Temporary Tasks Assignment Resolved [*]

Amitai Armon [†] Yossi Azar [‡] Leah Epstein [§] Oded Regev [¶]

Abstract

Among all basic on-line load balancing problems, the only unresolved problem was load balancing of temporary tasks on unrelated machines. This open problem exists for almost a decade, see [Borodin El-Yaniv]. We resolve this problem by providing an inapproximability result. In addition, a newer open question is to identify the dependency of the competitive ratio on the durations of jobs in the case where durations are known. We resolve this problem by characterizing this dependency. Finally, we provide a $PTAS$ for the off-line problem with a fixed number of machines and show a 2 inapproximability for the general case.

1 Introduction

On-line load balancing was extensively studied in the last decade (e.g., [1, 2, 3, 5, 10, 11, 13, 16, 18, 20, 23]). The basic problem contains the identical, related, restricted and unrelated models for permanent and temporary tasks. Tight bounds were given to all these problems except one: the assignment of temporary tasks to unrelated machines remained open. We present an inapproximability result by employing a cyclic load transfer method. Another more recent open question posed in [12] is whether the case where job durations are known at their arrival is provably harder than that of permanent jobs. We answer this question in the affirmative. We first summarize the results presented in this paper. Let m denote the number of machines and T the duration of the longest job.

- For on-line unrelated assignment of temporary tasks with unknown durations, we show an $\Omega(m/\log m)$ lower bound which almost matches a trivial $O(m)$ upper bound. For randomized algorithms we show an $\Omega(m/(\log m)^2)$ lower bound.
- For on-line restricted assignment of temporary tasks with known durations, we present a lower bound of $\Omega(\sqrt{m})$ and of $\Omega(\sqrt{\frac{\log T}{\log\log T}})$ on the competitive ratio of any on-line algorithm (deterministic or randomized). These lower bounds also hold for assignment on unrelated machines.

[*]A preliminary version of this paper appears in the proceedings of SODA'02.

[†]Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: armon@tau.ac.il.

[‡]Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: azar@tau.ac.il. Research supported in part by the Israel Science Foundation and by a grant of the European Commission.

[§]School of Computer and Media Sciences, The Interdisciplinary Center, Herzliya, Israel. E-Mail: lea@idc.ac.il.

[¶]Institute for Advanced Study, Princeton, NJ. E-Mail: odedr@ias.edu. Research supported in part by NSF grant CCR-9987845

- For offline assignment of temporary tasks on unrelated machines, we present a PTAS for the case where the number of machines is fixed. For the case where the number of machines is not fixed, we present a lower bound of 2 (provided that $P \neq NP$).

We also provide an additional result for interesting special cases of temporary tasks assignment in the unrelated machines model with unknown durations. Specifically, we show tight results for certain cases of the related-restricted model.

Definitions and previous results: We consider the problem of non-preemptive load balancing of temporary tasks on m unrelated machines. Each job (task) has an arrival time and a departure time, and should be assigned to one machine immediately upon its arrival. Each job j is associated with a loads vector $(w_j(1), ..., w_j(m))$. If job j is assigned to machine i, it increases the load of machine i by its *weight* on that machine, $w_j(i)$, for the duration of the job. The load on a machine at a certain time is the sum of the loads caused by the jobs assigned to it at that time. The goal is to minimize the maximum load over machines and time. Note that the load and the time are two separate axes of the problem. In the *known durations* setting we assume that when a job arrives its duration is given to the on-line algorithm and in the *unknown durations* setting we assume that the departure time is known only when the job actually departs.

An important special case of the unrelated machines model is the special case called the *restricted assignment model.* In this case, a job j can only be assigned to a subset of the machines that depends on the job. On each of these machines it causes the same load w_j. This is equivalent to a loads vector that contains only the values w_j or ∞. The results in [4] (and later in [21]) show a lower bound of $\Omega(\sqrt{m})$ on the competitive ratio that any on-line algorithm (deterministic or randomized) may have for restricted assignment of temporary tasks with unknown durations. An on-line algorithm for this problem with a competitive ratio of $O(\sqrt{m})$, was later presented in [6] (the "Robin-Hood" algorithm) thereby proving that the lower bound of $\Omega(\sqrt{m})$ is tight.

Unknown durations: Apart from a trivial upper bound, no on-line algorithm for the unrelated assignment model exists. The $\Omega(\sqrt{m})$ lower bound for restricted assignment mentioned above ([4]) dates back to 1992 but was still the best lower bound for unrelated assignment. Our results show that a trivial $O(m)$ competitive algorithm is almost optimal hence proving the inapproximability of this model. Specifically, by using a cyclic load transfer method we achieve an $\Omega(m/\log m)$ lower bound. The open problem mentioned in [12] regarding the existence of a better approximation algorithm is thus answered negatively. We extend the inapproximability result to randomized algorithms as well.

Known durations: The competitive ratio of $\Theta(\sqrt{m})$ for the restricted assignment model is usually regarded as a high competitive ratio, so in [6] it was suggested to consider this problem in the known-durations case, hoping to beat the $\Omega(\sqrt{m})$ bound, and get a polylogarithmic competitive ratio. This seemed plausible since the best known lower bound for the known durations case was only $\Omega(\log m)$ [7] (proven for the special case of permanent tasks). Indeed, the work of [6] made a step in this direction by showing an $O(\log mT)$ - competitive algorithm, where T is the duration of the longest job (in discrete time units). The competitive ratio of this algorithm is lower than $\Theta(\sqrt{m})$ for a large range of T. The obvious intriguing question, also presented in the survey of [3], was whether we can indeed improve on the $\Omega(\sqrt{m})$ lower bound and achieve a $O(polylog(m))$-competitive algorithm.

Surprisingly, we answer this question negatively, by extending the lower bound of [21] to a lower bound of $\Omega(\sqrt{m})$ that holds for the known durations case. This bound is tight by the upper bound for unknown durations. Our $\Omega(\sqrt{m})$ lower bound holds for unrelated machines as well (since restricted assignment is a special case). This result also answers an open question presented by Borodin and El-Yaniv in [12]. They asked whether there is any machine model in which one can prove a lower bound for temporary tasks assignment with known durations which is higher than the competitive ratio of permanent tasks assignment in that model. Our $\Omega(\sqrt{m})$ lower bound for restricted assignment of temporary tasks with known durations is strictly higher than the $O(\log m)$ competitive ratio for the case of permanent tasks. We note that as a function of T our lower bound is $\Omega(\sqrt{\frac{\log T}{\log\log T}})$ which can be compared to the $O(\log T)$ upper bound of [6] (assuming $T \geq m^{\epsilon}$).

Offline results: Next we consider the assignment of temporary tasks on unrelated machines in the offline setting. In the special case of *permanent* tasks, the tasks do not depart (all the departure times equal ∞). Horowitz and Sahni presented an FPTAS for permanent tasks assignment on a fixed number of unrelated machines (i.e., the number of machines is not a part of the input) [17]. A PTAS for this problem was also presented by Lenstra et al. [19]. For permanent tasks assignment on an arbitrary number of unrelated machines, Lenstra et al. [19] and Shmoys and Tardos [22] presented algorithms with an approximation ratio of 2. In addition, Lenstra et al. proved that no algorithm can reach an approximation-ratio better than $\frac{3}{2}$ for the arbitrary number of machines case, unless $P = NP$ [19].

Unlike the permanent case, solving the problem of *temporary* tasks assignment on unrelated machines cannot be done by standard rounding techniques. The problem arises from the two separate axes of the problem and this extra dimension is known to turn problems into intractable ones or very hard to approximate [15, 14, 9]. In order to obtain a PTAS after all, we had to use a two-dimensional rounding technique. While the above applies to the case of fixed number of machines, we also prove that when the number of machines is not fixed (i.e. is part of the input), no algorithm can achieve an approximation ratio lower than 2 unless $P = NP$. This lower bound is higher than the $\frac{3}{2}$ lower bound for permanent tasks and almost separates the two settings.

