Off-line Temporary Tasks Assignment *

Yossi Azar † **Oded Regev** ‡ **Jiří Sgall** §

Gerhard J. Woeginger ¶

May 3, 2000

Abstract

In this paper we consider the temporary tasks assignment problem. In this problem, there are m parallel machines and n independent jobs. Each job has an arrival time, a departure time and some weight. Each job should be assigned to one machine. The load on a machine at a certain time is the sum of the weights of jobs assigned to it at that time. The objective is to find an assignment that minimizes the maximum load over machines and time.

We present a polynomial time approximation scheme for the case in which the number of machines is fixed. We also show that for the case in which the number of machines is given as part of the input (i.e., not fixed), no polynomial algorithm can achieve a better approximation ratio than $\frac{3}{2}$ unless $P = NP$.

*A preliminary version of this paper appears in the proceedings of the 7th European Symp. on Algorithms (ESA), Lecture Notes in Comput. Sci. 1643, pages 163-171. Springer-Verlag, 1999.

†Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel. Research supported in part by the Israel Science Foundation and by the US-Israel Binational Science Foundation (BSF). E-Mail: azar@math.tau.ac.il

‡Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel. E-Mail: odedr@math.tau.ac.il

§Mathematical Inst., AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic; and Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha. Partially supported by grant A1019901 of GA AV ČR, postdoctoral grant 201/97/P038 of GA ČR, and cooperative research grant INT-9600919/ME-103 from the NSF and MŠMT ČR. E-Mail: sgall@math.cas.cz

¶Inst. für Mathematik, TU Graz, Steyrergasse 30, A-8010 Graz, Austria. This research has been supported by the START program Y43-MAT of the Austrian Ministry of Science. E-Mail: gwoegi@opt.math.tu-graz.ac.at

1 Introduction

We consider the off-line problem of non-preemptive load balancing of temporary tasks on m identical machines. Each job has an arrival time, departure time and some weight. Each job should be assigned to one machine. The load on a machine at a certain time is the sum of the weights of jobs assigned to it at that time. The goal is to minimize the maximum load over machines and time. Note that the weight and the time are two separate axes of the problem.

The load balancing problem naturally arises in many applications involving allocation of resources. As a simple concrete example, consider the case where each machine represents a communication channel with bounded bandwidth. The problem is to assign a set of requests for bandwidth, each with a specific time interval, to the channels. The utilization of a channel at a specific time t is the total bandwidth of the requests, whose time interval contains t, which are assigned to this channel.

Load balancing of permanent tasks is the special case in which jobs have neither an arrival time nor a departure time. This special case is also known as the classical scheduling problem which was first introduced by Graham [5, 6]. He described a greedy algorithm called "List Scheduling" which has a $2 - \frac{1}{m}$ approximation ratio where m is the number of machines. Interestingly, the same analysis holds also for load balancing of temporary tasks. However, until now, it was not known whether better approximation algorithms for temporary tasks exist.

For the special case of permanent tasks, there is a polynomial time approximation scheme (PTAS) for any fixed number of machines [6, 10] and also for arbitrary number of machines by Hochbaum and Shmoys [7]. That is, it is possible to obtain a polynomial time $(1+\epsilon)$-approximation algorithm for any fixed $\epsilon > 0$.

In contrast we show in this paper that the model of load balancing of temporary tasks behaves differently. Specifically, for the case in which the number of machines is fixed we present a PTAS. However, for the case in which the number of machines is given as part of the input, we show that no algorithm can achieve a better approximation ratio than $\frac{3}{2}$ unless $P = NP$. (A weaker lower bound of $\frac{4}{3}$ was presented in the proceedings version of this paper.)

Note that similar phenomena occur at other scheduling problems. For example, for scheduling (or equivalently, load balancing of permanent jobs) on unrelated machines, Lenstra et al. [9] showed a PTAS for a fixed number of machines. On the other hand, they showed that if the number of machines is part of the input then no algorithm with an approximation ratio better than $\frac{3}{2}$ can exist unless $P = NP$.

In contrast to our result, in the on-line setting it is impossible to improve the performance of Graham's algorithm for temporary tasks even for a fixed number of machines. Specifically, it is shown in [2] that for any m there is a lower bound of $2 - \frac{1}{m}$ on the performance ratio of any on-line algorithm (see also [1, 3]).

Our algorithm works in four phases: the rounding phase, the combining phase, the solving phase and the converting phase. The rounding 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. In the combining phase, we combine several jobs with the same arrival and departure time and unite them into jobs with higher weights. Solving the resulting assignment problem in the solving phase is easier and its solution can be converted into a solution for the original problem in the converting phase.

The novelty of our algorithm is in the rounding phase. Standard rounding techniques are usually performed on the weights. If one applies similar techniques to the time the resulting algorithm's running time is not polynomial. Thus, we had to design a new rounding technique in order to overcome this problem.

Our lower bound is proved directly by a reduction from edge-coloring of cubic graphs. It remains as an open problem whether one can improve the lower bound using more sophisticated techniques such as PCP reductions.

