Max-min online allocations with a reordering buffer

Leah Epstein[1], Asaf Levin[2,⋆], and Rob van Stee[3,⋆⋆]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel.
lea@math.haifa.ac.il
[2] Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel.
levinas@ie.technion.ac.il.
[3] University of Karlsruhe, Department of Computer Science, 76128 Karlsruhe, Germany.
vanstee@ira.uka.de

Abstract. We consider online scheduling so as to maximize the minimum load, using a reordering buffer which can store some of the jobs before they are assigned irrevocably to machines. For m identical machines, we show an upper bound of $H_{m-1}+1$ for a buffer of size $m-1$. A competitive ratio below H_m is not possible with any finite buffer size, and it requires a buffer of size $\tilde{\Omega}(m)$ to get a ratio of $O(\log m)$. For uniformly related machines, we show that a buffer of size $m+1$ is sufficient to get an approximation ratio of m, which is best possible for any finite sized buffer. Finally, for the restricted assignment model, we show lower bounds identical to those of uniformly related machines, but using different constructions. In addition, we design an algorithm of approximation ratio $O(m)$ which uses a finite sized buffer. We give tight bounds for two machines in all the three models.

These results sharply contrast to the (previously known) results which can be achieved without the usage of a reordering buffer, where it is not possible to get a ratio below an approximation ratio of m already for identical machines, and it is impossible to obtain an algorithm of finite approximation ratio in the other two models, even for $m=2$. Our results strengthen the previous conclusion that a reordering buffer is a powerful tool and it allows a significant decrease in the competitive ratio of online algorithms for scheduling problems. Another interesting aspect of our results is that our algorithm for identical machines imitates the behavior of the greedy algorithm on (a specific set of) related machines, whereas our algorithm for related machines completely ignores the speeds until the end, and then only uses the relative order of the speeds.

1 Introduction

Scheduling problems are most frequently described in a framework of assigning jobs to machines. There are various kinds of scheduling problems depending upon the properties of the machines, allowable assignments, and the cost criteria. Our goal is to study a natural model where the assignment of jobs is performed dynamically, in the sense that jobs are being considered one by one, and generally must be assigned in this order,

⋆ Chaya fellow.
⋆⋆ Research performed at Max Planck Institute for Informatics, Germany, and supported by the German Research Foundation (DFG).

while the information on future jobs is unknown. We consider parallel machines, and a problem variant which possesses features of both offline and online scenarios. These are not offline problems, since the input arrives gradually, but they are not purely online either, since we allow partial reordering of the input.

More specifically, in this paper we study a variant called *scheduling with a reordering buffer*, where a buffer, which can store a fixed number of unassigned jobs, is available. Thus each new job must be either assigned to a machine or stored in the buffer (possibly in the case where the buffer is already full, the algorithm is forced to evict another job from the buffer, which must be assigned to a machine immediately). A job which is assigned to a machine, is assigned irrevocably to that machine.

In this paper, we are concerned with max-min allocations, that is, the goal is maximizing the minimum load. The concept of such allocations is related to the notion of fair division of resources [2]. Originally, the goal function was used for describing systems where the complete system relies on keeping all the machines productive for as long as possible, as the entire system fails even in a case that just one of the machines ceases to be active [18]. Additional motivations for the goal function come from issues of Quality of Service. From the networking aspect, this problem has applications to basic problems in network optimization such as fair bandwidth allocation. Consider pairs of terminal nodes that wish to communicate; we would like to allocate bandwidth to the connections in a way that no link unnecessarily suffers from starvation, and all links get a fair amount of resources. Another motivation is efficient routing of traffic. Consider parallel links between pairs of terminal nodes. Requests for shifting flow are assigned to the links. We are interested in having the loads of the links balanced, in the sense that each link should be assigned a reasonable amount of flow, compared to the other links. Yet another incentive to consider this goal function is congestion control by fair queuing. Consider a router that can serve m shifting requests at a time. The data pieces of various sizes, needing to be shifted, are arranged in m queues (each queue may have a different data rate), each pays a price which equals the delay that it causes in the waiting line. Our goal function ensures that no piece gets a "preferred treatment" and that they all get at least some amount of delay.

We are mainly interested in two models of parallel machines, namely identical machines [20] and uniformly related machines [3, 7]. The input is a stream of jobs, of indices $1, 2, \ldots$, where the single attribute of each job j is its processing time, p_j, also known as its size. The goal is always to partition the jobs into m subsets, where each subset is to be executed on one specific machine, that is, there are m machines, of indices $\{1, 2, \ldots, m\}$, and each job is to be assigned to one of them. Online algorithms see the input jobs one at a time, and need to assign each job before becoming familiar with the remaining input. In the identical machines model, the completion time (also called load) of a machine is the total size of the jobs assigned to it, while for uniformly related machines (also known as related machines), each machine i has a speed s_i associated with it, and the completion time (or load) of a machine is the total size of jobs assigned to it, scaled by the speed of the machine. Without loss of generality, let the speeds be $s_1 \leq \cdots \leq s_m$.

