On-Line Machine Covering

Yossi Azar[1], Leah Epstein[2]

[1] Dept. of Computer Science, Tel-Aviv University. ***
[2] Dept. of Computer Science, Tel-Aviv University. †

Abstract. We consider the problem of scheduling a sequence of jobs to m parallel machines as to maximize the minimum load over the machines. This situation corresponds to a case that a system which consists of the m machines is alive (i.e. productive) only when all the machines are alive, and the system should be maintained alive as long as possible. It is well known that any on-line deterministic algorithm for identical machines has a competitive ratio of at least m and that greedy is an m competitive algorithm. In contrast we design an on-line randomized algorithm which is $\tilde{O}(\sqrt{m})$ competitive and a matching lower bound of $\Omega(\sqrt{m})$ for any on-line randomized algorithm. In the case where the jobs are polynomially related we design an optimal $O(\log m)$ competitive randomized algorithm and a matching tight lower bound for any on-line randomized algorithm. In fact, if F is the ratio between the largest job and the smallest job then our randomized algorithm is $O(\log F)$ competitive.
A sub-problem that we solve which is interesting by its own is the problem where the value of the optimal algorithm is known in advance. Here we show a deterministic (constant) $2 - \frac{1}{m}$ competitive algorithm. We also show that our algorithm is optimal for two, three and four machines and that no on-line deterministic algorithm can achieve a better competitive ratio than 1.75 for $m \geq 4$ machines.
For related machines we show that there is no on-line algorithm, whose competitive ratio is a function of the number of machines. However, for the case where the value of the optimal assignment is known in advance, and for the case where jobs arrive in non increasing order, we show that the exact competitive ratio is m. We show a constant 2 competitive algorithm for the intersection of the above two cases, i.e. the value of the optimal assignment is known in advance and the jobs arrive in non increasing order.

1 Introduction

We consider the problem of scheduling a sequence of jobs to m parallel machines as to maximize the minimum load over the machines. This situation is motivated by the following scenario. A system consists of m (identical or related)

*** E-Mail: azar@math.tau.ac.il. Research supported in part by Allon Fellowship and by the Israel Science Foundation administered by the Israel Academy of Sciences.
† E-Mail: lea@math.tau.ac.il.

machines. The system is alive (i.e. productive) only when all the machines are alive. In order to keep a machine alive it requires resources (e.g. tanks of fuel). The various size resources arrive one after the other, and each resource should be assigned immediately upon its arrival to one of the machines. The goal is clearly to keep the system alive as long as possible. The above problem has applications also in the sequencing of maintenance actions for modular gas turbine aircraft engines [15]. To conform with the standard scheduling terminology we view the resources as jobs. Thus, jobs are assigned to machines as to maximize the minimum load. If all the machines are identical then the problem corresponds to the identical machines scheduling/load-balancing problem and if each machine has its own size (of the engine that operates on the fuel) then it corresponds to the related machines problem.

We give a formal definition of the problem discussed above. Consider a set of m identical machines and a set of jobs that arrive on-line. Each job j has a weight w_j. The load of a machine i is the sum of the weights of the jobs assigned to it. That is, $l_i = \sum_{j \in J_i} w_j$, where J_i is the of jobs assigned to machine i. If machine i has a speed v_i (related machines case) then $l_i = \sum_{j \in J_i} \frac{w_j}{v_i}$. The goal is to assign the jobs to the machines as to maximize the minimum load over the machines. The problems are on-line versions of classical covering problems and are called the machine covering problems. Note that these problems are different from the bin covering problems [2, 4, 5, 13] where the goal is to maximize the number of covered bins, i.e. bins of load of at least 1.

We use the standard definition of the competitive ratio. Denote by $V_{on}(\sigma)$ (or just V_{on}) the value of the on-line algorithm for a sequence σ which is the load incurred on the least loaded machine. The value of the optimal assignment for this sequence would be denoted by $V_{opt}(\sigma)$ (or justV_{opt}). The algorithm is c competitive (the competitive ratio is c) if for every sequence σ, $V_{opt}(\sigma) \leq c \cdot V_{on}(\sigma)$.

Known results: The off-line problem of maximizing the load of the least loaded machine, is known to be NP-complete in the strong sense [17]. Woeginger [22] designed a polynomial time approximation scheme for the identical machines case. He also showed that the greedy algorithm is m competitive. (It is well known that no deterministic algorithm can achieve a better competitive ratio.) Deuermeyer, Friesen and Langston [14] studied a semi on-line problem on identical machines. They examined the LPT-heuristic which order the jobs by non increasing weights and assigns each job to the least loaded machine at the moment. It is shown in [14] that the competitive ratio of this heuristic is at most $\frac{4}{3}$. The tight ratio $\frac{4m-2}{3m-1}$ is given by Csirik, Kellerer and Woeginger [12].

Our Results: We consider the on-line version of the machine covering problems. We show the following results for the identical machines case.

