2^i$ final jobs did not start yet.

Proof: By induction. The claim holds for $i = 0$. Assume it holds for some $i \geq 0$.

A final job is called *missing* if it did not start before time $2\lambda i$. Let k be the number of missing jobs. We have $k \leq m/2^i$ starting at time $2\lambda i$ or later. The total size of non-final jobs running at any time after $2\lambda i$ is at most $k\lambda$. This follows because GREEDY schedules the jobs with monotonically increasing start times, hence if there are k missing final jobs, then all the unstarted jobs must have arrived after the $m-k$-th final job. That job is started before time $2\lambda_i$ and hence the unstarted jobs must be scheduled by OPT on the machines where it runs the last k final jobs. Since OPT completes all these (non-final) jobs no later than at time λ, the total size of these jobs is at most $k\lambda$.

At most $k/2$ machines can be busy with these jobs during the entire time interval $[2\lambda i, 2\lambda(i+1)]$. Hence $k/2$ or more final jobs start in this interval (one for every machine that is not busy with non-final jobs during the entire interval and that was also not running a final job already). At most $k/2$ final jobs will be missing at time $2\lambda(i+1)$, and $k/2 \leq m/2^{i+1}$. □

At time $2\lambda \log_2 m$, only one final job is missing, therefore $\textsc{Greedy}(\sigma) \leq 2\lambda \log_2 m + \lambda$, hence $\mathcal{R}(\textsc{Greedy}) = O(\log m)$.

To show that $\mathcal{R}(\textsc{Greedy}) = \Omega(\log m)$, we use a job sequence that consists of a job of size 1 followed by a job of size M (a large constant, e.g. $M = m$), repeated m times. The optimal algorithm can assign the jobs so that no job starts later than at time 1, whereas Greedy starts the last job at time $1 + \lfloor \log_2 m \rfloor$. □

We now consider the competitive ratio of Greedy for $m = 2, 3$. In Section 4, we will show matching lower bounds. Hence, Greedy is optimal for $m = 2, 3$.

Lemma 1 *On identical machines,* $\mathcal{R}(\textsc{Greedy}) \leq 2$ *for* $m = 2$*, and we have* $\mathcal{R}(\textsc{Greedy}) \leq 5/2$ *for* $m = 3$.

Proof. We start with the case $m = 2$. We need to show that the competitive ratio of Greedy is at most 2. Assume by contradiction that Greedy has competitive ratio of $\rho > 2$. Define $\varepsilon = \frac{1}{2}(\rho - 2)$ and consider a sequence σ for which Greedy has a ratio of at least $\rho - \varepsilon$. Without loss of generality we assume that $\textsc{opt}(\sigma) = 1$. We denote the last job in σ by J_ℓ. This is a final job.

Since J_ℓ was assigned by Greedy to the least loaded machine, both of the on-line machines are busy until time $\rho - \varepsilon$. Hence the total size of all jobs but J_ℓ is at least $2(\rho - \varepsilon) > 4$. The volume of jobs that opt runs before time $\textsc{opt}(\sigma) = 1$ is at most 2. opt can run only two additional (final) jobs after time 1, one on each machine. One of those jobs is J_ℓ. Hence the other job, J_0, must have a size greater than $2(\rho - \varepsilon) - 2 > 2$.

Hence there exists a job J_0 of size greater than 2. The volume of the remaining jobs (apart from J_ℓ) is at most 2. Hence Greedy will not schedule J_ℓ on the same machine as J_0, because the other machine must be less loaded. Scheduled on that machine, J_ℓ starts no later than at time 2, since at most a volume of 2 of jobs is scheduled before it.

For $m = 3$, suppose Greedy has competitive ratio $\rho > 5/2$ and define σ and J_ℓ as above (taking $\varepsilon = \frac{1}{2}(\rho - 5/2)$). Assume $\textsc{opt}(\sigma) = 1$. Denote the total size of all jobs but J_ℓ by V. Note that the size of J_ℓ is irrelevant for the competitive ratio; we may assume it has size 0. Denote the total size of all jobs of size at most 1 by V'. Since $\textsc{opt}(\sigma) = 1$, opt starts all its jobs no later than at time 1; the jobs that it completes before time 1 have total size at most 3.

We have $V \geq 3(\rho - \varepsilon) > 15/2$, since all three of Greedy's machines are busy until past time $\rho - \varepsilon > 5/2$ when J_ℓ arrives.