An additional common variant considered here is *restricted assignment* [5]. The machines have identical speeds, though each job j is associated with a subset of the machines, $M_j \subseteq \{1, 2, \ldots, m\}$, and can be assigned only to a machine in M_j.

Notations Throughout the paper we use the following notations. The size of the buffer is denoted by K. We use OPT to denote an optimal solution as well as its value or profit (i.e., the load of its least loaded machine). For an (offline or online) algorithm ALG we denote its value or profit by ALG as well. For a minimization problem, the value of a solution is called its *cost* and the approximation ratio of ALG is the infimum $\mathcal{R}$ such that for any input, $\text{ALG} \leq \mathcal{R} \cdot \text{OPT}$, whereas for a maximization problem (such as the problem studied here), the approximation ratio of ALG is the infimum $\mathcal{R}$ such that for any input, $\mathcal{R} \cdot \text{ALG} \geq \text{OPT}$ (note that we use numbers greater than 1 for approximation ratios for a maximization problem). If the approximation ratio of ALG is at most r, then we say that it is an r-approximation. Let H_t denote the harmonic series, that is, $H_t = \sum_{i=1}^{t} \frac{1}{t}$.

Related work This problem (without a buffer) has been well studied (known by different names such as "machine covering" and "the Santa Claus problem") in the computer science literature (see e.g. [18, 12, 11, 24, 6, 15, 17, 2]). For identical machines, it is known that any online algorithm for identical machines has an approximation ratio of at least m (the proof is folklore, see [24, 4]), and this bound can be achieved using a greedy algorithm which assigns each job to the least loaded machine, as was shown by Woeginger [24]. Before the usage of our approach of incorporating a reordering buffer became popular, there were other attempts to overcome this bound. Randomized algorithms were studied in [4], where it was shown that the best approximation ratio which can be achieved using randomization is $\tilde{\Theta}(\sqrt{m})$. Several types of semi-online variants were considered, where one of the following was assumed: the largest processing time of any job is given in advance, the total size is given in advance, or both are given. It was shown that in these cases, the best approximation ratio remains $\Theta(m)$ [23, 8]. Here we show that our approach allows us to reduce the approximation ratio much more significantly. It should be noted, that an additional, much stronger, semi-online variant was studied as well, where it is assumed that jobs arrive sorted by non-increasing processing time. In this variant, which is much closer to an offline problem than our model, the approximation ratio is at most $\frac{4}{3}$ [11, 12].

For uniformly related machines, no algorithm with finite approximation ratio exists even for two machines [4]. The semi-online problem where jobs arrive sorted by non-increasing order, and the problem where the profit of an optimal algorithm is known in advance, admit m-approximations, which is best possible in both cases. If the two types of information are combined, a 2-approximation is known [4]. To the best of our knowledge, no positive results are known for the case of restricted assignment. It is known that no finite approximation ratio can be achieved for a purely online algorithm, even for two machines [9], and even in the model of hierarchical machines, where for every j, the set M_j is a prefix of the machine set.

In all variants of scheduling with a buffer studied in the past, a finite length buffer (usually of size $O(m)$) already allowed to achieve the best possible approximation ratio (for any finite sized buffer). Moreover, in almost all cases, the approximation ratio is

significantly reduced, compared to the best possible purely online algorithm. We next survey the results for the min-max allocation problem (also known as the minimum makespan problem), whose goal is dual to our goal function, with a buffer.

Kellerer et al. [22] and Zhang [25] considered the case of two identical machines, and showed that a buffer of size 1 allows to achieve an approximation ratio of $\frac{4}{3}$, which is best possible. For m identical machines, Englert et al.[16] showed that a buffer of size $O(m)$ is sufficient. Their algorithm has the best possible approximation ratio for every value of m, while this ratio tends to 1.47 for large m. It is known that for online scheduling, no algorithm can have an approximation ratio smaller than 1.853 [1, 19]. For the more general case of uniformly related machines, it was shown in [13] that for two machines, a buffer of size 2 is sufficient to achieve the best approximation ratio. In fact, for some speed ratios between the two machines, a buffer of size 1 is sufficient, while for some other speed ratios, a buffer of size 1 provably is not enough to achieve the best bound. Note that it was shown by [16] that a buffer of size $m-1$ reduces the approximation ratio for uniformly related machines below the lower bound of the case without a reordering buffer, specifically, in addition to the case of identical machines, Englert et al. [16] designed a 2-approximation algorithm for related machines, which uses a buffer of size $O(m)$. Whereas, without a buffer it is known that no algorithm can have an approximation ratio below 2.43 [7]. Finally, for the standard online scheduling problem in the restricted assignment model, there are tight bounds of $\Theta(\log m)$ on the approximation ratio [5]. The lower bound still holds if there is a buffer, since the construction of [5] holds for fractional assignment (where a job can be split arbitrarily among the machines that can run it); each job can be replaced by very small jobs, and so the usage of a buffer does not change the result.