- There is a randomized $O(\sqrt{m}\log m)$ competitive algorithm and any randomized algorithm is at least $\Omega(\sqrt{m})$ competitive. This is in contrast to the competitive ratio of the best possible deterministic algorithm which is m.
- There is a randomized $O(\log F)$ competitive algorithm for jobs of weights which may vary up to a factor of F. In particular, there is an $O(\log m)$

competitive randomized algorithm for polynomially related weight jobs. (The deterministic lower bound of m holds already for jobs of weights that are polynomially related.) Also, any randomized algorithm is at least $\Omega(\log m)$ competitive for polynomially related weight jobs.

A sub-problem that we solve which is interesting by its own is the problem where the value of the optimal algorithm is known in advance. Here we show the following results:

- There is a deterministic $2 - \frac{1}{m}$ competitive algorithm.
- The algorithm is optimal for two, three and four machines and no on-line deterministic algorithm can achieve a better competitive ratio than 1.75 for $m \geq 4$ machines.

For related machines we show the following results

- There is no algorithm whose competitive ratio is a function of the number of machines.
- For the case where the value of the optimal assignment is known in advance the exact competitive ratio is m.
- For the case where jobs arrive in non increasing order the exact competitive ratio is also m.
- For the case where both the value of the optimal assignment is known in advance and jobs arrive in non increasing order there is 2 competitive algorithm.

Other related work: The original scheduling problem of minimizing the maximum load over all machines has been widely studied. The problem was introduced by Graham [18, 19] who gave a greedy algorithm "List Scheduling" which is $2 - \frac{1}{m}$ competitive. It turns out that the algorithm of Graham is not optimal for minimizing the maximum load (for all $m \geq 4$) [16, 11]. Bartal et al. [8] were the first to show an algorithm whose competitive ratio is strictly below $c < 2$ (for all m). More precisely, their algorithm achieves a competitive ratio of about $2 - \frac{1}{70}$. Later, the algorithm was generalized by Karger, Phillips and Torng [20] to yield an upper bound of 1.945. Very recently, Albers [1] designed 1.923 competitive algorithm and improved the lower bound to 1.852 (the previous lower bound was 1.8370 [7]). Clearly randomized algorithms for the problem may improve the bounds only by a constant factor. Unlike the scheduling problem, there are no constant competitive algorithms for the covering problem on identical machines. Moreover, as we mentioned our results show that the competitive ratio of randomized algorithms are significantly better than the competitive ratio of the deterministic ones.

For the related machines case it was proved in [3] that there is a constant 8 competitive algorithm for minimizing the maximum load. (the constant was recently improved by [9].) If the value of the optimal assignment is known in advance then the competitive ratio is 2. This is in contrast to the "more difficult" covering problem where in order to get comparable results one need to assume that the jobs arrive in non-increasing order and the optimal value is known in advance.

It is interesting to note that a measure which is different than the maximum load has been already used for scheduling problems. Specifically, Awerbuch et al.[6] studied the case of minimizing the sum of the squares of the load (and general L_p norm). It is shown that there is a constant competitive algorithm for a general class of scheduling problems.

Structure of the paper: In section 2 we consider the sub-problem where the value of the optimal assignment is known in advance. In section 3 we use an algorithm from section 2 as a procedure, and discuss randomized algorithms for identical machines. In section 4 we consider algorithms for related machines.

2 Known optimal value

In this section we show a simple algorithm for identical machines for the case where the value of the optimal assignment is known in advance. We assume without loss of generality that $V_{opt} = 1$. The competitive ratio of our algorithm is $2-\frac{1}{m}$, which is the same ratio as for the algorithm of Graham for scheduling. In contrast to the algorithm of Graham which tries to fill all the machines evenly, our algorithm fills the machines one by one. We define $\beta = \frac{m}{2m-1}$ and call a machine of load at least β a full machine. A non empty machine which is not full is called active. Our algorithm maintains at most one active machine.

Algorithm FILL:

If there are no empty machines, assign a new job to the least loaded machine. Otherwise, if the weight of the new job is at least β assign it to an empty machine (the machine becomes full). If the weight of the job is less than β assign it to the active machine if exists, and otherwise to an empty machine (which becomes active).

Theorem 1. *The algorithm Fill has a competitive ratio of* $\frac{1}{\beta} = 2 - \frac{1}{m}$.

Proof. We show that when the algorithm terminates, all machines are full. This implies that the competitive ratio is at most $\frac{1}{\beta}$. Assume that there is a non full machine. First we may replace all jobs of weight more than 1 by jobs of weight 1, which would not influence the optimal algorithm or our algorithm. We show that all jobs of weight at least β were assigned to empty machines. Otherwise the first such job that was assigned to a non empty machine was assigned to the only active machine, and after it was assigned all machines becomes full which contradicts the assumption. We estimate the load of each machine. Denote the load of the least loaded machine by h. The load of machines that have only one job is clearly at most 1. If there are at least two jobs on a machine, its load is bounded by 2β, since just before the last job was placed the load was less than β and the weight of the last job is also less than β. Thus the load of each machine is bounded by 2β. Since the optimal value is 1 then

$$h \geq m - (m-1) \cdot 2\beta \geq m - (m-1)\frac{2m}{2m-1} = \frac{m}{2m-1} = \beta$$

which completes the proof.