- If σ contains no jobs larger than 1, consider the optimal off-line schedule. Two final jobs are of size at most 1, and the third (J_ℓ) is of size 0. The rest of the jobs are completed by time 1, and their total size is at most 3. Hence $V = V' \leq 5$, a contradiction.
- If σ contains one job larger than 1, then $V' \leq 4$: one final job has size 0, and one must have size at most 1 (since only one can be larger than 1). The rest of the jobs are of size at most 1, and have total size at most 3. Consider the

least loaded machine among the two machines that do not run the job larger than 1, at the time J_ℓ arrives. Since $V' \leq 4$, it cannot have a load more than 2. But then GREEDY starts J_ℓ no later than at time 2.

- If σ contains two jobs larger than 1, then analogously to the previous cases, $V' \leq 3$. Denote the time that GREEDY starts the second large job by t_2. Similarly to in the previous case, we have $t_2 \leq 3/2 < 5/2$. At most a volume of 1 of jobs starts after t_2, since OPT has to run all these jobs and J_ℓ on one machine if $\text{OPT}(\sigma) = 1$: two of OPT's machines are already running large jobs and cannot be used anymore.
  - If $t_2 \geq 1/2$, then in the worst case GREEDY assigns all the jobs that arrive after t_2 to one machine and starts J_ℓ no later than at time $5/2$.
  - If $t_2 < 1/2$, then at the time the second large job arrives GREEDY starts no job later than at time $1/2$. Hence the on-line machine that has no large job has load at most $3/2$ at this time, since all jobs on that machine have size at most 1 and GREEDY always uses the least loaded machine. Since after t_2, at most a volume 1 of jobs still arrives, J_ℓ starts no later than at time $5/2$. □

We now turn to the performance of GREEDY on related machines. We set the speed of the slowest machine to 1 and denote the speed of the fastest machine by $q > 1$. On the fastest machine, it takes w/q time to complete a job of size w.

Lemma 2 *For two related machines, $\mathcal{R}(\text{GREEDY}) \leq q + 1$.*

Proof. Suppose the competitive ratio of GREEDY is $\rho > 1 + q$. We define J_ℓ and σ analogously to in Lemma 1, taking $\varepsilon = \frac{1}{2}(\rho - q - 1)$. In the present case, we find that the total size of all jobs but J_ℓ must be greater than $(q+1)^2$, OPT can run at most $1 + q$ before time 1 and there must be a job J_0 of size greater than $q(1+q)$. Again GREEDY will not schedule J_ℓ on the same machine as J_0 (even if J_0 is run on the fast machine), and hence not start it later than at time $q + 1$ (assuming that J_ℓ is run on the slow machine, otherwise it starts not after time $(q+1)/q$). □

3 Algorithm BALANCE

We give an algorithm for identical machines of competitive ratio 12. This algorithm works in phases and uses an estimate on $\text{OPT}(\sigma)$ which is denoted by λ. A job is called *large* if its size is more than λ, and *small* otherwise; if $\lambda \geq \text{OPT}(\sigma)$, OPT can only run one such job on each machine. Also, once OPT has done this, it cannot use that machine anymore for any job.

A phase of BALANCE ends if it is clear from the small jobs that arrived in the phase, and from the large jobs that exist, that if another job arrives then $\lambda \leq \text{OPT}(\sigma)$. In this case we double λ and start a new phase.

In every phase, Balance only uses machines that do not already have large jobs. Each such machine will receive jobs according to one of the two following possibilities.

1. Only small jobs, of total weight in that phase less than 3λ.

2. Small jobs of weight less than 2λ, and one large job on top of them.

A machine that received a large job is called *large-heavy*, a machine that received weight of at least 2λ of small jobs in the current phase is called *small-heavy*. Both small-heavy and large-heavy machines are considered *heavy*. A machine that received more than a weight of λ of small jobs in the current phase but at most 2λ (and no large job) is considered *half-heavy*. Other machines are *non-heavy*. A machine that is not heavy (but possibly half-heavy) is called *active*. The algorithm Balance also maintains a set Q that contains the active machines.

Define λ_i as the value of λ in phase i. The algorithm Balance starts with phase 0 which is different from the other phases. In phase 0, m jobs arrive that are assigned to different machines. We then set λ_0 equal to the size of the smallest job that has arrived. Then the first of the regular phases starts.