The analogous questions for preemptive scheduling on identical machines were resolved in [14]. In this variant, jobs can be arbitrarily distributed among the machines, with the restriction that two parts of one job cannot be simultaneously processed on two machines. The best possible upper bound over all values of m is $\frac{4}{3}$, while for the case without a buffer, this value is approximately 1.58, as was shown by Chen et al. [10].

Our results For identical machines, we design an algorithm which uses a buffer of size $m-1$, and has an approximation ratio of at most $H_{m-1}+1$. For $m=2$, we design a different $\frac{3}{2}$-approximation algorithm, which is optimal. We show a lower bound of H_m for any finite sized buffer, and in addition, we show that for a buffer size of $o(\frac{m}{\log m})$, an upper bound of $O(\log m)$ cannot be achieved. Interestingly, our algorithm IMITATE imitates the behavior of a greedy algorithm for uniformly related machines, with the speeds $\{1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{m}\}$.

For the cases of related machines and restricted assignment, we extend the known negative results of [4, 9], and show that a buffer of size at most $m-2$ does not admit an algorithm of finite approximation ratio. For an arbitrary sized buffer, we show lower bounds of m on the approximation ratio of any algorithm. The proofs of the lower bounds in the two models are very different, but the results are similar. We complement these result by $O(m)$-approximation algorithms, which use a finite sized buffer. Specifically, for the related machines model we design an algorithm of approximation ratio m, which uses a buffer of size $m+1$, and an algorithm of approximation ratio below $2m-1$, which uses a buffer of size $m-1$. For two machines we design a different

2-approximation algorithm, which uses a buffer of size 1. For the restricted assignment model we present an algorithm of approximation ratio at most $2m$ which uses a buffer of size $O(m^2)$. For two machines, we give a simpler algorithm of approximation ratio 2, which only requires a buffer of size 1. In contrast to the algorithm for identical machines, which attributes *phantom speeds* to the machines, the algorithm for related machines ignores the speeds during the main loop, and in the final loop, only uses the relative order of speeds, rather than their values.

Omitted proofs are deferred to the full version.

2 Identical machines

We start with the most elementary case of identical machines. We first show limitations on the performance and the required buffer size of any algorithm.

2.1 Lower bounds

In some of the lower bound proofs, the input sequence starts with a large number of very small jobs. In such a case, having a buffer is not meaningful, since almost all jobs must be assigned. The difficulty is in spreading these small jobs among the machines without knowing the sizes of future jobs. The next proof has some similarities with a lower bound for the preemptive variant [21].

Theorem 1. *For any buffer size K, no online algorithm can have an approximation ratio below H_m.*

Proof. We only give the construction here, and defer the proof to the full version. Let $\varepsilon > 0$ be a very small constant such that $\varepsilon = 1/N$ for some $N \in \mathbb{N}$ which is divisible by $m!$. The input contains N jobs of size ε, where $N >> K$. Let $b_1 \leq \cdots \leq b_m$ be the resulting loads for a given algorithm, after the last of these jobs has arrived. Since at most K of them remain in the buffer, we have $1 - K\varepsilon \leq \sum_{i=1}^{m} b_i \leq 1$.

Next, j jobs of size $1/(m-j)$ arrive for some $0 \leq j \leq m-1$. These j jobs are called *large jobs*. For this input, the optimal value is $\text{OPT}_j = 1/(m-j)$. □

The following proposition implies that we in fact need a buffer of size $\Omega(m/\log m)$ to get an approximation ratio which is logarithmic in m.

Proposition 1. *Let $f(m)$ be a function where $f(m) < m$ for all $m > 1$. For a buffer size $f(m)$, no algorithm can have an approximation ratio below $m/(f(m)+1)$.*

2.2 Algorithm

While the lower bound constructions are relatively simple, it is harder to see what an algorithm of sufficiently small approximation ratio should do. Intuitively, given the lower bound construction in the proof of Theorem 1, the machines should be relatively unbalanced. We achieve this by acting as if the machines have speeds (the so-called phantom

speeds). We next define the algorithm IMITATE, which has a buffer of size $m-1$. IMITATE always keeps the largest $m-1$ jobs in the buffer.