Notice that later we will use algorithm Fill with $\beta' \leq \beta$ which is clearly $\frac{1}{\beta'}$ competitive. The algorithm Fill with the parameter β turns out to be optimal for two, three and four machines.

Theorem 2. *The competitive ratio of any algorithm for two machines is at least* 1.5 .

Proof. We consider the following sequence of jobs. First two jobs of weight 2/3 arrive. There are two cases:

- If both jobs were placed on one machine, then two jobs of weight 1/3 arrive. The optimal algorithm assigns one small job and one big job on each machine. The best value the on-line is 2/3 by assigning both small jobs to the empty machine. Thus the competitive ratio in this case is at least 1.5 .
- If each job was placed on a different machine, then one job of weight 1 arrives. The optimal algorithm assignment the two jobs of weight 2/3 on one machine, and the unit job on the other machine. The on-line places the unit job on one machine, and the load of the other machine is still 2/3. The competitive ratio in this case is also at least 1.5 .

Theorem 3. *The competitive ratio of any algorithm for three machines is at least* 5/3 .

Proof. We construct a sequence such that $V_{on} \leq 0.6$ and thus the competitive ratio is at least 5/3. We consider the following sequence of jobs. First two jobs of weight 0.6 arrive. If the on-line algorithm assigns the jobs on different machines, two unit jobs arrive. The minimum load of the on-line is 0.6 while the optimal algorithm assigns the jobs of weight 0.6 on one machine, and one unit job on each other machine.

If the on-line algorithm assigns them to the same machine, we continue the sequence by two jobs of weight 0.3 . Clearly, the on-line will not use the machine with the first two jobs any more, because it already has load of at least 1. Thus we are left with two machines. There are two cases:

- If the two jobs of weight 0.3 are placed on different machines, then two jobs of weight 0.1 and one unit job arrive. Clearly, the best value the on-line can obtain is 0.5 (by assigning the unit job to one of the two machines, and both jobs of weight 0.1 to the other). The optimal algorithm assigns the unit job on one machine, and three jobs of weights $0.6, 0.3, 0.1$ on each of the other two machines and achieves a minimum load of 1.
- If the two jobs of weight 0.3 are placed on the same machine, then a job of weight 0.6 arrives. If it is placed on the only empty machine, a unit job arrives, and since there are two on-line machines with load 0.6, $V_{on} = 0.6$. The optimal algorithm assigns two jobs of weight 0.6 on one machine, the unit job on another machine and all the other jobs on the third machine and thus achieves a minimum load of 1. If the job of weight 0.6 is placed on the

machine with load 0.6, three jobs with the weights $0.4, 0.1, 0.1$ arrive. Even if they are all placed on the third machine, $V_{on} = 0.6$. The optimal algorithm assigns a job of weight 0.6 with a job of weight 0.4 on one machine and a job of weight 0.6 with two jobs of weights 0.1 and 0.3 on each of the other two machines. Thus the competitive ratio is at least $5/3$ for all the cases.

The proof of the following theorem is omitted.

Theorem 4. *The competitive ratio of any algorithm for four machines is at least* 1.75 *.*

We give a general lower bound for $m > 4$ machines by noticing that the competitive ratio of the problem is non-increasing as a function of the number of machines.

Lemma 5. *Let r be a lower bound for k machines, then r is also a lower bound for $m > k$ machines.*

Proof. We assume that r is a lower bound for k machines. We convert it into a lower bound for m machines, where $m > k$. We begin the lower bound with $m - k$ unit jobs. After those jobs arrive, we continue with the original lower bound for k machines. The on-line and the off-line algorithms both assign the first $m-k$ jobs to $m-k$ different machines and do not use those machines later. Thus, the competitive ratio of any algorithm for $m > k$ machines is at least r.

Corollary 6. *The competitive ratio of any algorithm for $m \geq 4$ machines is at least* 1.75 *.*

3 The identical machines case

We recall that no deterministic algorithm can achieve a better competitive ratio than m (consider a sequence of m jobs of weight 1 which may be followed with $m-1$ jobs of weight m) and that greedy is m competitive. Here we show that randomization really helps in reducing the competitive ratio. We first introduce an algorithm whose competitive ratio depends on the maximum ratio between the largest job and the smallest job. We denote this ratio by F. We show an $O(\log F)$ competitive algorithm for the case that F is known in advance. If F is not known in advance we can apply the standard technique of [21] and get an $O(\log^{1+\varepsilon} F)$ competitive algorithm.