Phases: A new phase starts when $Q = \emptyset$, i. e. there are no active machines anymore. (Phase 1 starts when phase 0 ends.) At the start of phase $i > 0$, we set $\lambda_i = 2\lambda_{i-1}$. Then Q contains all machines that do not have a large job. This holds because no machine has yet received any job in the current phase, so no machine can be small-heavy. Note that such a large job has arrived in some previous phase, but that the definition of large jobs has changed compared to the previous phase. I. e. not all the large jobs from previous phases are still large.

At all times, the algorithm only uses active machines. When the phase starts, all active machines are non-heavy. Each phase consists of two parts. The first part continues as long as there is at least one non-heavy machine among the active machines. As soon as no machine is non-heavy, the second part starts.

Part 1 For small jobs, Balance uses the machines in Q in a Next Fit-fashion, moving to the next machine as soon as a machine has received a load of more than λ_i in the current phase. An arriving large job is assigned to a machine that already has weight of more than λ_i. If no such machine exists, it is assigned to the active machine that Balance is currently using or going to use for small jobs (there is a unique such machine, and all other non-heavy machines did not receive any jobs in the current phase). A machine that receives a large job becomes large-heavy, and is removed from Q.

Part 2 We again start using the machines in Q in a Next Fit-fashion, moving to the next machine as soon as the machine has received a total load at least $2\lambda_i$ in the current phase. A machine that receives weight of at least $2\lambda_i$ of small jobs in total in this phase becomes small-heavy and hence stops being active (is removed from Q). A machine that receives a large job becomes large-heavy and also stops being active (it is removed from Q).

As long as $|Q| > 0$, there are active machines. When $Q = \emptyset$, a new phase starts. An example of a run of BALANCE can be seen in Figure 1.

We show that as soon as a first job in the new phase arrives, then $\lambda_{i-1} \le \textsc{opt}(\sigma)$. (Note that it is possible that no jobs arrive in a phase; this happens if $Q = \emptyset$ at the beginning of a phase.)

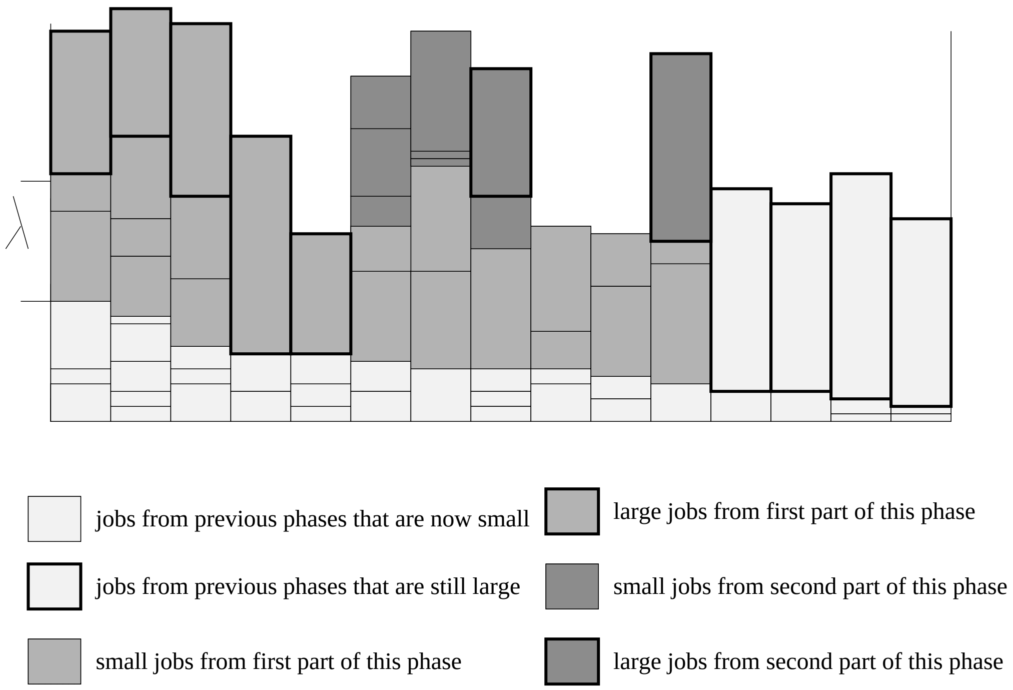
λ
jobs from previous phases that are now small
jobs from previous phases that are still large
small jobs from first part of this phase
large jobs from first part of this phase
small jobs from second part of this phase
large jobs from second part of this phase

Figure 1: A run of BALANCE

Lemma 3 *In each phase $i > 0$ in which jobs arrive, we have $\textsc{opt}(\sigma) \ge \lambda_i/2$, where σ is the sequence of jobs that arrived until phase i, including the first job of phase i.*

Proof. The lemma holds for phase 1, since there is at least one machine of the optimal off-line algorithm that has two scheduled jobs after the first job in phase 1 arrives.