The problem of temporary tasks assignment on identical machines is another special case of our problem which was considered in [8] and provides the foundation for our algorithm. In this special case, the load that a job causes depends only on the job and it is identical for all the machines (i.e. $w_j(i) = w_j$ for $1 \leq i \leq m$). A PTAS for this problem in the case where the number of machines is fixed was presented in [8]. They also proved a lower bound of $\frac{3}{2}$ for the case of an arbitrary number of machines, provided that $P \neq NP$. Again, the lower bound that we present for our problem is higher than the lower bound for this special case.

2 Tasks of Unknown Duration

2.1 Inapproximability of the Unrelated Machines Model

In this section we present the inapproximability result for online load balancing of temporary tasks on unrelated machines. Namely, we show a lower bound of $\Omega(m/\log m)$ almost matching an upper bound of $O(m)$. It can be seen that the simple algorithm assigning each job to its fastest machine is an $O(m)$ competitive algorithm.

We proceed with the inapproximability result:

Theorem 2.1 *Any online algorithm for the load balancing of temporary tasks on unrelated machines is $\Omega(m/\log m)$ competitive.*

Proof: Let k be the largest integer power of 2 such that $k \le m/\log m$. Assume by contradiction that there is an online algorithm whose competitive ratio is below $k/2$. We describe a sequence of jobs given by an adversary such that there exists an optimal assignment whose maximum load is at most 1. The sequence ends as soon as there is a machine whose load in the online assignment is at least $k/2$.

The lower bound uses $l = \log k$ sets of k machines each. The sets are denoted by $M_1, M_2, ..., M_l$. In addition, a machine denoted by m_0^1 is used. Also, denote by m_i^j the j'th machine in the set M_i, $1 \le i \le l$, $1 \le j \le k$. Note that the total number of machines used, $l \cdot k + 1$, does not exceed m (when $m > 2$).

The adversary proceeds in phases. Before the start of phase t, we define a set of $l+1$ machines which we call active. One machine in each M_i is active and we denote its index by $a_i(t)$. The machine m_0^1 is always active and we use the notation $a_0(t) = 1$. The load of $m_i^{a_i(t)}$, $i = 0, ..., l$, in the online assignment is denoted by $b_i(t)$. We begin with setting $a_i(0) = 1$ for $i = 1, ..., l$.

A phase is composed of an arrival of one job and the departure of a set of jobs. The job presented by the adversary has infinite weight on non-active machines. Its weight on $m_i^{a_i(t)}$ is $2^i/k$ for $i = 0, ..., l$. Assume the online algorithm assigns it to machine $m_i^{a_i(t)}$. In case the new load, $b_i(t+1)$, is $k/2$ or more the sequence stops. Otherwise, the phase is completed with the departure of the jobs assigned by the online algorithm to $m_{i+1}^{a_{i+1}(t)}, ..., m_l^{a_l(t)}$ (no job leaves when $i = l$). The set of active machines for the next phase is set as follows: $a_j(t+1) = 1$ for any $i + 2 \le j \le l$ and unless $i = l$ we also set $a_{i+1}(t+1) = 1 + \lfloor 2b_i(t) \rfloor$. All other active machines stay the same. Note that since $b_i(t) < k/2$ the above definition of $a_i(t+1)$ is valid, that is, $a_i(t+1) \le k$. Also note that by the above construction, non-active machines are always empty.

If we consider the vector of loads of the online assignment, $(b_0(t), b_1(t), ..., b_l(t))$, we note that the vector increases lexicographically after each phase. That is, at least one of the coordinates increases while all previous coordinates do not change. The increase is by at least $1/k$. Since the adversary sequence is completed once one of the coordinates exceeds $k/2$, the sequence is completed after a finite number of phases, or specifically, at most $O(k^{2l}) = O(m^{2\log m})$ phases.

In what follows we complete the proof by showing an optimal assignment where the maximum load does not exceed 1 during the whole sequence. In case the job arriving at phase t is assigned by the online algorithm to $m_i^{a_i(t)}$, $i = 0, ..., l-1$, then the optimal algorithm assigns it to machine $m_{i+1}^{a_{i+1}(t)}$. Otherwise, the job arriving at phase t is assigned by the online algorithm to $m_l^{a_l(t)}$ and the optimal algorithm assigns it to machine m_0^1.

First, jobs assigned by the optimal assignment to m_0^1 are assigned by the online algorithm to an active machine in M_l. Since that machine's load is not more than $k/2$ and all other machines in M_l are empty, the incurred load on m_0^1 is at most $1/2$. Now consider jobs assigned to m_{i+1}^j, $i = 0, ..., l-1$, by the optimal assignment. These are assigned by the online algorithm to the active machine in M_i. Moreover, when they were assigned, the load on the active machine in M_i was at least $(j-1)/2$ and less than $j/2$ because a_{i+1}, which was equal to j, is defined as twice this load plus one rounded down. Therefore, their total load on a machine in M_i is at most $1/2$ and their total load on a machine in M_{i+1} is at most 1.

Concluding, the above sequence was shown to have an optimal assignment of maximum load 1. Moreover, as long as the online maximum load is below $k/2$ the online load vector was shown to increase lexicographically. This contradicts our assumption that an online algorithm with competitive ratio below $k/2$ exists and completes the proof. ■

The following lemmas demonstrate a general technique for converting deterministic lower bounds into randomized ones.

Lemma 2.2 *Let R be a randomized on-line algorithm for load balancing of temporary tasks on unrelated machines which achieves a competitive ratio of c. Then there exists a deterministic on-line algorithm D, allowed to split jobs between different machines, which achieves a competitive ratio $\leq c$ against an optimal algorithm which is not allowed to split jobs.*

Proof: Algorithm R determines the probability $p_j(i)$ that a job j is assigned to machine m_i, $\sum_i p_j(i) = 1$. We denote the expectation of the maximum load of R by l^R. For any input sequence, $l^R \leq c \cdot l^{OPT}$.

We define D as follows. For an arriving job j, D splits the job between the machines according to the probabilities set by R. So D assigns a load of $p_j(i) \cdot w_j$ to each machine m_i. We denote the maximum load of D by l^D.

We now prove that for any input sequence $l^D \leq l^R$. Note that at any given time t, the load of D on machine m_i is $l_i^D(t) = E(l_i^R(t))$. Hence, $l^D(t) = max_i(l_i^D(t)) = max_i(E(l_i^R(t)))$. Using the fact that the maximum of expectations is at most the expectation of maxima, $l^D(t) \leq E(max_i(l_i^R(t))) = l^R(t)$. The proof is completed by noting that $l^D = max_t(l^D(t)) \leq max_t(l^R(t)) = l^R \leq c \cdot l^{OPT}$. ■

Lemma 2.3 *Let c be a lower bound on the competitive ratio of any deterministic algorithm for unrelated assignment of temporary tasks. If c can be proven with an adversarial strategy of jobs in which each job has an admissible set of at most k machines, then there is a lower bound of $\frac{c}{k}$ on the competitive ratio of any randomized on-line algorithm for the same problem.*

Proof: We will prove that there is a lower bound of $\frac{c}{k}$ on the competitive ratio of any deterministic on-line algorithm D which is allowed to split jobs (compared to an optimum which is not allowed to split them). According to the previous lemma, this implies a lower bound of $\frac{c}{k}$ for randomized on-line algorithms as well. Consider the sequence used for the deterministic lower bound. Since each job in that sequence can only be assigned to at most k machines, D must assign at least $\frac{1}{k}$ of the job's weight to one machine. Now, consider a deterministic online algorithm D' which simulates D and assigns each job to just one machine: the machine to which D assigned the largest part of the job. This is an online algorithm that does not split jobs and therefore our adversarial strategy creates a load of at least $l^{D'} \geq c \cdot l^{OPT}$. Since D assigns at least a $\frac{1}{k}$-fraction of each job to the same machine as D', its load at the end of the same sequence must be at least $l^D \geq \frac{1}{k} \cdot c \cdot l^{OPT}$ which gives the required competitive ratio. ■