Lemma 7. *Assume that it is known in advance that $\lambda \leq V_{opt} < 2\lambda$. Then algorithm Fill with parameter λ as the known optimal value and the parameter $\beta' = \frac{1}{2}$ is 4 competitive.*

Proof. The results of running the algorithm Fill with these parameters, is the same as running the algorithm with the correct value of V_{opt} and $\beta'' = \frac{\lambda}{2V_{opt}}$. Since $\frac{1}{4} < \beta'' \leq \frac{1}{2}$ then by Theorem 1 the algorithm succeeds and is 4 competitive.

Our randomized algorithm normalizes the weight of the first job to 1. Thus the weights of the jobs are bounded between F and $\frac{1}{F}$. We first define two procedures that the algorithm uses.

Procedure RFILL:

The procedure chooses an integer i^* uniformly at random, where 2^{i^*} is in the range $\frac{1}{2F} \leq 2^{i^*} \leq 2F$ (Assume that F is known in advance). Then the procedure runs the algorithm Fill with the value 2^{i^*} as the known optimal value and $\beta' = 1/2$.

Lemma 8. *The procedure RFill is $O(\log F)$ competitive if $F \geq V_{opt}/2$.*

Proof. Let us bound the value of V_{opt}. If $V_{opt} = 0$, then the lemma is trivially correct. Thus we may assume that the optimal algorithm has at least one job on each machine, and since the weight of a job is at least $\frac{1}{F}$ then $\frac{1}{F} \leq V_{opt} \leq 2F$. For each possible value of V_{opt}, a suitable value of i^* such that $2^{i^*} \leq V_{opt} < 2^{i^*+1}$ was chosen with probability $O(\frac{1}{\log F})$. Hence the procedure is $O(\log F)$ competitive. If F is not known in advance we choose an integer value i, $-\infty < i < \infty$ with probability $\frac{1}{(|i|)^{1+\epsilon}+1}$ and achieve competitive ratio of $O(\log^{1+\epsilon} F)$.

Procedure GREEDY:

Assign each new job to a machine that has the current minimum load.

Lemma 9. *The procedure is 2 competitive if $F \leq V_{opt}/2$*

Proof. Denote by h the minimum on-line load, by H the maximum on-line load and by x a machine with load H. Clearly $V_{opt} \leq H$ and $V_{on} = h$. Let Y be the last job assigned to x. By the definition of F, $w(Y) \leq F$. Clearly at the time that Y was assigned, x had the minimum load and hence $H - w(Y) \leq h$. Thus

$$H - h \leq w(Y) \leq F \leq V_{opt}/2 \leq H/2$$

which implies that $h \geq H/2$ and $\frac{V_{opt}}{V_{on}} \leq \frac{H}{h} \leq 2$.

Algorithm RFILL-GREEDY: Run RFill with probability $1/2$ and Greedy with probability $1/2$.

Theorem 10. *The algorithm RFill-Greedy is $O(\log F)$ competitive.*

Proof. The proof follows immediately from the proofs of Lemmas 8 and 9.

The above theorem immediately implies the following:

Corollary 11. *The algorithm RFill-Greedy is $O(\log m)$ competitive for polynomially related weight jobs.*

In fact, we cannot achieve a better bound up to a constant factor.

Theorem 12. *Any randomized algorithm for a sequence for which the ratio between the largest and the smallest jobs is at least m is $\Omega(\log m)$ competitive.*

Proof. We use an adaptation of Yao's theorem for randomized on-line algorithms which states that a lower bound on the competitive ratio of deterministic algorithms on any distribution on the input is also a lower bound for randomized algorithms, and is given by $\max\{1/E\left(V_{on}/V_{opt}\right), E(V_{opt})/E(V_{on})\}$ [10]. Assume for simplicity that $m = 2^k$. Consider the following sequence. At first m jobs of weight 1 arrive. After those unit jobs are placed then $m - 2^{k-i^*}$ big jobs of weight 2^{i^*} arrive, where $0 \leq i^* \leq k$ is an integer chosen uniformly at random. The optimal off-line assignment would be to put one big job on each machine, and 2^{i^*} unit jobs on the remaining 2^{k-i^*} machines, yielding the value 2^{i^*}. It is easy to see that the best strategy for the on-line algorithm is to assign the big jobs to the least loaded machines. After the first phase, we sort the machines in non decreasing order, according to the number of jobs assigned to each machine: $M_1, M_2, ..., M_m$, where M_1 is the most loaded machine. Let $a_i, 0 \leq i \leq k$ be