Consider a phase $i > 1$. If phase i starts when phase $i-1$ is still in its first part, then no machines are small-heavy. Hence in total m jobs have arrived that were considered large in phase $i-1$ (where some may have arrived before phase $i-1$). After the first job arrives in phase i, we have $\textsc{opt}(\sigma) > \lambda_{i-1} = \lambda_i/2$.

If phase i starts while phase $i-1$ is in its second part, let K be the set of large jobs that were assigned to non-heavy machines in phase $i-1$. (If no such jobs exist, $K = \emptyset$). The jobs in K arrived in part 1 of phase $i-1$, since in part 2 only half-heavy machines are used. In part 1 of a phase, the active machines that have already been used are half-heavy or large-heavy.

Assume by contradiction that $\text{OPT}(\sigma) < \lambda_{i-1}$. Suppose $K \neq \emptyset$. Denote the last job in K by J_K and denote the set of machines that are still active after J_K has arrived by Q'. Write $q = |Q'|$. There was no half-heavy machines available for J_K, so all the machines that already received jobs in phase $i-1$, including the one that received J_K, are large-heavy at this point (they cannot be small-heavy in part 1). If $K = \emptyset$, define Q' as the set of active machines at the start of phase $i-1$. Clearly, all machines not in Q' are large-heavy at that point.

From this, we have that there exist $m-q$ large jobs after J_K has arrived (or at the start of phase $i-1$): all machines not in Q' either were large-heavy when phase $i-1$ started, or became large-heavy during it. Hence there are $m-q$ machines of OPT with a large job, since OPT cannot put two large jobs on one machine; OPT cannot put any more jobs on those machines if $\text{OPT}(\sigma) < \lambda_{i-1}$. Consider the set Q'_{OPT} of machines of OPT that do not run any of the $m-q$ large jobs that arrived already. We have $|Q'_{\text{OPT}}| = |Q'| = q$.

We calculate how much weight can be assigned by Balance to the machines in Q' (or equivalently, by OPT to the machines in Q'_{OPT}) in the remainder of phase $i-1$. In the schedule of OPT, the machines in Q'_{OPT} have some q jobs running last on them. Apart from that they have at most an amount of $\text{OPT}(\sigma) < \lambda_{i-1}$ small jobs.

Let $q_1 \leq q$ be the number of large jobs assigned by Balance to machines in Q' in the remainder of phase $i-1$. At the end of phase $i-1$, each machine in Q' is either small-heavy, or has an amount of at least λ_{i-1} small jobs and a large job. The total weight of small jobs assigned in phase $i-1$ to the machines of Q' by Balance is at least $(2q - q_1)\lambda_i$.

Suppose we remove the q largest jobs assigned in phase $i-1$ to the machines of the set Q' in the assignment of Balance. This means that we remove q_1 large jobs and $q - q_1$ small jobs. By definition, each small removed job has size of at most λ_{i-1}, so we removed at most an amount of $(q-q_1)\lambda_{i-1}$ small jobs. Therefore we are left with total weight of at least $q\lambda_{i-1}$ on the machines in Q', counting only weight from jobs that arrived in this phase.

This implies that even if OPT runs the largest q jobs last on the machines in Q'_{OPT}, it starts at least one of them at time λ_{i-1} or later, by the total weight of the other jobs. This gives a contradiction, already without the first job in phase i. This proves the lemma. □

Theorem 2 *Algorithm* Balance *has a competitive ratio of 12.*

Proof. Consider the last phase $\ell > 0$ in which jobs arrived. (If $\ell = 0$, Balance is optimal.) Let $\Lambda = \lambda_\ell$. We have $\text{OPT} \geq \Lambda/2$ by Lemma 3. Consider the machines that received jobs in phase ℓ, and for each such machine, consider the total size of jobs below the last job that is run on that machine. (For the machines that did not receive jobs in this phase, we have stronger bounds.) This size consists of three parts:

- The small jobs of phase ℓ

- The small jobs of previous phases
- The large jobs of previous phases

For the computation, for phases $0 < i < \ell$ in which a machine got only small jobs, we replace an amount of $2\lambda_i$ of small jobs from that phase by one (large) job. (Possibly a small job is broken in two parts to get a total of exactly $2\lambda_i$.) Because we only consider machines that received jobs in phase ℓ, the maximum starting time is unaffected by this substitution. As a result, each machine receives at most a weight of $2\lambda_i$ of small jobs in phase i.