ALGORITHM IMITATE. We store the first $m-1$ jobs in the buffer. Upon an arrival of a job, in the case that the buffer already contains $m-1$ jobs, consider those jobs and the new job. Let the size of the smallest job among these m jobs be X. A job of size X will be assigned while the other jobs are stored in the buffer. The algorithm imitates the behavior of the so-called post-greedy algorithm for uniformly related machines. We define the phantom speed of the machine of index i to be $\frac{1}{i}$. Let L_i denote the current load of machine i. The next job is assigned to a machine which minimizes $i \cdot (L_i + X)$.

Upon termination of the input sequence, let $b_2 \leq \ldots \leq b_m$ be the sizes of jobs remaining in the buffer. (If there are only $j < m-1$ jobs in the buffer, then we let $b_i = 0$ for $2 \leq i \leq m-j$.) The job of size b_i is assigned to machine i (machine 1 does not get a job). We call this job *the final job* of machine i.

In what follows, we use ℓ_i to denote the load of machine i before the assignment of the final job, and L_i be the final load of machine i. That is, $L_i = \ell_i + b_i$ for $2 \leq i \leq m$ and $L_1 = \ell_1$. Note that if $b_2 = 0$, that is, the buffer contains less than $m-1$ jobs upon termination of the input sequence, then $\ell_i = 0$ for all $1 \leq i \leq m$.

To analyze the algorithm, we start with proving the following lemmas.

Lemma 1. *Let $1 \leq i \leq m$ and $2 \leq k \leq m$, $i \neq k$. Then $k \cdot L_k \geq i \cdot \ell_i$.*

Proof. If $\ell_i = 0$, then the claim trivially holds. Otherwise, let μ be the size of the last job ever assigned to machine i (neglecting the final job). Let λ_k be the load of machine k at that time in which μ was assigned. Then $k(\lambda_k + \mu) \geq i\ell_i$ by the post-greedy assignment rule. We have $L_k = \ell_k + b_k \geq \lambda_k + \mu$, using the properties $b_k \geq \mu$, since just before the assignment of the final jobs, the buffer contains the largest $m-1$ jobs in the input, and $\ell_k \geq \lambda_k$, since ℓ_k is the load of machine k just before the final jobs are assigned, while λ_k is the load of this machine at an earlier time. Hence $L_k \geq \lambda_k + \mu \geq \frac{i \cdot \ell_i}{k}$. □

Lemma 2. *For $i \geq 2$, $\ell_1 \geq (i-1)\ell_i$.*

Proof. If $\ell_i = 0$, then we are done. Otherwise let μ be the size of the last job ever assigned to machine i prior to the allocation of the final jobs. Let λ_1 be the load of machine 1 at the time when we assigned μ. Then $\lambda_1 + \mu \geq i\ell_i$ by the post-greedy assignment. Since $\ell_i \geq \mu$ and $\ell_1 \geq \lambda_1$, we get $\ell_1 \geq \lambda_1 \geq i\ell_i - \mu \geq (i-1)\ell_i$. □

We denote the total size of all jobs except for the final ones by Δ. That is, we let $\Delta = \sum_{i=1}^{m} \ell_i$.

Lemma 3. $(H_{m-1} + 1)\ell_1 \geq \Delta$.

Proof. By Lemma 2, we conclude that for all $i \geq 2$, $\ell_1 \geq (i-1)\ell_i$, and hence $\frac{\ell_1}{i-1} \geq \ell_i$. We sum up the last inequality for all $2 \leq i \leq m$, and we get $\sum_{i=2}^{m} \frac{\ell_1}{i-1} \geq \sum_{i=2}^{m} \ell_i = \Delta - \ell_1$. On the other hand, $\sum_{i=2}^{m} \frac{\ell_1}{i-1} = H_{m-1}\ell_1$, so $(H_{m-1}+1)\ell_1 \geq \Delta$. □

Lemma 4. *For any $k > 1$, $(H_m - \frac{1}{k})kL_k \geq \Delta - \ell_k$.*

Proof. For all $1 \leq i \leq m$ such that $i \neq k$, we have that $\ell_i \leq \frac{k \cdot L_k}{i}$ by Lemma 1. Summing up for all $i \neq k$ we get $\Delta - \ell_k = \sum_{i \neq k} \ell_i \leq \sum_{i \neq k} \frac{k \cdot L_k}{i} = kL_k \sum_{i \neq k} \frac{1}{i} = kL_k(H_m - \frac{1}{k})$. □

Lemma 5. *For any* $1 \leq k \leq m$, OPT $\leq \frac{1}{k}(\Delta + \sum_{j=2}^{k} b_j)$.