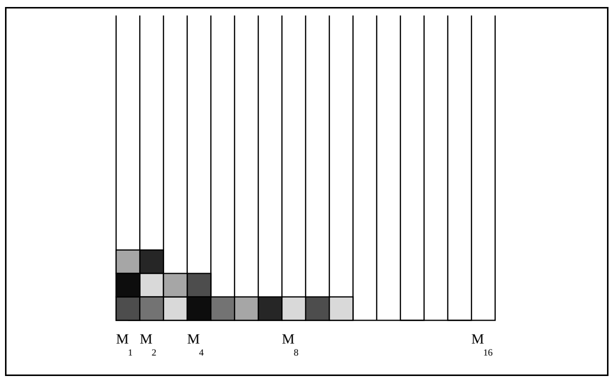

Fig. 1. The machines sorted by non increasing load

the number of jobs on the machine M_{2^i}. Since there are m unit jobs, and the machines are sorted, $a_0 + \sum_{1 \leq i \leq k} a_i \cdot (2^i - 2^{i-1}) \leq m$. Now we can estimate the value of the on-line algorithm for each one of the $k+1$ cases. For a fixed value of i^*, there are $m - 2^{k-i^*}$ big jobs, that are placed on the least loaded machines, which implies a load of at least 2^{i^*} on those machines. After that, the least loaded machine is $M_{2^{k-i^*}}$, which has load a_{k-i^*}. The competitive ratio in this case is $\frac{2^{i^*}}{a_{k-i^*}}$. Hence,

$$E\left(\frac{V_{on}}{V_{opt}}\right) \leq \frac{1}{k+1} \sum_{0 \leq i \leq k} \frac{a_{k-i}}{2^i} = \frac{1}{k+1} \sum_{0 \leq i \leq k} \frac{a_i}{2^{k-i}}$$

$$= \frac{1}{2^{k-1}(k+1)} \sum_{0 \leq i \leq k} a_i 2^{i-1} \leq \frac{1}{2^{k-1}(k+1)} \left(a_0 + \sum_{1 \leq i \leq k} a_i \cdot 2^{i-1}\right)$$

$$\leq \frac{1}{2^{k-1}(k+1)} \cdot m = \frac{2}{k+1}$$

Thus, by Yao's theorem the competitive ratio is at least $\frac{k+1}{2} = \Omega(\log m)$.

Next we provide an algorithm whose performance is as a function only of the number of machines. We introduce a randomized algorithm Partition which is $O(\sqrt{m}\log m)$ competitive. Assume without loss of generality that m is a power of 4. We round down the weights of jobs in the sequence to powers of 2. Clearly the competitive ratio of a general weight sequences is at most twice the competitive ratio of power of 2 sequences. Next with probability $1/2$ we apply the procedure Greedy and with probability $1/2$ we choose $0 \leq i \leq \frac{1}{2}\log m$ uniformly at random and apply the following partitioning procedure with $k = 2^i$. The procedure partitions the machines into two parts. The left part consists of k machines and the right part consists of the remaining machines. The idea of the procedure is to classify each new job to one of the sets, and assign it to a machine in this set. The procedure keeps a guess λ for the value of $\frac{V_{opt}}{\sqrt{m}}$, which is initialized to 0. Whenever a job j arrives, if $\lambda \geq w(j)$, the job is considered below threshold, and would be assigned to the left k machines. Otherwise, with probability $\frac{1}{2m}$ the procedure decides to increase λ to be $w(j)$, and the job which becomes below threshold is assigned to the left machines. If λ was not changed, the job is assigned to the right machines. Jobs on the right machines are assigned greedily, i.e. on a least loaded right machine. Jobs on the left machines are placed in a round Robin manner for each weight separately.

Theorem 13. *The algorithm Partition is $O(\sqrt{m}\log m)$ competitive.*

Proof. We will prove the following claim: For every sequence, either greedy is $O(\sqrt{m})$ competitive or there exists a choice of k, for which the partitioning procedure is $O(\sqrt{m})$ competitive. Since there are $O(\log m)$ choices of k, all with equal probability, the algorithm is $O(\sqrt{m}\log m)$ competitive.

For a fixed sequence we define a job j as big if $w(j) \geq \frac{V_{opt}}{8\sqrt{m}}$ and otherwise small. Let r be the number of big jobs in the sequence and $k_1 = \max(0, m-r)$.

We first show that if $k_1 = 0$, the simple greedy algorithm yields the desired result. We show that the load of every machine is at least $\frac{V_{opt}}{8\sqrt{m}}$. The definition of k_1 implies that there are at least m big jobs. If at least one big job was placed on each machine, the claim on the load holds. Otherwise, there a machine with at least two big jobs. At the moment the second job was placed, this machine had the minimum load among all machines, which is at least the weight of one big job, and thus all machines had load of at least $\frac{V_{opt}}{8\sqrt{m}}$.

Next we show that if $k_1 > \sqrt{m}$, the simple greedy algorithm also yields the desired result. Here we show that the small jobs give enough load for all machines. Consider the greedy assignment. Assume that machine z has minimum load h. Since the algorithm is greedy, if we remove the latest job from all other machines, they all would have loads that do not exceed h. We bound the total load achieved from small jobs. It is h for machine z and h plus one small job for all other machines. The total is bounded by $hm+(m-1)\cdot V_{opt}/(8\sqrt{m})$. Since the total load of small jobs is at least $k_1 V_{opt} \geq \sqrt{m}V_{opt}$, we conclude that $h \geq \frac{V_{opt}}{2\sqrt{m}}$.