In phase ℓ, each machine receives at most 2Λ of small jobs before it receives its last job. The value of λ is doubled between phases, hence the total amount of small jobs from previous phases on a machine is at most $\sum_{i<\ell} 2\lambda_i \leq 2\Lambda$.

We still need to consider the large jobs from previous phases. We count the large jobs not by the phases they arrive; instead, each large job is counted in the first phase where it is not large anymore, and the machine is active again. The large jobs that replace 2Λ worth of small jobs as described above, are always already small in the subsequent phase. For each phase $i \leq \ell$, a machine has at most one job that has just become small. This job is of size at most λ_i. Hence in total the size of all these jobs is at most $\sum_{i\leq\ell} \lambda_i \leq 2\Lambda$. Therefore the total load below the last jobs on any machine is at most $2\Lambda + 2\Lambda + 2\Lambda \leq 6\Lambda$. Since $\Lambda \leq 2\text{OPT}$, we are done. □

4 Lower Bounds

In the following proofs, we take M to be a large constant. If we construct a job sequence σ that contains a job of size M, then we assume that M is larger than $\mathcal{R}$ times the sum of smaller jobs in σ, where $\mathcal{R}$ is the competitive ratio that we want to show. This choice of M ensures that if a machine is assigned a job of size M, it cannot receive any other job after this without violating the competitive ratio.

Lemma 4 *Suppose we have a job sequence σ that shows that $\mathcal{R}(\mathcal{A}) \geq \mathcal{R}$ for all on-line algorithms on m_1 machines. Then for any $m > m_1$, $\mathcal{R}(\mathcal{A}) \geq \mathcal{R}$ for all on-line algorithms on m machines, as well.*

Proof. Construct the sequence σ' by adding $m - m_1$ jobs of size M before the first job of σ. The optimal cost for this sequence is the same as for σ on m_1 machines. On the machines that do not run the first $m - m_1$ jobs, we have that $\mathcal{A}$ must have a cost at least $\mathcal{R}$ times the optimal cost for σ on m_1 machines, and we are done. □

Theorem 3 *Take $\alpha = (\sqrt{5}+1)/2 \approx 1.618$ and M a large constant. For all on-line algorithms $\mathcal{A}$, we have the following lower bounds for the competitive ratio.*

Number of machines	*Job sequence*	$\mathcal{R}$
2	$1, M, 1, M$	2
3	$1/2, 1/2, M, 1, M, 1, M$	$5/2$
4	$\alpha - 1, \alpha - 1, M, M, 1, M, 1, M$	$\alpha + 1 \approx 2.618$

Moreover, as the number of machines tends to infinity, the competitive ratio tends to at least 4.

Proof. For $m \leq 4$, we use the job sequences described in the table above. For these sequences, any on-line algorithm that has a better competitive ratio than in the last column of the table must assign these jobs in the same way as the greedy algorithm, or violate the competitive ratio. In all cases, after the last job arrives we have $\text{OPT}(\sigma) = 1$ and $\mathcal{A}(\sigma) = \mathcal{R}$.

As an example, for $m = 4$, the first four jobs must be assigned to four different machines, the next two jobs to the machines with the jobs of size $\alpha - 1$, and the last two to the machine that does not have a job of size M yet. The sequence stops as soon as $\mathcal{A}$ assigns a job differently than described here, or after the fourth large job.

For larger m, we use the following job sequence. Assume $m = 2^r$ for some $r \geq 3$, and consider a sequence of real numbers $\{k_i\}_{i=0}^{\infty}$ with properties to be defined later. We will first define the job sequence and then specify for which value of r it works. The job sequence consists of $r+1$ steps. For $1 \leq i \leq r$, in step i first $m/2^i$ jobs of size k_i arrive, and then $m/2^i$ jobs of size M. In step $r+1$, one last job of size k_r arrives, followed by a job of size M.