2 Notation

We are given a set of n jobs that should be assigned to one of m identical machines. 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, w.l.o.g. 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 any job. We denote the weight of job j by w_j, its arrival time by a_j and its departure time by d_j. We say that a job is active at time τ if $a_j \leq \tau < d_j$. An assignment algorithm for the temporary tasks problem has to assign each job to a machine.

Let $Q_i = \{j \mid a_j \leq \sigma_i < d_j\}$ be the active jobs at time σ_i. For a given algorithm A let A_j be the machine on which job j is assigned. Let

$$l_k^A(i) = \sum_{\{j \mid A_j = k, j \in Q_i\}} w_j$$

be the load on machine k at time σ_i, which is the sum of weights of all jobs assigned to k and active at this time. The cost of an algorithm A is the maximum load ever achieved by any of the machines, i.e., $C_A = max_{i,k} l_k^A(i)$. We compare the performance of an algorithm to that of an optimal algorithm and define the approximation ratio of A as r if for any sequence $C_A \leq r \cdot C_{opt}$ where C_{opt} is the cost of the optimal solution.

3 The Polynomial Time Approximation Scheme

Assume without loss of generality that the optimal maximum load is in the range $(1, 2]$. That is possible since Graham's algorithm can approximate the optimal solution up to a factor of 2, and thus, we can scale all the jobs' weights by $2/l$ where l denotes the value of Graham's solution. This does not increase the running time of the scheme, since Graham's algorithm runs in linear time (for fixed m).

Let $\epsilon > 0$ be a constant, depending on the required precision (we will determine it later). We fix three constants, $\alpha = \epsilon / \lceil \log n \rceil$, $\beta = \alpha\epsilon/m = \epsilon^2/(m\lceil \log n \rceil)$, and $\gamma = \beta\epsilon/m = \epsilon^3/(m^2\lceil \log n \rceil)$.

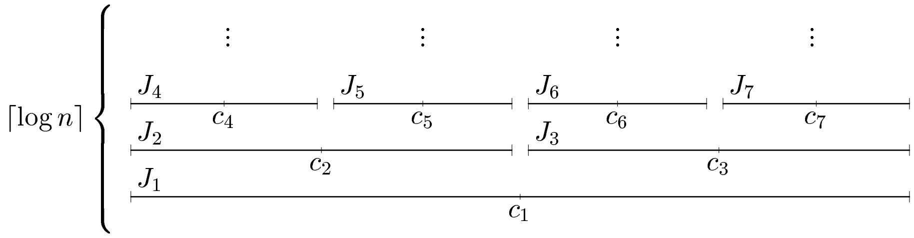

Figure 1: Partitioning $J - R$ into $\{J_i\}$

In order to describe the rounding phase with its two subphases we begin with defining the partitions based on which the rounding will be performed.

The set R contains all jobs with weight at least γ. 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 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 that are active at a certain time is partitioned into at most $\lceil \log n \rceil$ different sets J_i.

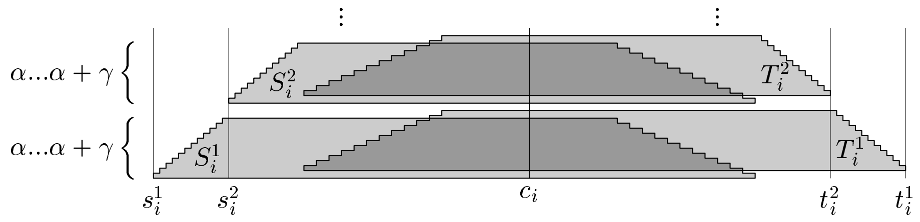

Figure 2: Partitioning J_i into $\{S_i^j, T_i^j\}$

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 weight is at least α by S_i^1. We order the same jobs as before according to their departure time. We take the smallest suffix whose 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 weight between α and $\alpha + \gamma$, except for the last pair which may have smaller weight than α. However, if the last pair has smaller 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 $s_i^1 \leq s_i^2 \leq ... \leq s_i^{k_i} \leq c_i \leq t_i^{k_i} \leq ... \leq t_i^2 \leq t_i^1$.

The first subphase of the 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, ${S'}^j_i$, ${T'}^j_i$.

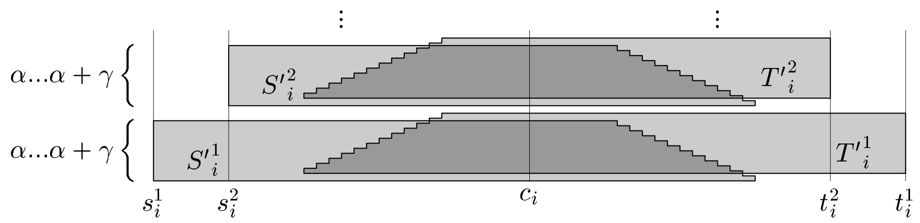

Figure 3: The set J'_i (after the first subphase)

The second subphase of the 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 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 weight of less than β. Similarly, extend the arrival times of jobs in ${T'}^j_i$. Note that if the weight of the last pair 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, ${S''}^j_i$, ${T''}^j_i$.