Finally we show that there is an appropriate choice of k for all cases $0 < k_1 \leq \sqrt{m}$. Assume $2^{i_1} < k_1 \leq 2^{i_1+1}$. We show that the choice $k = 2^{i_1+1}$ yields the load $\Omega\left(\frac{V_{opt}}{\sqrt{m}}\right)$ with constant probability. Notice that for this choice $2k_1 > k \geq k_1$. We first show that with constant probability all the big jobs, were placed on the right machines. In order for some of the big jobs to be placed on the left machines, the value λ should have been changed for at least one of them. The probability that λ was not changed for a single job is $1-1/(2m)$. The probability that for $r \leq m$ big jobs λ was not changed is $(1-\frac{1}{2m})^r \geq (1-\frac{1}{2m})^m \geq \frac{1}{2}$. Thus, with probability at least $1/2$, r big jobs are placed on the $m-k$ right machines. Since $m-k \leq m-k_1 = r$ the load of each right machine is at least $\frac{V_{opt}}{8\sqrt{m}}$. Next we estimate the expected minimum load on the left machines. We first prove the following lemma:

Lemma 14. *The expected minimum load on the left machines induced by the small jobs, is at least $\frac{V_{opt}}{32\sqrt{m}}$ for the appropriate choice of k.*

Proof. There are at most $2k_1$ left machines. We denote the total weight of the small jobs by W. We define a success for a set of $2\sqrt{m}$ jobs of equal weight w by the event that after the arrival of the first $\sqrt{m}$ jobs, the value of λ is at least w (and thus the last $\sqrt{m}$ jobs were below threshold, and were assigned to the left machines).

The failure probability for a set of $2\sqrt{m}$ jobs is most $(1-\frac{1}{2m})^{\sqrt{m}} \leq e^{-\frac{1}{2\sqrt{m}}}$. Since for $0 \leq x \leq 1$, $e^{-x} \leq 1-\frac{x}{2}$, the failure probability is at most $1-\frac{1}{4\sqrt{m}}$. Hence the success probability in a set of $2\sqrt{m}$ jobs is at least $\frac{1}{4\sqrt{m}}$. If there was a success in a subsequence of $2\sqrt{m}$ jobs of equal weight, then at least $\sqrt{m}$ jobs were assigned to all left machines, thus at least $\frac{\sqrt{m}}{k}$ jobs to each one of the left machines. We partition all small jobs into sets of size $2\sqrt{m}$. The last set, which may contain less than $2\sqrt{m}$ jobs, is ignored. The weight that is lost by ignoring those sets is bounded by

$$2\sqrt{m}\sum_{j\geq 0}\frac{w_{max}}{2^j} \leq 4\sqrt{m}w_{max} \leq 4\sqrt{m}\cdot\frac{V_{opt}}{8\sqrt{m}} \leq \frac{V_{opt}}{2}$$

where w_{max} is the weight of the largest small job. Denote the set of sets that are not ignored by A, and the total weight of A by W_A. Since the total weight of small jobs is at least V_{opt} then $W_A \geq \frac{W}{2}$. The expected of the minimum load over the left machines is at least

$$\frac{1}{4\sqrt{m}}\sum_{a\in A}\frac{\sqrt{m}w_a}{k} = \frac{1}{8k\sqrt{m}}\sum_{a\in A}2\sqrt{m}w_a \geq \frac{1}{8k\sqrt{m}}W_A \geq \frac{1}{8k\sqrt{m}}\frac{W}{2}$$

Clearly $W \geq k_1 V_{opt}$ and hence the expected minimum load on the left machines is at least

$$\frac{1}{16k\sqrt{m}}k_1 V_{opt} \geq \frac{k}{32k\sqrt{m}}V_{opt} \geq \frac{V_{opt}}{32\sqrt{m}}$$

We conclude that with probability of at least $1/2$ the expected minimum load over all the machines is $\Omega\left(\frac{V_{opt}}{\sqrt{m}}\right)$ and thus the partitioning procedure is $O(\sqrt{m})$ competitive, for the appropriate choice of k.

Surprisingly, no algorithm can achieve a significantly better competitive ratio.

Theorem 15. *Any randomized algorithm is $\Omega(\sqrt{m})$ competitive.*

Proof. By Yao's theorem it is enough to show a lower bound for some distribution on the input on $\max\{1/E\left(V_{on}/V_{opt}\right), E(V_{opt})/E(V_{on})\}$.

Assume for simplicity that $m = k^2 \geq 4$. The first phase of the sequence contains k parts, each of k jobs. All the jobs of part i have equal weight of $s_i = k^{2i}$ (for $1 \leq i \leq k$). In the second phase, an integer $1 \leq i^* \leq k$ is chosen uniformly at random, and additional $ki^* - 1$ jobs of weight k^{2i^*+1} arrive. Clearly the sequence can be assigned by the optimal algorithm so that the load of each machine is at least k^{2i^*+1}. This can be done since there are $k^2 - 1$ jobs of weight at least k^{2i^*+1} and k jobs of weight k^{2i^*}.