We denote the optimal maximum starting time after step i by OPT_i. If $k_i \leq k_{i+1}$ for all $1 \leq i \leq r-1$, then for $1 \leq i \leq r$, we have $\text{OPT}_i = k_{i-1}$ (we put $k_0 = 0$), which is seen as follows. We describe the optimal schedule after step i. (We note that the optimal schedules after different steps can be very different.) There are $m/2^i$ machines with one job of size k_i, and $m/2^i$ machines with one job of size M. These machines do not have any other jobs. The remaining machines have one job of size k_s for some $s < i$, and after it one job of size M. After the last step we have $\text{OPT}_{r+1} = k_r$. In this case, all machines have one job of size k_s for some $s \leq r$, and after it one job of size M.

We will now define the sequence $\{k_i\}_{i=0}^{\infty}$ in such a way that the on-line algorithm cannot place two jobs on the same machine in one step. By induction we can see that at the start of step i $(1 \leq i \leq r)$, $m(1 - 1/2^{i-1})$ jobs of size M have already arrived. Thus if the on-line algorithm places the $m/2^{i-1}$ jobs from step i on different machines (that moreover do not have a job of size M yet), then also by induction, after every step i $(1 \leq i \leq r)$, every machine of the on-line algorithm either has a job of size M, or it has one job of *each* size k_j, for $1 \leq j \leq i$.

Define $s_i = \sum_{j=1}^{i} k_j$. If the on-line algorithm does put two jobs on the same machine in some step $i \leq r$, then by the above the last job on that machine starts at time $\sum_{j=1}^{i} k_j$ and the implied ratio is

$$\mathcal{R}_i = \frac{\sum_{j=1}^{i} k_j}{k_{i-1}} = \frac{s_i}{k_{i-1}} \ . \tag{2}$$

If the on-line algorithm never does this, then in the final step $r+1$ it has only $m/2^r = 1$ machine left without a job of size M, and this machine has one job of each size k_j for $1 \leq j \leq i$. The on-line algorithm has minimal cost if it places the two jobs from step $i+1$ on this machine, and the implied competitive ratio is thus $\mathcal{R}_{r+1} = (\sum_{j=1}^{r} k_j + k_r)/k_r = (s_r + k_r)/k_r$. Using (2), we will define the sequence $\{k_i\}_{i=0}^{\infty}$ so that $\mathcal{R}_i = \mathcal{R}$ is a constant for $1 \leq i \leq r+1$. This implies $k_0 = 0$, $k_1 = 1$, $k_i = \mathcal{R}k_{i-1} - \sum_{j=1}^{i-1} k_j = \mathcal{R}k_{i-1} - s_{i-1}$ for $i > 1$. This proves a competitive ratio of $\mathcal{R}$ if $k_i \leq k_{i+1}$ for $1 \leq i \leq r-1$ and $(s_r + k_r)/k_r \geq \mathcal{R}$ (where this last condition follows from step $r+1$). We have

$$\begin{aligned}
(s_r + k_r)/k_r \geq \mathcal{R} \quad &\Longleftrightarrow \quad k_r + s_r \geq Rk_r = s_{r+1} \qquad \text{using (2)} \\
&\Longleftrightarrow \quad k_r \geq s_{r+1} - s_r = k_{r+1} \\
&\Longleftrightarrow \quad s_{r+1} \geq s_{r+2} \Longleftrightarrow k_{r+2} \leq 0 \Longleftrightarrow s_{r+3} \leq 0.
\end{aligned}$$

Hence it is sufficient to show that the sequence $\{s_i\}_{i=0}^{\infty}$ has its first nonpositive term s_{r+3} for some $r \geq 1$. This value of r determines for which m this job sequence shows a lower bound of $\mathcal{R}$, since $m = 2^r$. Note that if s_{r+3} is nonpositive, we have to stop the job sequence after step $r+1$ at the latest, because by the above $k_{r+2} \leq 0 < k_1 \leq k_r$: the sequence is no longer non-decreasing. As stated above, we will in fact give one final job of size k_r in step $r+1$, and a job of size M, and thus not use any value k_i for $i > r$. The sequence $\{s_i\}_{i=0}^{\infty}$ satisfies the recurrence $s_{i+2} - \mathcal{R}s_{i+1} + \mathcal{R}s_i = 0$. For $\mathcal{R} < 4$, the solution of this recurrence is given by

$$s_i = \frac{2\sin(\theta i)\sqrt{\mathcal{R}}^{\,i}}{\sqrt{4\mathcal{R} - \mathcal{R}^2}} \qquad \text{where} \quad \cos\theta = -\frac{1}{2}\sqrt{\mathcal{R}} \quad \text{and} \quad \sin\theta = \sqrt{1 - \frac{\mathcal{R}}{4}}.$$