Theorem 2.4 *Any online randomized algorithm for the load balancing of temporary tasks on unrelated machines is* $\Omega(m/(\log m)^2)$ *competitive.*

Proof: The construction in Theorem 2.1 uses admissible sets of at most $\log m$ machines. The theorem then follows as a corollary of Lemma 2.3. ■

2.2 Tight Results for the Related-Restricted Machines Model

The result in the previous section shows that approximating the unrelated machines model is almost infeasible. As an alternative to the unrelated machines model we consider the so called related-restricted machines model. Here, each machine has its own speed and each job has a weight and a set of admissible machines. However, note that the lower bound presented in the last section still applies here so approximating is still infeasible. We show that by limiting the number of different machine speeds to a constant number, we can approximate the problem better. Specifically, in the case where only two different machine speeds are involved we obtain the following lower bound. Later, we will present a matching upper bound.

Theorem 2.5 *Any online algorithm for the load balancing of temporary tasks in the related-restricted machines model with speeds* $\{1, s\}$ *is* $\Omega(\min\{\max\{m/s, \sqrt{m}\}, \sqrt{ms}\})$ *competitive. The same holds for randomized algorithms as well.*

Proof: A lower bound of $\Omega(\sqrt{m})$ already exists for the special case of the restricted assignment model and therefore we can limit ourselves to showing a lower bound of $\Omega(\min\{m/s, \sqrt{ms}\})$ for $s \leq \sqrt{m}$. Let $m' = m/3$ and $b = \min\{m'/s, \sqrt{m's}\}$. Assume by contradiction that there is an online algorithm whose competitive ratio is below b. We describe a sequence of jobs given by an adversary such that there exists an optimal assignment whose maximum load is at most 1. The sequence ends as soon as there is a machine whose load in the online assignment is at least b.

The lower bound uses three sets of machines, M_0, M_1 and M_2. Both M_1 and M_2 contain m' machines whereas M_0 contains m'/b machines. The j'th machine in M_i is denoted m_i^j. The idea behind the lower bound is to force the online algorithm to assign jobs to machines in M_0 while an optimal algorithm can assign them to machines in M_1.

The lower bound proceeds in phases and stops as soon as the load on a machine reaches b. At the beginning of a phase t we choose three active machines, one in each M_i. Their indices are denoted by $a_0(t)$, $a_1(t)$ and $a_2(t)$. They are initially set to $a_0(0) = a_1(0) = a_2(0) = 1$. One job is presented in each phase. Its weight on $m_0^{a_0(t)}$ or on $m_1^{a_1(t)}$ is $1/s$ where the weight on $m_2^{a_2(t)}$ is 1. In case the online algorithm assigned the job to $m_0^{a_0(t)}$, all jobs assigned by the online algorithm to M_1 or M_2 leave. If the job is assigned to $m_1^{a_1(t)}$ all jobs assigned to M_2 leave. Otherwise, the job was assigned to $m_2^{a_2(t)}$ and no jobs leave. For the next phase, we set $a_0(t+1) = 1+\lfloor b_2(t)/s \rfloor$, $a_1(t+1) = 1+\lfloor b_0(t) \rfloor$ and $a_2(t+1) = 1+\lfloor sb_1(t) \rfloor$ where $b_i(t)$ denotes the total on-line load on machines in M_i. This is possible first because in M_0 there are m'/b machines each of which has a load of less than b so that $a_1(t+1) = 1+\lfloor b_0(t) \rfloor \leq b \cdot m'/b = m'$. In addition, note that at any given time only one machine in M_1 and one machine in M_2

are not empty in the online assignment. Therefore, $a_2(t+1) = 1 + \lfloor sb_1(t) \rfloor \leq s \cdot b \leq m'$ and $a_0(t+1) = 1 + \lfloor b_2(t)/s \rfloor \leq b/s \leq m'/b$.

Note that the vector $(b_0(t), b_1(t), b_2(t))$ lexicographically increases after each phase by at least $1/s$. Moreover, the adversary described above proceeds as long as the maximum load is below b. Therefore, after a finite number of phases the load on one of the machines is going to reach b.

The proof is completed by showing an optimal assignment where the maximum load does not exceed 1. In case the job arriving at phase t is assigned by the online algorithm to machine $m_i^{a_i(t)}$, then the optimal algorithm assigns it to machine $m_j^{a_j(t)}$ where $j = i+1 mod 3$.

Now consider jobs assigned to m_1^j by the optimal assignment. These are assigned by the online algorithm to a machine in M_0. Moreover, when they were assigned, the total online load on machines in M_0 was at least $j-1$ and less than j. Since jobs assigned by the online algorithm to M_0 never leave, the load on m_1^j is at most 1. Similarly, consider jobs assigned to m_2^j by the optimal assignment. These are assigned by the online algorithm to an active machine in M_1. When they were assigned, the total online load on the active machine in M_1 was at least $(j-1)/s$ and less than j/s. Since jobs assigned by the online algorithm to M_1 leave all together and never leave individually, the load on m_2^j is at most $s \cdot 1/s = 1$. A similar argument holds for jobs assigned to M_0 by the optimal assignment.

Concluding, the above adversarial sequence was shown to have an optimal assignment of maximum load 1 and as long as the online maximum load is below b, an online load vector was shown to increase lexicographically. This contradicts our assumption that a b-competitive online algorithm exists and completes the proof. By Lemma 2.3, the result also holds for the randomized case since there are exactly three machines in any admissible set. ■

In the rest of this section, we describe two online algorithms whose combination achieves a competitive ratio matching the lower bound stated above. In both algorithms we assume that $OPT = 1$. Nevertheless, by using standard doubling techniques we can overcome that assumption while increasing the competitive ratio by a factor of at most 4. Both algorithms are based on a modified Robin Hood algorithm [6] with some threshold b. A machine is said to be *poor* at a certain time if its load is less than b and is said to be *rich* otherwise. For a machine rich at time t we define its *windfall time* as t_0 if it is rich from time t_0 to t but is poor at time $t_0 - 1$. As its name implies, the Robin Hood algorithm tries to assign jobs to the poor machines or in case none exists, to the a rich machine with the most recent windfall time.

Algorithm 1 *In case an incoming job has at least one admissible fast machine, remove all slow machines from its admissible machines. Then use the Robin Hood algorithm with a threshold* $b = \max\{\sqrt{m}, m/s\}$.

Claim 2.6 *The above algorithm is* $O(\max\{\sqrt{m}, m/s\})$ *competitive.*

Proof: First notice that jobs assigned to slow machines are not allowed to be assigned to any fast machine. Therefore, the contribution of a job to the total load of an optimal assignment is at least its contribution to the total load of the online assignment. Hence, the total load on all machines at any given time is at most that of an optimal assignment which is at most m. This implies that the number of rich machines at any given time is at most m/b.

By the definition of the algorithm, any job assigned to a slow rich machine i has only slow rich machines in its admissible set. Moreover, the windfall time of these machines is before the windfall time of i, that is, they are all already rich at the windfall time of i. Therefore, in an optimal assignment, all jobs assigned to i after its windfall time are assigned to at most m/b slow machines. Their total load on i is at most m/b. Adding the job made i rich, we get that the total load on i at any given time is at most $b + 1 + m/b = O(b)$.