Theorem 2. *The approximation ratio of* IMITATE *is at most* $H_{m-1} + 1$.

3 Uniformly related machines

3.1 Lower bounds

Theorem 3. *For a buffer of size at most* $m - 2$*, no algorithm can have a finite approximation ratio on related machines.*

Proof. Let the speed of machine i be S^{i-1} for some large $S \geq m + 1$. Without loss of generality assume that the buffer has size $m - 2$. The input sequence starts with $m - 1$ jobs of sizes $S, S^2, \ldots, S^{m-1}$. At least one of these jobs cannot remain in the buffer. Let S^k be the size of the assigned job. Let j be the machine it is assigned to. If $j \leq k$, there is another job of size 1. Otherwise, another job of size S^m arrives.

In the first case, there is a machine in $\{k + 1, \ldots, m\}$ which does not receive a job in $\{S^k, \ldots, S^{m-1}\}$, this machine has a profit of at most $S^{k-1}/S^k = 1/S$ (there are m jobs, so if a machine gets two jobs then ALG $= 0$). In this case, OPT $= 1$.

In the second case, if the machines $\{k+1, \ldots, m\}$ have all jobs $\{S^k, \ldots, S^m\}$, then machines $\{1, \ldots, k\}$ only have $k - 1$ jobs and ALG $= 0$. Again, each machine must have one job exactly. Thus the machine having S^k has a profit of at most $S^k/S^k = 1$, while OPT $= S$. Letting $S \to \infty$, we get a ratio of ∞. □

Theorem 4. *For any buffer size* K*, no algorithm can have an approximation ratio below* m *for related machines.*

Proof. Again, we only give the construction here. Let the speed of machine i be S^{i-1} for some large $S \geq m + 1$. The first phase is the same as for identical machines (see Lemma 1). Then in the second phase for some $1 \leq j \leq m$, the next jobs are of sizes $1/S^{j-1}, 1/S^{j-2}, \ldots, 1/S$ and $S, S^2, \ldots, S^{m-j}$ (if $j = 1$ then the first set is empty and if $j = m$ then the second set is empty). In the second phase there is one job of each size, so there are $m-1$ jobs. We have that the optimal value in this case is $\text{OPT}_j = 1/(S^{j-1})$ (put the job of size $1/S^i$ on machine $j - i$ for $i \neq 0$ and all small jobs on machine j; note that i can be negative). □

3.2 An algorithm for m related machines

We next present an algorithm of approximation ratio m which uses a buffer of size $m+1$. In the full version, we design another algorithm of an approximation ratio slightly less than $2m - 1$, which uses a buffer of size $m - 1$. Our algorithms ignore the exact speeds, and only use their relative order.

Here we show that by using just two extra buffer positions compared to the minimum number of positions required to get an algorithm of finite approximation ratio, it is possible to get an optimal approximation ratio of m and hence save a factor of almost 2. In the case $m = 2$, the algorithm can be easily modified to keep only two jobs in the buffer, rather than three.

The algorithm works as follows. The first $m + 1$ jobs are stored in the buffer. Upon arrival of a job, it is possibly swapped with a job of the buffer to maintain the property that the buffer contains the largest jobs seen so far.

The algorithm runs List Scheduling (LS) [20] on the machines, while ignoring the speeds. That is, a job is assigned to a minimally loaded machine, that is, a machine for which the total size of jobs assigned to it so far is minimum. Let the remaining jobs in the buffer when the input ends be $c_0 \leq \cdots \leq c_m$. We slightly abuse notation and let c_i denote also the size of job c_i. If the buffer contains at most $m - 1$ jobs, then any assignment is optimal, since in this case, $\text{OPT} = 0$. If there are m jobs in the buffer, then assigning job c_i to machine i results in an optimal solution. The case that the buffer contains $m + 1$ jobs is analyzed below (even if no other jobs were assigned).

The jobs $\{c_0, c_1, \ldots, c_m\}$ are assigned in one of the following $2m - 1$ ways, depending on which option gives the largest profit.

1. Assign c_0 to the least loaded machine (in terms of the total size of jobs assigned to it), and c_i to machine i for all $i = 1, 2, \ldots, m$.
2. Assign the jobs as in the previous case, but move c_j to machine m for some $1 \leq j \leq m - 1$.
3. Assign c_i to machine i for all $i = 1, 2, \ldots, m$. assign c_0 to a machine j for some $1 \leq j \leq m - 1$.

In our analysis below, we show that there is always at least one assignment which shows that the approximation ratio is at most m.