We consider the on-line assignment after the first phase. We sort the on-line machines in non decreasing order according to the load on each machine: $M_1, M_2, ..., M_m$, where M_1 is the most loaded machine. Notice that for each i, a job of weight s_i is larger than the sum of all smaller jobs, since

$$k^{2i} > 2k \cdot k^{2(i-1)} \geq k(k^{2(i-1)} + k^{2(i-2)} + ... + 1) \ .$$

Thus the largest ki jobs are assigned to machines $M_1, ..., M_{ki}$, for all $1 \leq i \leq k$ (not necessary all of them). All the jobs of weight s_{k-i+1} jobs are assigned to machines $M_1, ..., M_{ki}$. The largest weight of a job on machine M_{1+ki} is at most s_{k-i}. Denote by b_i the number of jobs of weight s_{k-i+1} jobs on the machine $M_{1+k(i-1)}$, and by l_i the load of this machine. Notice that $l_i \leq (b_i + 1)s_{k-i+1}$.

Next we show that $B = b_1 + b_2 + ... + b_k \leq 3k$. Let j be the minimum index (if exists) such that $\sum_{i<j}(b_i - 1) \geq k$. Clearly, all the kj largest jobs were assigned to $M_1, .., M_{k(j-1)}$. Thus $b_j = 0$ and similarly $b_i = 0$ for all $i \geq j$. Since by the definition of j $\sum_{i<j-1}(b_j - 1) < k$ and since $b_{j-1} \leq k$ then $\sum_{i<j}(b_j - 1) < 2k$. If j is not defined, then $\sum_{1 \leq i \leq k}(b_j - 1) < k < 2k$ as well. Thus $\sum_{1 \leq i \leq k}(b_j - 1) < 2k$ and $B < 3k$.

We now evaluate $E(V_{on}/V_{opt})$. Consider the case $V_{opt} = k^{2i^*+1}$. It is easy to see that the best for the on-line algorithm is to assign the jobs of the second phase, to the least loaded machines. Thus $ki^* - 1$ jobs are assigned to machines $M_{k(k-i^*)+2}, ..., M_m$, and the ratio is

$$\frac{ks_{k-i-1}}{l_{k(k-i^*)+1}} \leq \frac{ks_{k-i-1}}{(b_{k-i^*+1} + 1)s_{k-i-1}} = \frac{k}{b_{k-i^*+1} + 1} \ .$$

Thus

$$E\left(\frac{V_{on}}{V_{opt}}\right) \leq \frac{1}{k} \sum_{1 \leq i \leq k} \frac{b_{k-i+1} + 1}{k} \leq \frac{1}{k} \sum_{1 \leq i \leq k} \frac{b_i + 1}{k} \leq \frac{1}{k^2}(B + k) \leq \frac{4k}{k^2} = \frac{4}{\sqrt{m}}$$

which completes the proof of the theorem.

4 The related machines case

In this section we consider the case where the machines are related. Machine j has a speed v_j, and if job i is assigned to machine j, the load of the machine is increased by w_i/v_j. Most of the proofs in this section are omitted. First we show that it is impossible to design an algorithm whose competitive ratio is a function of the number of the machines.

Lemma 16. *There is no on-line algorithm for related machines whose competitive ratio is a function of the number of the machines (already when the number of machines is 2).*

We consider the case where the value of the optimal off-line assignment is known in advance. We show that the exact competitive ratio in this case is m.

Theorem 17. *The competitive ratio of any deterministic algorithm for related machines, even when the value of the optimal off-line assignment is known in advance, is at least m.*

Now, we show an m competitive algorithm for related machines case in which the value of the optimal off-line algorithm is known in advance. We use a constant $\alpha = \frac{1}{m}$. Assume without loss of generality that the value of the optimal off-line algorithm is 1.
Algorithm SLOW-FAST**:** Assign each new job on the fastest machine whose current load is less than α and its load with the job would be at least α. If no such machine exists, put the job on the slowest machine that has load less than α. If all machines have load of at least α, put the job on an arbitrary machine.

Theorem 18. *The algorithm Slow-Fast is m competitive for the related machines case in which the optimal value is known in advance.*

We call an algorithm "semi on-line" if jobs arrive in non increasing weight order.

Theorem 19. *Any semi on-line algorithm for related machines has competitive ratio of at least m.*

We show a matching m competitive algorithm.
Algorithm BIASED-GREEDY**:** Assign each new job to the machine with the current minimum load. In case of ties assign the job to the fastest machine among those with the minimum load.