Now consider a fast rich machine. Unlike slow machines, some of the jobs assigned to fast machines might result from the algorithm's removal of slow admissible machines. Thus jobs assigned to a fast rich machine i after its windfall time are assigned in an optimal assignment to one of at most m/b fast machines or to any one of the slow machines. The total load on machine i is therefore at most $b + 1 + m/b + m/s = O(b)$ and the required competitive ratio is achieved. ■

Algorithm 2 *Use the Robin Hood algorithm with a threshold $b = \sqrt{ms}$.*

Claim 2.7 *The above algorithm is $O(\sqrt{ms})$ competitive.*

Proof: The total load on the fast machines at any given time is at most that of an optimal assignment which is at most m. The total load on the slow machines at any given time is at most sm since these jobs might be assigned to fast machines by an optimal assignment. This implies that the number of fast rich machines at any given time is at most m/b and that the number of slow rich machines is at most sm/b.

Consider a job assigned to a rich machine i. All other machines in its admissible set are rich and their windfall time is before the windfall time of i, that is, they are all already rich at the windfall time of i. Therefore, all jobs assigned to i after its windfall time are assigned in an optimal assignment to at most sm/b slow machines and to at most m/b fast machines. Therefore, if i is slow, their total load on i is at most $2sm/b$ and if i is fast, their total load on i is at most $2m/b$. Adding the job that made i rich, we get that the total load on i at any given time is at most $b + s + 2sm/b = O(\sqrt{ms})$. ■

Algorithm 3 *If $s > m^{1/3}$ use the first algorithm; otherwise, use the second algorithm.*

Theorem 2.8 *The above algorithm is $O(\min\{\max\{m/s, \sqrt{m}\}, \sqrt{ms}\})$-competitive for load balancing of temporary tasks in the related-restricted machines model with speeds $\{1, s\}$.*

3 Tasks of Known Duration

Theorem 3.1 *Any deterministic on-line algorithm for load-balancing of temporary tasks with known durations in the restricted assignment model has a competitive ratio of at least $\sqrt{m}$.*

Proof: Let ON be an on-line algorithm for the problem, and let OFF be an optimal offline algorithm for solving it. We will show a sequence of jobs for which ON reaches a load of at least $\sqrt{m}$, whereas OFF maintains a load of one. First we describe the sequence, and then we prove the lower bound.

We denote the set of the first $\sqrt{m}$ machines by A, and the set of the remaining machines by B. The i'th machine in A (respectively B) will be denoted by A_i (respectively, B_i), for $1 \le i \le \sqrt{m}$ (respectively, for $1 \le i \le m - \sqrt{m}$).

We force ON to assign $\sqrt{m}$ jobs to a single machine in B, or to assign m jobs to A (which consists of only $\sqrt{m}$ machines).

Our input sequence only includes unit jobs and consists of at most $m - \sqrt{m}$ phases. Each phase p ($p \geq 1$) consists of at most $\sqrt{m}$ jobs. The j'th job in phase p ($j \geq 1$) is admissible to two machines: A_j and B_p. The exact arrival and departure time of each job will be described later. The number of jobs arriving in each phase is determined by the behavior of ON. Jobs keep arriving in phase p as long as ON assigns them to B_p (up to the maximum of $\sqrt{m}$ jobs). When ON assigns a job to a machine in A, the phase ends (i.e. no more jobs arrive in this phase). Let N_p be the number of jobs which arrived in phase p. By definition, $1 \leq N_p \leq \sqrt{m}$. The number of phases is also determined by the behavior of ON. If $N_p = \sqrt{m}$ for a certain phase p (i.e., ON assigns all the jobs of that phase to B_p), then the sequence stops. If phase $m - \sqrt{m}$ has less than $\sqrt{m}$ jobs, then we bring one more unit job ("extra job"), which is restricted to the most loaded machine that ON has in A.

We now describe the arrival and departure times of the jobs in each phase. We first describe these times for the first phase, and then we inductively define them for the other phases. The length of the time interval that our sequence will use is $T = \sqrt{m}^{(m-\sqrt{m}+1)}$. Let $S_1 = 0$ and let $T_1 = T$. The first phase starts at time $S_1 = 0$. The jth job of phase 1 arrives at time $j - 1$ ($1 \leq j \leq N_1$). Its departure time is $\frac{j \cdot T_1}{\sqrt{m}}$.

For each phase $p > 1$, we inductively define the arrival and departure times of the jobs to be between the departure times of the last two jobs of the previous phase. For $p \geq 1$, we define T_{p+1} as the departure time of the last (i.e. N_p'th) job of phase p. We also define S_{p+1} as the departure time of the $N_p - 1$'st job of phase p. If $N_p = 1$ then S_{p+1} is equal to S_p. Each phase p starts at time S_p, and only uses the time interval $[S_p, T_p]$. The arrival time of the j'th job in phase p is $S_p + j - 1$, and its departure time is $S_p + \frac{j \cdot (T_p - S_p)}{\sqrt{m}}$. Recall that in case $N_{m-\sqrt{m}} < \sqrt{m}$ we add one more unit job, restricted to the most loaded machine that ON has in A. This "extra job" lasts just one time unit and arrives at time $S_{m-\sqrt{m}+1}$. This completes the description of our sequence.

We first prove that ON achieves a load of at least $\sqrt{m}$ for the above sequence. Note that the last arrival in phase p occurs before time $S_p + \sqrt{m}$ and that the first departure occurs at time $S_p + (T_p - S_p)/\sqrt{m}$. Also, the duration of phase p can be seen to be $T_p - S_p = \sqrt{m}^{m-\sqrt{m}+2-p}$. Hence, in every phase p, the first departure occurs after the last arrival. Therefore, when a job arrives all the previous jobs in the same phase are still active. This means that if there exists a phase p in which ON assigns all the jobs to machine B_p, then it reaches a load of at least $\sqrt{m}$ (which occurs at time $S_p + \sqrt{m} - 1$), and we are done.

Otherwise, ON must assign a job to A at a certain stage of each phase. By definition, the phase ends as soon as this happens. Recall that phase $p + 1$ starts when the $N_p - 1$'st job of phase p leaves, and ends before the departure of the N_p'th job of phase p. So the last job of phase p is active throughout phase $p + 1$ and no other job from phase p is active during phase $p + 1$. Since $[S_{p+1}, T_{p+1}] \subseteq [S_p, T_p]$ we conlude that the last job of each of the phases $1, ..., p$ is active throughout phase $p + 1$, and no other job from phases $1, ..., p$ is active during phase $p + 1$. Recall that the last job in each phase is assigned to A by ON. So at time S_p, the active jobs are exactly all the jobs that ON assigned to A in phases $1, \ldots, (p-1)$. At time $S_{m-\sqrt{m}+1}$, ON has $m - \sqrt{m}$ jobs in A. There are only $\sqrt{m}$ machines in this set, so the most loaded machine in A, machine A_i, has a load of at least $\sqrt{m} - 1$. As we explained before, the "extra job" now arrives, and can only be assigned to machine A_i. This makes the load of ON on that machine at least $\sqrt{m}$.

Now we describe a possible assignment of algorithm OFF. When the j'th job of phase p arrives, it can be assigned either to A_j or to B_p. If ON assigns the job to the machine in B, then OFF assigns it to the machine in A. If ON assigns it to the machine in A, then OFF assigns it to the machine in B. If the "extra job" arrives, then OFF assigns it to its admissible machine.