Theorem 5. *The approximation ratio of this algorithm is at most m.*

Proof. We scale the input such that $\text{OPT} = 1$. Let T be the total size of all the jobs. Then $T \geq \sum_{i=1}^{m} s_i$. If $c_i \geq s_i/m$ for $i = 1, \ldots, m$, we are done. Else, let k be the maximum index for which $c_k < s_k/m$. We consider three cases.

Case 1: $k = m$ (i.e., $c_m < s_m/m$). We analyze only options where c_0 is assigned greedily to the least loaded machine. Let $a_1, \ldots, a_m$ be total sizes of jobs assigned to machines, neglecting $c_1, \ldots, c_m$ but not c_0. That is, since c_0 is assigned greedily, it is counted as part of some a_i.

If $a_m = \max_i a_i$, we analyze the first assignment option. Recall that T is the total size of all the jobs, that is $T = \sum_{i=1}^{m}(a_i + c_i)$. The load of machine m is

$$\frac{a_m + c_m}{s_m} = \frac{\max_i a_i + \max_i c_i}{s_m} \geq \frac{T/m}{s_m} \geq \frac{1}{m} .$$

For each other machine $i \leq m - 1$, we have $a_i + c_i \geq a_m$. To see this, note that if $a_m = 0$ this clearly holds. Otherwise, let x be the size of the last job assigned to machine m before c_m. Due to the assignment rule $a_i \geq a_m - x$. Since all the jobs in

the buffer are larger than all other jobsand since $c_0 \leq c_i$, we conclude that $x \leq c_i$ and the claim follows. Hence, $\sum_{j=1}^{m} a_j = T - \sum_{j=1}^{m} c_j \geq T - m \cdot \frac{s_m}{m} \geq s_i$, due to the definition of T, the property $c_i \leq c_m < \frac{s_m}{m}$ and $T \geq \sum_{j=1}^{m} s_j \geq s_i + s_m$.

Finally,

$$\frac{a_i + c_i}{s_i} \geq \frac{a_m}{s_i} \geq \frac{1}{m s_i} \sum_{j=1}^{m} a_j \geq \frac{1}{m}$$

holds since $a_i + c_i \geq a_m$, $a_m = \max_j a_j \geq \frac{1}{m}\sum_{j=1}^{m} a_j$, and $\sum_{j=1}^{m} a_j \geq s_i$.

If $a_m < \max_i a_i$, let $j = \arg\max_i a_i < m$. Consider the second type of assignment, for this specific value of j. As before, $\sum_{t=1}^{m} a_t \geq s_i$, $a_j = \max_t a_t \geq \frac{1}{m}\sum_{t=1}^{m} a_t$ and so $a_j \geq \frac{s_i}{m}$ for all $i < m$. Similarly to the previous proof, we can show $a_i + c_i \geq a_j$, so

$$\frac{a_i + c_i}{s_i} \geq \frac{a_j}{s_i} \geq \frac{1}{m} \text{ for } 1 \leq i < m, i \neq j.$$

For machine j, already $a_j/s_j \geq 1/m$. Finally, for machine m, $a_m + c_j \geq a_j \geq a_i$ (for all $1 \leq i \leq m$). We need to bound $(a_m + c_j + c_m)/s_m$ and we get $(a_m + c_j) + c_m \geq \max_i a_i + \max_i c_i \geq T/m \geq s_m/m$.

Case 2: $1 < k < m$. Consider the third assignment option, where c_0 is assigned to machine k. We now let a_i denote the total size of jobs assigned to machine i before the jobs $c_0, \ldots, c_m$ are assigned, so $T = \sum_{i=1}^{m} a_i + \sum_{i=0}^{m} c_i$.

For machines $k+1 \leq i \leq m$, we have $(a_i + c_i)/s_i \geq c_i/s_i \geq \frac{1}{m}$ and are done.

Let $j = \arg\max_i a_i$. We have $\sum_{i=0}^{k} c_i < (k+1) \cdot \frac{s_k}{m} \leq s_k$ since $k < m$. There are at least k machines of OPT that have no jobs in the set $\{c_{k+1}, \ldots, c_m\}$, so $\sum_{i=1}^{m} a_i + \sum_{i=0}^{k} c_i \geq s_1 + \cdots + s_k$, or $\sum_{i=1}^{m} a_i \geq s_{k-1}$ (using the property $k > 1$) and therefore $a_j \geq s_{k-1}/m$. As before, we have $a_i + c_i \geq a_j$ for any machine $1 \leq i \leq m$, so for $1 \leq i \leq k-1$, we find $(a_i + c_i)/s_i \geq a_j/s_{k-1} \geq 1/m$.