Since $\sin\theta \neq 0$, then $\theta \neq 0$, which implies $s_i < 0$ for some value of i. Furthermore, this value of i tends to ∞ as $\mathcal{R}$ tends to 4 from below. Direct calculations show that for $i = 1, 2, 3, 4$, $s_i > 0$, hence given such minimal integer i, we can define $r = i - 3$. From the calculations it also follows that $\{k_i\}_{i=0}^{r}$ is non-decreasing.

In conclusion, for any value of $\mathcal{R} < 4$ it is possible to find a value r so that any on-line algorithm has at least a competitive ratio of $\mathcal{R}$ on 2^r machines. By Lemma 4, this implies that for every $\varepsilon > 0$, there exists a value m_1 such that for any on-line algorithm $\mathcal{A}$ on $m > m_1$ machines, $\mathcal{R}(\mathcal{A}) \geq 4 - \varepsilon$. □

Note that this proof does not hold for $\mathcal{R} \geq 4$, because the solution of the recurrence in that case is not guaranteed to be below 0 for any i.

Corollary 1 *On identical machines,* GREEDY *is optimal for $m = 2, 3$.*

Proof. This follows from Lemma 1 and Theorem 3. □

5 Related machines

We only study the special case $m = 2$. The reason for this is that already for $m = 2$, the competitive ratio is unbounded as $q \to \infty$. We give a matching lower bound to the upper bound from Lemma 2, showing that Greedy is optimal for this case.

Theorem 4 *For the problem of minimizing the maximum starting time on two related machines, the competitive ratio is at least* $q+1$.

Proof. Consider an algorithm $\mathcal{A}$ for this problem and suppose it has a competitive ratio of less than $q+1$. A job of size 1 arrives. If $\mathcal{A}$ places it on the slow machine (the machine with speed 1), then a job of size M arrives (which has to go on the other machine; M is defined as in Section 4), followed by a job of size q and another job of size M. The maximum starting time of $\mathcal{A}$ is at least $q+1$, whereas the optimal maximum starting time is 1, by putting the job of size 1 on the slow machine, the job of size q on the fast machine, and starting both the large jobs at time 1.

If $\mathcal{A}$ places the first job on the fast machine, then take N a large constant. The second job has size Nq and must be placed on the slow machine. The third job has size NM, where $M=(q+1)N$, and must be placed on the fast machine, otherwise a competitive ratio of $Nq/(1/q)=Nq^2$ is implied. Then a job of size $N-1$ arrives which must go on the slow machine; finally another job of size NM arrives. $\mathcal{A}$ starts its last job at time $Nq+(N-1)$ whereas in the optimal schedule, no job starts after time N. By letting N grow without bound (maintaining $M=(q+1)N$), this proves the ratio. □

6 Resource Augmentation

We now consider on-line algorithms that have more resources than the off-line algorithm. It turns out that in these changed circumstances, GREEDY is optimal in the sense that it requires the minimum possible number of machines to have a competitive ratio of 1. We only consider identical machines in this section.

Lemma 5 $\mathcal{R}(\text{GREEDY})=1$ *if it has at least* $2m-1$ *machines.*

Proof. Let $h=\text{GREEDY}(\sigma)$ and $h^*=\text{OPT}(\sigma)$. Note that the last job J_ℓ that is assigned at time h by GREEDY is a final job for OPT as well, since this is the very last job in the sequence. Let S be the set of on-line machines of GREEDY that only contain non-final jobs or J_ℓ. Since there are at most m final jobs, $|S|\geq 2m-1-(m-1)=m$. All of GREEDY's machines are occupied from 0 to h. The machines in S are occupied during this time by non-final jobs. Let W be the total size of non-final jobs. We have $W\geq mh$. But $W\leq h^*m$. Hence $h\leq h^*$. □

Note that a similar proof shows that the competitive ratio of GREEDY tends to zero as the number of on-line machines tends to ∞.