Let us consider now the load of OFF. At the beginning of phase p, OFF has no active jobs in A, since we saw that ON has no active jobs in B. OFF has one active job on each of the machines $B_1, ..., B_{p-1}$, since ON has one active job from each phase in A. During phase p, as long as ON assigns jobs to B_p, OFF assigns each of them to a different machine in A (which was empty at the beginning of the phase). When ON assigns a job to A, OFF assigns it to B_p (which is empty), and the phase ends (so no other job will be assigned to B_p). Therefore, OFF maintains a load of 1 throughout the phases. At time $S_{m-\sqrt{m}+1}$, OFF has one active job on each machine in B (one job from each phase), and no jobs in A. So it can assign the "extra job" to A_i without exceeding the maximum load of 1. Thus we have reached the required competitive ratio. ■

Let us denote the total length of the time interval used by the input sequence by T. The result above also applies when we limit the length T of the sequence and when we allow randomization to be used. The results are summarized in the next two theorems:

Theorem 3.2 *Any deterministic on-line algorithm for load-balancing of temporary tasks with known durations in the restricted assignment model has a competitive ratio of at least* $\Omega(\sqrt{\frac{\log T}{\log\log T}})$*, for any* $T < \sqrt{m}^{(m-\sqrt{m}+1)}$.

Note that this lower bound is at most $\sqrt{m}$ for this range of T.

Proof: For any $T = \sqrt{x}^{x-\sqrt{x}+1}$ where $x \le m$, we can clearly apply the exact steps of the previous proof, limiting ourselves to the first x machines instead of using all the machines. We will have a sequence with at most $x - \sqrt{x}$ phases, each of them having at most $\sqrt{x}$ jobs, and we will obtain a lower bound of $\sqrt{x}$. We can easily see that in this case: $\sqrt{x} = \Omega(\sqrt{\frac{\log T}{\log\log T}})$. This means that for any $T < \sqrt{m}^{(m-\sqrt{m}+1)}$), we have a lower bound of $\Omega(\sqrt{\frac{\log T}{\log\log T}})$, as required. ■

Theorem 3.3 *A randomized on-line algorithm for solving the problem of restricted assignment of temporary tasks with known durations cannot achieve a competitive ratio smaller than* $\frac{1}{2}\sqrt{m}$*. Moreover, for any* $T < \sqrt{m}^{(m-\sqrt{m}+1)}$ *no algorithm can be better than* $\Omega(\sqrt{\frac{\log T}{\log\log T}})$*-competitive.*

Proof: Note that in Theorems 3.1 and 3.2 admissible sets contain at most two machines. The results follow by using Lemma 2.3. ■

4 Off-line Temporary Assignment

4.1 Fixed Number of Machines - A *PTAS* for temporary assignment of unrelated tasks

4.1.1 Overview

We start with an overview of the polynomial-time approximation scheme and give the details later. We begin with scaling the weights of the jobs, in order to limit the possible range of the optimal maximum load. It is well-known that we can achieve an approximation ratio of m simply by assigning each job to its fastest machine. We will refer to this simple algorithm as "Fastest-Assign". We apply this algorithm to our input, and denote the maximum load reached by l. Now we multiply each of our jobs' weights by $\frac{m}{l}$. This assures us that the optimal maximum load is in the range $[1, m]$. Note that this scaling requires only linear time.

The algorithm then follows with five phases: the weight-rounding and grouping phase, the time- rounding phase, the combining phase, the solving phase and the converting phase. In the first phase, the weights of the jobs are rounded upwards, and then they are divided into a large number of subsets based on their rounded weights, as will be explained later. Next the time-rounding phase is applied to each of these subsets. This phase actually consists of two subphases. In the first subphase the jobs' active time is extended: some jobs will arrive earlier, others will depart later. In the second subphase, the active time is again extended but each job is extended in the opposite direction to which it was extended in the first subphase. The combining phase is also applied to each subset separately. In this phase the algorithm combines several jobs from the same subset into jobs with higher load vector coordinates. In the solving phase, we find an optimal solution for the modified problem (the solving is performed for all the jobs together). The solution we found can be converted into a solution for the original problem in the converting phase, which is again applied separately to each subset.

4.1.2 Description of the PTAS

We denote the sequence of events by $\sigma = \sigma_1, ..., \sigma_{2n}$, where each event is an arrival or a departure of a job; we assume that at each time only one job arrives or departs. Since all the events are known at the beginning, we view σ as a sequence of times, the time σ_i is the moment after the i'th event happened. In addition, σ_0 denotes the moment at the beginning, before the arrival of the first job. We assume without loss of generality that $m \geq 2$ (otherwise the approximation ratio is always 1).

Let $\epsilon' > 0$ be the precision required by the PTAS. We assume that $\epsilon' < 1$. We choose $\epsilon = \epsilon'/7$, and fix the following 3 constants:

$$\begin{aligned}
\alpha &= \frac{\epsilon^2}{m\lceil\log n\rceil \cdot (\lceil\log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)^m} \\
\beta &= \frac{\alpha\epsilon^2}{m^2} = \frac{\epsilon^4}{m^3\lceil\log n\rceil \cdot (\lceil\log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)^m}
\end{aligned}$$

$$\gamma = \frac{\beta\epsilon^2}{m^2} = \frac{\epsilon^6}{m^5\lceil\log n\rceil \cdot (\lceil\log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)^m}$$

Phase 1: The weight-rounding and grouping phase. We start by describing the weight-rounding and grouping phase. For each job j, we denote by W_j the load that j causes on its fastest machine: $W_j = min_i(\bar{w}_j(i))$. We will refer to W_j as the "min-weight" of the job j. We define the "relative speed" vector of job j, $\bar{v}_j$ by: $\bar{v}_j(i) = \bar{w}_j(i)/W_j$, for $1 \le i \le m$. Note that $\bar{v}_j(i) \ge 1$. We now perform a rounding of the vector $\bar{v}_j$ and obtain the rounded "relative speed" vector $\bar{v}'_j$. For each $1 \le i \le m$, if $\bar{v}_j(i) \ge m/\epsilon$, then $\bar{v}'_j(i) = \infty$, and we will refer to machine i as an "illegal machine" for job j. Otherwise, we obtain $\bar{v}'_j(i)$ by rounding $\bar{v}_j(i)$ upwards to the nearest power of $1+\epsilon$, and we will refer to machine i as a "legal machine" for job j. Note that each coordinate of the vector $\bar{v}'_j$ may have $\lceil\log_{(1+\epsilon)}(\frac{m}{\epsilon})\rceil + 1$ possible values, since its value is either ∞ or a power of $(1+\epsilon)$ between 1 and m/ϵ. Now we define a new loads vector for the job j, $\bar{w}'_j$, by: $\bar{w}'_j(i) = W_j \cdot \bar{v}'_j(i)$, for $1 \le i \le m$. In this we completed the rounding of the weights.

Next we divide the jobs into subsets according to their $\bar{v}'$ vector. This division splits the jobs into at most $(\lceil\log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)^m$ subsets, since each of the m coordinates of that vector may have $(\lceil\log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)$ possible values, as we noted before. This completes the description of the first phase.

Phase 2: The time-rounding phase. This phase is similar to the time-rounding phase described by [8]. We will apply this phase separately to each subset of jobs $J_{\bar{v}'}$ (having the same "relative speeds" vector $\bar{v}'$).

In order to describe the time-rounding phase with its two subphases, we start with defining partitions of each subset $J_{\bar{v}'}$, based on which the rounding will be performed. The set $R_{\bar{v}'}$ contains all jobs with $W_j \ge \gamma$ out of the jobs in $J_{\bar{v}'}$.

From now on, we fix $\bar{v}'$ in the description of this phase, and refer to $J_{\bar{v}'}$ as J and to $R_{\bar{v}'}$ as R. All the following definitions are made for these fixed J and R.

We begin by defining a partition $\{J_i\}$ of the set of jobs $J - R$. We set $M_1 = J - R$ and define sets J_i and M_i iteratively as follows. Let M_i be a set of jobs and consider the sequence of times in σ in which jobs of M_i arrive and depart. The number of such times is $2r$ for some r, let c_i be any time between the r'th and the $r+1$-st elements in that set. The set J_i contains the jobs in M_i that are active at time c_i. The set M_{2i} contains the jobs in M_i that depart before or at c_i and the set M_{2i+1} contains the jobs in M_i that arrive after c_i. We stop when all unprocessed M_i's are empty. The important property of that partition is that the set of jobs from $J - R$ that are active at a certain time is partitioned into at most $\lceil\log n\rceil$ different sets J_i.