Theorem 20. *The semi on-line algorithm Biased-Greedy is m competitive for related machines.*

If, in addition, the value of the optimal off-line algorithm is to be known in advance, it is possible to reduce the competitive ratio. We show a constant competitive semi on-line algorithm for related machines, for which the value of the optimal off-line algorithm is known in advance. We assume without loss of generality that this value is 1.

Algorithm Next-Cover**:** Assign a new job to the fastest machine, whose current load does not exceed $\frac{1}{2}$. If the load of all machines is at least $\frac{1}{2}$, assign it to an arbitrary machine.

Theorem 21. *The semi on-line algorithm Next-Cover is 2 competitive for the related machines case in which the optimal value is known in advance.*

Proof. From the definition of the algorithm it is clear that if the algorithm fails, the slowest machine has load less than $\frac{1}{2}$. We assume by contradiction that the slowest machine has load less than $1/2$. We sort the machines by non increasing speed, i.e. machine 1 is the fastest machine, and machine m is the slowest machine.

We show the following invariant. Let W_i be the total weight of the jobs assigned to machines 1 through i by the on-line algorithm and let W_i^* be the respective value for the optimal algorithm. We show that there exists an optimal assignment such that $W_i \leq W_i{}^*$ for all i. In particular $W_{m-1} \leq W_{m-1}{}^*$, and the slowest machine of the on-line is loaded by at least the same load of the optimal algorithm which is at least 1 which is a contradiction.

For $i = 0$ the invariant is trivially true. Assume it for $i - 1$ and prove it for i. If the on-line load on machine i does not exceed 1, the claim is trivially true. (Since in the optimal assignment, the load on this machine is at least 1). Thus, we may assume that the on-line load on the machine exceeds 1. We consider two cases according to the number of jobs on machine i.

Assume first that there are at least two jobs on the machine. The last job is larger than half the speed of the machine, since before it was placed, the load of the machine was less than $1/2$, and after it was placed, the load is larger than 1. Since the jobs arrive in non increasing order, then the first job on the machine is also larger than half the speed which contradicts the fact that the load was less than $1/2$.

Thus, we may assume that only one job placed on the machine i. Consider the set S of all jobs that arrived before the only job on machine i (including the job). Clearly the on-line assigned those jobs and only those jobs on machines 1 through i, thus $W_i = W(S)$. If all jobs in S are assigned by the optimal algorithm to machines 1 through i, then $W_i = W(S) \leq W_i{}^*$. Otherwise change the off-line packing in the following way: take one job of S that is assigned to a machine x, $x > i$, and put it on machine i; this gives load of at least 1 to machine i, since even the smallest job of S is at least of weight v_i. Put on x all jobs that were on i. Since x is a slower machine, its load is also at least 1. Hence $W_i \leq W_i{}^*$.

References

1. S. Albers. Better bounds for on-line scheduling. In *Proc. 29th ACM Symp. on Theory of Computing*, 1997. To appear.
2. N. Alon, J. Csirik, S. V. Sevastianov, A. P. A. Vestjens, and G. J. Woeginger. On-line and off-line approximation algorithms for vector covering problems. In *Proc. 4th European Symposium on Algorithms*, LNCS. Springer, 1996.

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. In *Proc. 25th ACM Symposium on the Theory of Computing*, pages 623–631, 1993.
4. S.F. Assmann. Problems in discrete applied mathematics. Technical report, Doctoral Dissertation, Mathematics Department, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1983.
5. S.F. Assmann, D.S. Johnson, D.J. Kleitman, and J.Y.-T. Leung. On a dual version of the one-dimensional bin packing problem. *J. Algorithms*, 5:502–525, 1984.
6. B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan, and J. Vitter. Load balancing in the l_p norm. In *Proc. 36th IEEE Symp. on Found. of Comp. Science*, pages 383–391, 1995.
7. Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.
8. Yair Bartal, Amos Fiat, Howard Karloff, and R. Vorha. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symp. on Theory of Computing*, 1992.
9. P. Berman and M. Karpinski. A note on on-line load balancing for related machines. Unpublished notes.
10. A. Borodin and R. El-Yaniv. On randomization in online computations. In *Computational Complexity*, 1997.
11. B. Chen, A. van Vliet, and G. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16:221–230, 1994.
12. J. Csirik, H. Kellerer, and G. Woeginger. The exact lpt-bound for maximizing the minimum completion time. *Operations Research Letters*, 11:281–287, 1992.
13. J. Csirik and V. Totik. On-line algorithms for a dual version of bin packing. *Discr. Appl. Math.*, 21:163–167, 1988.
14. B. Deuermeyer, D. Friesen, and M. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIAM J. Discrete Methods*, 3:190–196, 1982.
15. D. Friesen and B. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Math. Oper. Res.*, 6:74–87, 1981.
16. G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *Siam Journal on Computing*, 22(2):349–355, 1993.
17. M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, San Francisco, 1979.
18. R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
19. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:263–269, 1969.
20. D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994.
21. R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.
22. G. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. Technical Report, 1995.