Lemma 6 *Any algorithm that has at most* $2m-2$ *machines has a competitive ratio greater than 1.*

Proof. Suppose $\mathcal{A}$ has a competitive ratio of at most 1. We use a construction in phases, where in each phase the size of the arriving jobs is equal to the total size of all the jobs from the previous phases. Let n_i denote the number of jobs in phase i, and M_i denote the size of the jobs in phase i. We determine the number of phases later. We take $n_0 = m$ and $n_i = 2m-1$ for $i > 0$. Furthermore, we take $M_0 = 1$, $M_1 = n_0 M_0 = m$ and

$$M_i = \sum_{j=0}^{i-1} n_j M_j = \sum_{j=0}^{i-2} n_j M_j + n_{i-1} M_{i-1} = 2m M_{i-1} \text{ for } i > 1.$$

Claim: After i phases, at least $\min(m + (m-1)(1 - \frac{1}{2^i}), 2m-2)$ machines are non-empty.

Proof: We use an induction. All jobs from phase 0 have to be assigned to different machines to have a finite competitive ratio, so m machines are non-empty after phase 0.

Consider phase i for $i > 0$. During each phase $i > 0$, the optimal costs are at most M_i: all the jobs from the previous phases go together on one machine, followed by one job of size M_i. All other machines have two jobs of size M_i. In order to have a competitive ratio of 1, $\mathcal{A}$ can assign at most one job of size M_i on each non-empty machine, and at most 2 such jobs on each empty machine. Let x be the number of non-empty machines at the start of phase i. If $x = 2m-2$ we are done immediately. Else, we have $x \geq m + (m-1)(1 - \frac{1}{2^{i-1}})$ by induction. The number of machines that become non-empty in phase i is at least $(2m-1-x)/2$, so after phase i, at least $m - \frac{1}{2} - \frac{1}{2}x + x$ machines are non-empty. By induction, we have $m - \frac{1}{2} + \frac{1}{2}x \geq m - \frac{1}{2} + (m + (m-1)(1 - \frac{1}{2^{i-1}}))/2 = m + (m-1)(1 - \frac{1}{2^i})$. $\square$

Taking $k = \lceil \log_2 m \rceil$, we have that after k phases, $m + (m-1)(1 - \frac{1}{2^k}) \geq m + (m-1)(1 - \frac{1}{m}) > 2m-2$, hence $\mathcal{A}$ needs more than $2m-2$ machines to maintain a competitive ratio of 1. $\square$

Note that no algorithm $\mathcal{A}$ which uses $2m-1$ machines can have competitive ratio less than 1, due to the sequence $1, \ldots, 1$ ($2m$ jobs). At least two jobs run on the same on-line machine, hence $\mathcal{A}(\sigma) = \text{OPT}(\sigma) = 1$.

7 Conclusions

We showed that the greedy algorithm is far from being optimal in one measure (competitive ratio), but optimal in a different measure (amount of resource augmentation). This phenomenon raises many questions. Which of the two measures is more appropriate for this problem? Furthermore, which measure is appropriate for other problems? Is it possible to introduce a different measure that would solve the question: is GREEDY a good algorithm to use?

Acknowledgement

The authors would like to thank Han La Poutré for helpful suggestions.

References

[1] S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, Jul 1997.

[2] A. Avidor, Y. Azar, and J. Sgall. Ancient and new algorithms for load balancing in the l_p norm. In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms*, pages 426–435, 1998.

[3] Y. Azar, L. Epstein, and R. van Stee. Resource augmentation in load balancing. *Journal of Scheduling*, 3(5):249–258, 2000.

[4] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symposium on Theory of Algorithms*, pages 51–58, 1992. To appear in *Journal of Computer and System Sciences.*

[5] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, 6:181–193, 1999.

[6] M. Brehob, E. Torng, and P. Uthaisombut. Applying extra-resource analysis to load balancing. *Journal of Scheduling*, 3(5):273–288, 2000.

[7] J. Edmonds. Scheduling in the dark. *Theoretical Computer Science*, 235:109–141, 2000.

[8] R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3(5):343–353, 2000.

[9] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–565. ACM-SIAM, 2000.

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

[11] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:617–643, 2000.

[12] B. Kalyanasundaram and K. Pruhs. Maximizing job completions online. In G. Bilardi, G.F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms - ESA '98, Proceedings Sixth Annual European Symposium*, volume 1461 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 1998. To appear in Journal of Algorithms.

[13] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[14] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.