However, $a_k + c_0 \geq a_j$, since the assignment of the last job of machine j (excluding c_j) to machine k would have increased the total size of jobs assigned to it to at least a_j, and c_0 is no smaller than that job, so

$$a_k + c_0 + c_k \geq a_j + c_k \geq \frac{\sum_{i=1}^{m} a_i}{m} + \frac{\sum_{i=0}^{k} c_i}{k+1} \geq \frac{s_k}{m},$$

since $a_j \geq a_i$ for all $1 \leq i \leq m$, $c_k \geq c_i$ for all $0 \leq i \leq k$, and $k+1 \leq m$.

Case 3: $k = 1$. We have $c_i \geq s_i/m$ for $i = 2, \ldots, m$, so we only need to consider machine 1. Consider the first assignment, and use the notations a_i as in the first case. At least one machine of OPT has no jobs of $c_2, \ldots, c_m$, so $\sum_{i=1}^{m} a_i + c_1 \geq s_1 = \min_i s_i$. We have $a_1 + c_1 \geq a_i$ for $2 \leq i \leq m$, so $a_1 + c_1 \geq s_1/m$ and $(a_1 + c_1)/s_1 \geq 1/m$. □

4 Restricted assignment

4.1 Lower bounds

We first state the lower bounds which we show for the case of restricted assignment.

Theorem 6. *For a buffer of size at most $m-2$, no algorithm can have a finite approximation ratio in the restricted assignment setting, that is, a buffer of size $\Omega(m)$ is required for a finite approximation ratio.*

Theorem 7. *For any finite-sized buffer, any online algorithm has an approximation ratio of at least m.*

4.2 An algorithm for restricted assignment with a finite-sized buffer

We now present an algorithm for restricted assignment. At each time, for every machine i, we keep the m largest jobs which can be processed on i, in the buffer. Every job which is not in this set (or a job which stops being in this set) is assigned in a greedy fashion to a least loaded machine which can process it. Thus, the required size of the buffer is m^2. Assume $\text{OPT} > 0$, so every machine has at least one job that can be assigned to it.

At the end of the input we assign the jobs from the buffer in an optimal way (i.e., we test all possible assignments and pick the best one). Note that it is possible to reduce the number of tested assignments while maintaining optimality. We will analyze a fixed assignment which depends on a (fixed) optimal solution OPT, which we define next.

Let S be the set of machines such that OPT has at least one job of size $\text{OPT}/(2m)$ assigned to it. Then, we would like to pick for every machine in S one job from the buffer of size at least $\text{OPT}/(2m)$, and assign it to that machine. Note that a machine having less than m jobs which it can process must belong to S, since in the optimal solution it has less than m jobs assigned to it, the largest of which has a size of at least $\text{OPT}/(m-1)$.

Lemma 6. *Such an assignment of jobs of size at least $\text{OPT}/(2m)$ is possible.*

Proof. Let j_i be the largest job (of size at least $\text{OPT}/(2m)$) assigned to machine i by OPT. If j_i is in the buffer, then we assign it to machine i. We do this for every $i \in S$. So far jobs were only assigned to their machines in OPT, so each job was assigned to at most one machine.

At the end of this process there might be additional machines in which we planned to assign to each such machine is not in the buffer. Therefore, the reason that for such a machine i we did not assign jobs is that there are more than m jobs which can be processed by i and which are larger than j_i. This means that i is a machine which initially (after the termination of the input) has m jobs in the buffer which can be assigned to it. At least one such job was not assigned before (since at most $m-1$ jobs from the buffer have been assigned), so we can assign it to i. Applying this procedure for one machine at a time, we will get an allocation of one job for each machine in S and such a job has size at least $\text{OPT}/(2m)$ as we required. □

After dealing with the set S, we continue to allocate one job to each machine not in S. We apply the following rule, for each machine $i \notin S$ we allocate to i the largest job in the buffer, which can be run on i, and which was not allocated before. We again assign a job to one machine at a time. We can always allocate a job from the buffer to i because initially there are at least m jobs which can be processed by i, and every machine is allocated exactly one job from the buffer.

Lemma 7. *If machine $i \notin S$ is allocated a job j, then for every job j' of size $p_{j'}$, which can be processed on machine i and is either still in the buffer after each machine received a job out of the buffer, or was allocated by the list scheduling algorithm before the input terminated, we have $p_j \geq p_{j'}$.*

Proof. Job j is larger than any job which is left in the buffer after every machine was allocated one job out of those left in the buffer (taking into account only the jobs which can be processed by machine i). The jobs assigned greedily before the final step are no larger than j, since the buffer keeps the m largest jobs which i can receive. □

The jobs remaining in buffer, after each machine received a job out of the buffer, are assigned one by one, so that each job is assigned greedily to the least loaded machine that can process it. We say that a job j is small if it was allocated greedily (either at the earlier stage or after the input ends). Other jobs are called large. Therefore, the algorithm has allocated exactly one large job for each machine, and if OPT has assigned a job of size at least $\text{OPT}/(2m)$ to machine i, then the large job of i is of size at least $\text{OPT}/(2m)$.