We continue by further partitioning the set J_i. We order the jobs according to their arrival time. We denote the smallest prefix of the jobs whose total min-weight is at least α by S_i^1. We order the same jobs according to their departure time. We take the smallest suffix whose min-weight is at least α and denote that set by T_i^1. Note that there might be jobs that are both in S_i^1 and T_i^1. We remove the jobs in $S_i^1 \cup T_i^1$ from J_i, repeat the process with the jobs left in J_i and similarly define $S_i^2, T_i^2, \ldots, S_i^{k_i}, T_i^{k_i}$. Each set S_i and T_i has total min-weight between α and $\alpha + \gamma$, except for the last pair which may have smaller min-weight than α. However, if the last pair has smaller min-weight than α, then it satisfies $S_i^{k_i} = T_i^{k_i}$. We denote by s_i^j the arrival time of the first job in S_i^j and by t_i^j the departure time of the last job in T_i^j. Note that $s_i^1 \le s_i^2 \le ... \le s_i^{k_i} \le c_i \le t_i^{k_i} \le ... \le t_i^2 \le t_i^1$.

The first subphase of the time-rounding phase creates a new set of jobs J' which contains the same jobs as in J with slightly longer active times. We change the arrival time of all the jobs in S_i^j for $j = 1, ..., k_i$ to s_i^j. Also, we change the departure time of all the jobs in T_i^j to t_i^j. The jobs in R are left unchanged. We denote the sets resulting from the first subphase by J', J'_i, S'^j_i, T'^j_i.

The second subphase of the time-rounding phase further extends the active time of the jobs resulting from the first subphase. We take one of the sets J'_i and the partition we defined earlier to $S'^1_i \cup T'^1_i$, $S'^2_i \cup T'^2_i$, $\ldots$, $S'^{k_i}_i \cup T'^{k_i}_i$. For every $j \leq k_i$, we order the jobs in S'^j_i according to an increasing order of departure times. We take the smallest prefix of this ordering whose total min-weight is at least β. We extend the departure time of all the jobs in that prefix to the departure time of the last job in that prefix. The process is repeated until there are no more jobs in S'^j_i. The last prefix may have a min-weight of less than β. Similarly, we extend the arrival times of jobs in T'^j_i. Note that if the total min-weight of either $S'^{k_i}_i$ or $T'^{k_i}_i$ is smaller than α then $S'^{k_i}_i = T'^{k_i}_i$ and these jobs are left unchanged since they already have identical arrival and departure times from the first phase. We denote the sets resulting from the second subphase by J'', J''_i, S''^j_i, T''^j_i.

Phase 3: The combining phase. This phase involves the load vectors of the jobs. It is also applied to each subset $J''_{\bar{v}'}$ separately, so we again fix $\bar{v}'$ in the description of this phase and refer to $J''_{\bar{v}'}$ as J''. Let J''_{st} be the set of jobs in J'' that arrive at s and depart at t. Assume the total min-weight of jobs in J''_{st} is x. The combining phase replaces these jobs by $\lceil x/\gamma \rceil$ jobs, which have a load-vector of $\gamma \cdot \bar{v}'$. Note that the maximum finite weight in their loads vector may be $\frac{m}{\epsilon} \cdot \gamma$. We denote the resulting sets by J'''_{st}. The set J''' is created by replacing every J''_{st} with its corresponding J'''_{st}, that is, $J''' = \bigcup_{s,t} J'''_{st}$.

Phase 4: The solving phase. This phase solves the modified decision problem, i.e. it solves the problem after each subset $J_{\bar{v}'}$ has been replaced by a modified subset $J'''_{\bar{v}'}$. The solving phase is performed once for all the jobs together (not for each $J'''_{\bar{v}'}$ subset separately). We solve the modified decision problem by building a layered graph. Every time σ_i, $i = 0, \ldots, 2n$, in which jobs arrive or depart (including the initial state with no job) has its own set of vertices called a layer. Each layer holds a vertex for every possible assignment of the current active jobs to machines (except assignments of weight ∞); furthermore, we label each node by the maximum load of a machine in that configuration.

Two vertices of adjacent layers σ_{i-1} and σ_i, $i = 1, \ldots, 2n$, are connected by an edge if the transition from one assignment of the active jobs to the other is consistent with the arrival and departure of jobs at time σ_i. More precisely, the vertices are connected if and only if every job active both before and after σ_i is assigned to the same machine in the assignments of both vertices. At each event, jobs either arrive or depart but not both (due to the assumption at the beginning that all the original events are distinct; during rounding we do not mix arrival and departure events). If σ_i is an arrival, the indegree of all vertices in the layer σ_i is 1, since the new configuration determines the old one. Similarly if σ_i is a departure, the outdegree of all vertices in the layer σ_{i-1} is 1. In both cases, the number of edges between two layers is linear in the number of vertices on these layers. It follows that the total number of edges is linear in the number of vertices.

We define a value of a path as the maximal value of its nodes. Now we can simply find a path with smallest value from the first layer to the last one by any shortest path algorithm in linear time (since the graph is layered).

Phase 5: The converting phase. In this phase the algorithm converts the assignment found for the modified problem into an assignment for the original problem. This phase is performed separately for the jobs in each of the subsets $J'''_{\bar{v}'}$. Each assignment of the jobs of a modified subset $J'''_{\bar{v}'}$ is converted into an assignment for the jobs of subset $J_{\bar{v}'}$, which is also an assignment for the original problem. Again we fix $\bar{v}'$ throughout the description of this phase, and refer to $J'''_{\bar{v}'}$ as J'''. Assume the number of jobs having $W_j = \gamma$ in J'''_{st} that are assigned to a certain machine i is r_i. Remove these jobs and assign all the jobs having $W_j \leq \gamma$ in J''_{st} to the machines such that a total weight of at most $(r_i+1)\gamma \cdot \bar{v}'(i)$ is assigned to machine i.

Note that all the jobs will be assigned that way. The replacement involves jobs whose min-weight is at most γ. We know that the total min-weight of these jobs is at most $\gamma \cdot \sum_{i=1}^{m} r_i$.

If they made a load of $(r_i+1)\bar{v}'(i)\gamma$ on each of the machines, then it would mean that their total min-weight was at least $\gamma \cdot (m + \sum_{i=1}^{m} r_i)$. So it is possible to assign all these jobs so that they will make a load of at most $(r_i+1)\gamma \cdot \bar{v}'(i)$ on each machine i.

The assignment for J'' is also an assignment for J' and J. An assignment for J is also an assignment for the original problem.

4.1.3 Analysis of PTAS

We now perform an analysis of our algorithm. Our steps of proof are similar to those used by [8], but our lemmas and proofs will have some adjustments. We will denote the problem for the original input by I, the problem after the first phase by I^+, the problem after the first time-rounding phase by I', the problem after the second time-rounding phase by I'', and the problem after the combining phase by I'''.

Lemma 4.1 *Given a solution for the problem I whose maximum load is λ, we can convert it to a solution for I^+ by moving jobs that were assigned to one of their "illegal machines" to their fastest machine. This solution for I^+ will have a maximum load of at most $\lambda(1+\epsilon)^2$. Also, given a solution for I^+ whose maximum load is λ, the same solution applied to I has a maximum load of at most λ.*

Proof: The second claim is obvious since the jobs in I have smaller weights than the corresponding jobs in I^+. As for the first claim, we first consider the jobs we move from one of their "illegal machines" (machines on which $\bar{w}'_j(i) = \infty$ to their fastest machine. Consider a machine i. Jobs might have been moved to this machine from all the other $m-1$ machines. Each of these machines had a load of at most λ. A job j was moved to machine i only if it was assigned to a machine k s.t. $\bar{w}_j(k) \geq \frac{m}{\epsilon} w_j(i)$. So when moving the jobs to machine i, their weights are $\frac{m}{\epsilon}$ times smaller. This means that the total increase in the load of machine i is at most $(m-1)\cdot\frac{\epsilon}{m}\cdot\lambda < \epsilon\lambda$. Hence, the maximum load was increased by at most $\epsilon\lambda$ by the moving of jobs.

In the weight-rounding phase we also increase the weight of each job on its "legal machines". At most, we multiply the weight of each job by $1+\epsilon$. Altogether, these two changes mean that our solution for I has a maximum load of at most $\lambda(1+\epsilon)^2$. ■

Lemma 4.2 *Given a solution for the problem I^+ whose maximum load is λ, the same solution applied to I' has a maximum load of at most $\lambda + \epsilon + \epsilon^4/16$. Also, given a solution* *for I' whose maximum load is λ, the same solution applied to I^+ has a maximum load of at most λ.*

Proof: The second claim is obvious since the durations of jobs in I^+ are contained in the corresponding durations in I'. As for the first claim, for each subset $J(= J_{\bar{v}'})$, every time τ is contained in at most $\lceil \log n \rceil$ sets J_i. Consider the added load at τ from jobs in a certain set J_i. If $\tau < s_i^1$ or $\tau \geq t_i^1$ then the same load is caused by J_i' and J_i. Assume $\tau < c_i$ and define $s_i^{k_i+1} = c_i$, the other case is symmetrical. Then for some (single) j, $s_i^j \leq \tau < s_i^{j+1}$ and the added load at τ is at most the total load of S_i^j which has a min-weight of at most $\alpha + \gamma$, so its total load is at most $\frac{m}{\epsilon}(\alpha + \gamma)$ on any machine. Summing on all sets J_i, we conclude that the maximal load of the jobs in the subset J has increased by at most $\frac{m}{\epsilon}(\alpha+\gamma)\lceil \log n \rceil$. We have at most $(\lceil \log_{1+\epsilon}(\frac{m}{\epsilon}) \rceil + 1)^m$ such subsets, so the overall increase in load is at most

$$(\lceil \log_{1+\epsilon}(\frac{m}{\epsilon}) \rceil + 1)^m \cdot \frac{m}{\epsilon}(\alpha + \gamma)\lceil \log n \rceil = \epsilon + \frac{\epsilon^4}{m^4} \leq \epsilon + \frac{\epsilon^4}{16}$$

. ■

Lemma 4.3 *Given a solution for I' whose maximum load is λ, the same solution applied to I'' has a maximum load of at most $\lambda(1+\epsilon)$. Also, given a solution for I'' whose maximum load is λ, the same solution applied to I' has a maximum load of at most λ.*

Proof: The second claim is obvious since the durations of jobs in I' are contained in the corresponding durations in I''. As for the first claim, given a time τ and a pair of sets S'^j_i, T'^j_i from J_i' from a certain subset $J'(= J'_{\bar{v}'})$, we examine the increase in load at τ. If $\tau < s_i^j$ or $\tau \geq t_i^j$ it is not affected by the transformation because no job in $T'^j_i \cup S'^j_i$ arrives before s_i^j or departs after t_i^j. Assume that $\tau < c_i$, the other case is symmetrical. So τ is affected by the decrease in arrival time of jobs in T'^j_i. It is clear that the way we extend the jobs in T'^j_i increases the the min-weight at τ by at most β, and hence the load at τ is increased by at most $\frac{m}{\epsilon}\beta$. Also, since $\tau \geq s_i^j$, we know that the total min-weight of S'^j_i is at least α if $j < k_i$. Thus, an extra load of at most $\frac{m}{\epsilon}\beta$ is created by every pair S'^j_i, T'^j_i for $1 \leq j < k_i$ only if the pair contributes at least α to the load. If the last pair $S_i^{k_i}$, $T_i^{k_i}$ has total min-weight smaller than α, it does not contribute, as it is not changed from J' to J''; otherwise the analysis is the same as for $j < k_i$. Since the total load on each machine at any time is at most λ, the increase in maximum load is at most $\frac{m}{\epsilon}\beta \cdot \frac{\lambda m}{\alpha} = \epsilon\lambda$. ■

Lemma 4.4 *Given a solution for I'' whose maximum load is λ, the modified problem I''' has a solution with a maximum load of $\lambda(1 + \epsilon + \epsilon^2)$. Also, given a solution for I''' whose maximum load is λ, the solution given by the converting phase for the problem I'' has a maximum load of at most $\lambda(1 + \epsilon + \epsilon^2)$.*

Proof: Consider a solution for I'' whose maximum load is λ. We consider the jobs having min-weight smaller than γ. Let x be the total min-weight of such jobs in a certain $J''_{st}(= J''_{\bar{v}'\,st})$ that were assigned to machine i. We replace them by $\lceil x/\gamma \rceil$ jobs with a loads vector of $\gamma \cdot \bar{v}'$, so that this is an assignment to I'''. The increase in load on every machine is at most $\frac{m}{\epsilon}\gamma$ times the number of sets J''_{st} that contain jobs which are scheduled on that machine. As for the other direction, consider a solution whose maximum load is λ to I'''. The increase in load on every machine by the replacement described in the algorithm is also at most $\frac{m}{\epsilon}\gamma$ times the number of sets J''_{st} that contain jobs which are scheduled on that machine.

It remains to estimate the number of sets J''_{st} that can coexist at a certain time. Most of these sets have a min-weight of at least β; their number is at most $\lambda m/\beta$, since the total load at any time is at most λ on each machine. For each set S_i^j and T_i^j, $j < k_i$, we have at most one set J''_{st} with total min-weight less than β. Since the total min-weight of S_i^j and T_i^j is at least α, there are at most $\lambda m/\alpha$ such sets (if S_i^j and T_i^j are not disjoint, the small sets J''_{st} in both of them have the same s and t, thus we do not need to multiply by 2). Last, there may be one set J''_{st} with a min-weight smaller than β in each $S_i^{k_i} = T_i^{k_i}$; there are only $\lceil \log n \rceil$ such sets in each subset, so their total number is at most $(\lceil \log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)^m \cdot \lceil \log n \rceil$ Therefore, the increase in maximum load is at most:

$$\begin{aligned} \frac{m\gamma}{\epsilon}\left(\frac{\lambda m}{\beta} + \frac{\lambda m}{\alpha} + (\lceil \log_{1+\epsilon}(\frac{m}{\epsilon})\rceil + 1)^m \cdot \lceil \log n \rceil\right) &= \epsilon\lambda + \frac{\epsilon^3\lambda}{m^2} + \frac{\epsilon^5}{m^4} \\ &\leq \lambda(\epsilon + \epsilon^2) \end{aligned}$$

■

Theorem 4.5 *The algorithm described in the last section is a PTAS running in time* $O(n^{1+\epsilon^{-6}m^7(\lceil \log_{1+\epsilon}(\frac{m}{\epsilon})\rceil+1)^m \log m})$.

Proof: First we estimate the approximation ratio of the algorithm. We are given some $\epsilon' > 0$, and we want to find a solution with a maximum load of at most $\lambda(1+\epsilon')$. We use the algorithm described above for $\epsilon = \epsilon'/7$. By the above lemmas, for an instance with optimal solution with maximum load λ, the algorithm yields a solution with maximum load at most:

$$\lambda(1+\epsilon)^2(1+\epsilon+\frac{\epsilon^4}{16})(1+\epsilon)(1+\epsilon+\epsilon^2)^2 < \lambda(1+\epsilon')$$

.