Theorem 8. *Every machine is allocated a load of at least* $\text{OPT}/(2m)$.

Proof. A machine $i \in S$ is allocated a large job of size at least $\text{OPT}/(2m)$ as we discussed above. Hence, it suffices to consider a machine $i \notin S$ whose large job is of size less than $\text{OPT}/(2m)$. Fix such a machine i. For every machine $j \neq i$ we denote by C_j the set of jobs which OPT assigns to i and the algorithm assigns to j. Note that C_j may contain a large job, but in this case the large job is of size at most $\text{OPT}/(2m)$ (as otherwise $i \in S$ which is a contradiction).

We consider the total size of small jobs in C_j. Denote by x the last small job from the set C_j assigned to j. Note that by the greedy fashion in which jobs are assigned we conclude that if we discard x from C_j, the total size of remaining small jobs in C_j is at most the total size of small jobs assigned to i (as otherwise the algorithm would not assign x to j). Recall that the large job of i is at least as large as x, and hence we conclude that the total size of small jobs of C_j is at most the total assigned jobs to machine i. Summing up over all j we conclude that the total size of small jobs which OPT assigns to machine i and the algorithm assigns to other machines is at most $m-1$ times the load of machine i in the solution returned by the algorithm. The total size of large jobs which OPT assigns to machine i is at most $\text{OPT}/2$ (as each such job has size less than $\text{OPT}/(2m)$). Hence, the total size of small jobs which OPT assigns to machine i is at least OPT/2. Therefore, the algorithm assigns at least $\text{OPT}/(2m)$ to machine i, and the claim follows. □

References

1. S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.
2. A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proc. 39th Symp. Theory of Computing (STOC)*, pages 114–121, 2007.

3. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44(3):486–504, 1997.
4. Y. Azar and L. Epstein. On-line machine covering. In *Proc. 5th European Symp. on Algorithms (ESA)*, pages 23–36, 1997.
5. Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18(2):221–237, 1995.
6. N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2006.
7. P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35:108–121, 2000.
8. S.-Y. Cai. Semi-online machine covering. *Asia-Pacific J. of Oper. Res.*, 24(3):373–382, 2007.
9. O. Chassid and L. Epstein. The hierarchical model for load balancing on two machines. *Journal of Combinatorial Optimization*, 15(4):305–314, 2008.
10. B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18:127–131, 1995.
11. J. Csirik, H. Kellerer, and G. Woeginger. The exact LPT-bound for maximizing the minimum completion time. *Operations Research Letters*, 11:281–287, 1992.
12. B. L. Deuermeyer, D. K. Friesen, and M. A. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIAM Journal on Discrete Mathematics*, 3(2):190–196, 1982.
13. Gy. Dósa and L. Epstein. Online scheduling with a buffer on related machines. *Journal of Combinatorial Optimization*. To appear, DOI: 10.1007/s10878-008-9200-y.
14. Gy. Dósa and L. Epstein. Preemptive online scheduling with reordering. In *"The 17th Annual European Symposium on Algorithms (ESA2009)*, pages 456–467, 2009.
15. T. Ebenlendr, J. Noga, J. Sgall, and G. J. Woeginger. A note on semi-online machine covering. In *Approximation and Online Algorithms, 3rd International Workshop (WAOA2005)*, pages 110–118, 2005.
16. M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *Proc. 48th Symp. Foundations of Computer Science (FOCS)*, pages 603–612, 2008.
17. L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004.
18. D. K. Friesen and B. L. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Mathematics of Operations Reasearch*, 6(1):74–87, 1981.
19. 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 (SODA)*, pages 564–565, 2000.
20. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Sys. Tech. J.*, 45:1563–1581, 1966.
21. Y. Jiang, Z. Tan, and Y. He. Preemptive machine covering on parallel machines. *Journal of Combinatorial Optimization*, 10(4):345–363, 2005.
22. H. Kellerer, V. Kotov, M. G. Speranza, and Zs. Tuza. Semi online algorithms for the partition problem. *Operations Research Letters*, 21:235–242, 1997.
23. Z. Tan and Y. Wu. Optimal semi-online algorithms for machine covering. *Theoretical Computer Science*, 372(1):69–80, 2007.
24. G. J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20(4):149–154, 1997.
25. G. Zhang. A simple semi on-line algorithm for $P2//C_{\max}$ with a buffer. *Information Processing Letters*, 61:145–148, 1997.