Now we estimate the running time of the algorithm. All the phases except the solving phase are easily performed in time $O(n^2)$. The running time of the solving phase takes the major part of the overall running time. It is linear in the number of edges in the layers graph that we built for the modified problem I"'. The number of edges is linear in the number of vertices (as was explained in our construction). We therefore estimate the number of vertices. Every layer in the graph stores all the possible assignments of jobs to machines. We have $2n$ layers, one for each departure or arrival. Since the minimal load that a job may cause in the modified problem is γ, the maximum number of active jobs at a certain time is $\lambda m/\gamma$. Recall that the scaling we performed at the beginning of our algorithm assures us that $\lambda \leq m$. So the maximum number of vertices in the graph and the running time of the algorithm is:

$$\begin{aligned} 2nm^{\lambda m/\gamma} &\leq 2nm^{\epsilon^{-6}m^7(\lceil \log_{1+\epsilon}(\frac{m}{\epsilon})\rceil+1)^m \lceil \log n \rceil} \\ &= O(n^{1+\epsilon^{-6}m^7(\lceil \log_{1+\epsilon}(\frac{m}{\epsilon})\rceil+1)^m \log m}) \end{aligned}$$

This yields the result. ■

4.2 Non-Fixed Number of Machines

In this section we consider offline load-balancing of temporary tasks on a non-fixed number of unrelated machines. We show that no polynomial approximation algorithm can achieve an approximation ratio smaller than 2, unless $P = NP$. We prove this lower bound for the special case of restricted assignment (so it obviously holds for the general case of any unrelated machines as well).

Theorem 4.6 *For every $\rho < 2$, there does not exist a polynomial ρ-approximation algorithm for restricted assignment of temporary tasks unless $P = NP$.*

Proof: We use a reduction from the 3-dimensional-matching problem ($3DM$), which is known to be NP-complete. In that problem, we are given three sets of elements, B, G, and H, each of them of size n ($B = b_1, ..., b_n$, $G = g_1, ..., g_n$, $H = h_1, ..., h_n$). We are also given a set $S = T_1, ..., T_m$ of m triplets, $S \subseteq B * G * H$. These are the possible matchings of 3 elements from B,G, and H. The goal is to decide whether there exists a matching for all the elements of B,G, and H, i.e. a subset of S, S', such that $|S'| = n$ and $\bigcup_{T_i \in S'} T_i = B \cup G \cup H$. Given an instance to the $3DM$ problem we construct an instance for our problem. For each triplet T_i, we have a machine m_i. All our jobs are of weight 1, and our total time interval will be of length 3 (from time 0 until time 3). Our first type of jobs will be "element jobs" (one job for each element), as described hereafter. For each element $b_i \in B$, we will have a job which arrives at time 0, departs at time 1, and is admissible to a machine m_k if and only if $b_i \in T_k$. For each element $g_i \in G$, we will have a job which arrives at time 1, departs at time 2, and is admissible to a machine m_k if and only if $g_i \in T_k$. Finally, for each element $h_i \in H$, we will have a job which arrives at time 2, departs at time 3, and is admissible to a machine m_k if and only if $h_i \in T_k$. Our second type of jobs will constitute of $m - n$ "dummy jobs", which arrive at time 0, depart at time 3, and are admissible to all the machines.

We prove that there is an assignment with a maximum load of 1, if and only if there is a solution to the $3DM$ problem. Suppose there is a $3DM$, $S' = \{T_{i_1}, ..., T_{i_n}\}$. Then for each $T_{i_k} \in S'$, we assign to the machine m_{i_k} the 3 "element jobs" which correspond to the 3 elements of T_{i_k}. They are admissible to m_{i_k}, because this is how we defined our assignment restrictions. Also notice that they are active in different times, so the machine maintains a load of 1. This way we assign the $3n$ "element jobs" to n of the machines. We assign the $m - n$ "dummy jobs" to the other $m - n$ machines, one job on each machine. They are admissible on any machine, and have a weight of 1 each. Therefore, this assignment maintains a maximum load of 1 as required.

Now, assume that there is an assignment having a maximum load of 1. The $m - n$ "dummy jobs" must have been assigned to $m - n$ different machines (if there is more than one "dummy job" on a machine, then its load is bigger than 1). A "dummy job" is active during our entire time interval, so a machine which has a "dummy job" on it cannot have any other job assigned to it. Therefore, the $3n$ "element jobs" must have been assigned to the remaining n machines. Each of these machines $m_{i_1}, ..., m_{i_n}$ must have one active "element job" on it at each moment (since the total volume of the "element jobs" is $3n$). This means that each of these machines, m_{i_k}, has a job which corresponds to an element of B assigned to it, a job which corresponds to an element of G assigned to it, and a job which corresponds to an element of H assigned to it (this is the only possibility to have an active "element job" at each moment). According to our assignment restrictions, these 3 elements from B,G, and H must be included in the triplet T_{i_k}. All of the "element jobs" were assigned, so $\bigcup_{T_{i_k}} T_i = B \cup G \cup H$, and therefore $T_{i_1}, ..., T_{i_n}$ is a $3DM$.

The above reduction shows that any approximation algorithm for our problem with an approximation ratio strictly less than 2 solves the $3DM$ problem. Hence, we have proven the theorem. ■

References

[1] S. Albers. Better bounds for on-line scheduling. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 130–139, 1997.

[2] Jim Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 623–631, May 1993.

[3] Y. Azar. On-line load balancing. In A. Fiat and G. Woeginger, editors, *Online Algorithms - The State of the Art*, chapter 8, pages 178–195. Springer, 1998.

[4] Y. Azar, A. Z. Broder, and A. R. Karlin. On-line load balancing. *Theoretical Computer Science*, 130(1):73–84, 1994. Also in *Proc. 33rd IEEE FOCS*, 1992, pp. 218-225.

[5] Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. In *5th Israeli Symp. on Theory of Computing and Systems*, pages 119–125, 1997.

[6] Y. Azar, B. Kalyanasundaram, S. Plotkin, Kirk R. Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22(1):93–110, 1997. Also in *Proc. WADS'93*, pp. 119-130.

[7] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18(2):221–237, 1995. Also in *Proc. 3rd ACM-SIAM SODA*, 1992, pp. 203-210.

[8] Y. Azar, O. Regev, J. Sgall, and G. Woeginger. Off-line temporary tasks assignment. *Theoretical Computer Science.* To appear. Also in *Proc. 7th Annual European Symposium on Algorithms* 1999, pp. 163-171.

[9] B. S. Baker, D. J. Brown, and H. P. Katseff. A 5/4 algorithm for two-dimensional packing. *J. Algorithms*, 2:348–368, 1981.

[10] 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. Also in *Journal of Computer and System Sciences* (1995) 359-366.

[11] P. Berman, M. Charikar, and M. Karpinski. A note on on-line load balancing for related machines. In *5th annual Workshop on Algorithms and Data Structures*, 1997.

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

[13] B. Chen, A. van Vliet, and G. J. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16:221–230, 1994.

[14] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[15] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. C. Yao. Resource constrained scheduling as generalized bin packing. *J. Comb. Th. Ser. A.*, 21:257–298, 1976.

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

[17] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non-identical processors. *Journal of the Association for Computing Machinery*, 23:317–327, 1976.

[18] D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 132–140, 1994.

[19] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Prog.*, 46:259–271, 1990.

[20] S. Phillips and J. Westbrook. On-line load balancing and network flow. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 402–411, 1993.

[21] S. Plotkin and Y. Ma. An improved lower bound for load balancing of tasks with unknown duration. *Inform. Process. Lett.*, 62:301–303, 1997.

[22] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461–474, 1993. Also in the proceeding of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993.

[23] J. Westbrook. Load balancing for response time. In *3rd Annual European Symposium on Algorithms*, 1995.