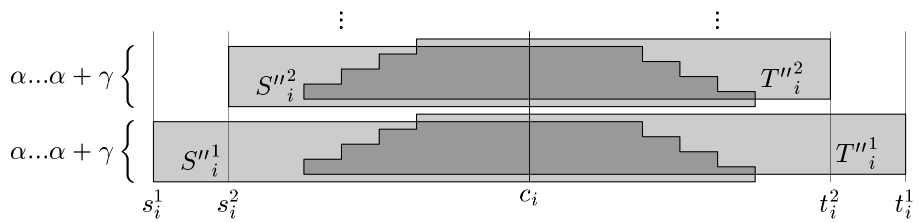

Figure 4: The set J''_i (after the rounding phase)

The combining phase of the algorithm involves the weight of the jobs. Let

J''_{st} be the set of jobs in J'' that arrive at s and depart at t. Assume the total weight of jobs whose weight is at most γ in J''_{st} is x. The combining phase replaces these jobs by $\lceil x/\gamma \rceil$ jobs of weight γ. 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, $J''' = \bigcup_{s,t} J'''_{st}$.

The solving phase of the algorithm solves the modified decision problem of J''' 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; furthermore, we label each node by the maximum load of a machine in that configuration. The first and last layers contain a single vertex, as there are no jobs at that point. These vertices are called a source and a sink.

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

In the converting phase the algorithm converts the assignment found for J''' into an assignment for J. Assume the number of jobs of weight γ in J'''_{st} that are assigned to a certain machine i is r_i. Remove these jobs and assign all the jobs smaller than γ in J''_{st} such that a total weight of at most $(r_i + 1)\gamma$ is assigned to each machine. This is possible since the replacement involves jobs whose weight is at most γ and from volume consideration there is always at least one machine with a load of at most $r_i\gamma$ of these jobs. The assignment for J'' is also an assignment for J' and J.

4 Analysis

Lemma 4.1 *Given a solution for the original problem J whose maximum load is λ, the same solution applied to J' has a maximum load of at most $\lambda + \epsilon + \epsilon^3/4$. Also, given a solution for J' whose maximum load is λ, the same solution applied to J has a maximum load of at most λ.*

Proof: The second claim is obvious since the jobs in J are shorter than the corresponding jobs in J'. As for the first claim, 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 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 is at most $\alpha + \gamma$. Summing on all sets J_i, we conclude that the maximal load has increased by at most $(\alpha + \gamma)\lceil \log n \rceil = \epsilon + \epsilon^3/m^2$. ■

Lemma 4.2 *Given a solution for J' whose maximum load is λ, the same solution applied to J'' has a maximum load of at most $\lambda(1+\epsilon)$. Also, given a solution for J'' whose maximum load is λ, the same solution applied to J' has a maximum load of at most λ.*

Proof: The second claim is obvious since the jobs in J' are shorter than the corresponding jobs in J''. As for the first claim, given a time τ and a pair of sets ${S'}_i^j$, ${T'}_i^j$ from J_i' 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 ${T'}_i^j \cup {S'}_i^j$ 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 ${T'}_i^j$. It is clear that the way we extend the jobs in ${T'}_i^j$ increases the load at τ by at most β. Also, since $\tau \geq s_i^j$, we know that the load caused by ${S'}_i^j$ is at least α if $j < k_i$. Thus, an extra load of at most β is created by every pair ${S'}_i^j$, ${T'}_i^j$ 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 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 all machines at any time is at most λm, the increase in load of any machine and therefore in maximum load is at most $\beta \cdot \lambda m/\alpha = \epsilon\lambda$. ■

Lemma 4.3 *Given a solution for J'' whose maximum load is λ, the modified problem J''' has a solution with a maximum load of $\lambda(1+\epsilon+\epsilon^2)$. Also,* *given a solution for J''' whose maximum load is λ, the solution given by the converting phase for the problem J'' has a maximum load of at most $\lambda(1+\epsilon+\epsilon^2)$.*

Proof: Consider a solution for J'' whose maximum load is λ. If the load of jobs smaller than γ in a certain J''_{st} on a certain machine i is x, we replace it by at most $\lceil x/\gamma \rceil$ jobs of weight γ so that this is an assignment to J'''. The increase in load on every machine is at most γ 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 J'''. The increase in load on every machine by the replacement described in the algorithm is also at most γ 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 weight at least β; their number is at most $\lambda m/\beta$, since the total load at any time is at most λm. For each set S_i^j and T_i^j, $j < k_i$, we have at most one set J''_{st} with weight less than β, since the 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} smaller than β in each $S_i^{k_i} = T_i^{k_i}$; there are only $\lceil \log n \rceil$ of such sets. Therefore, the increase in maximum load is at most $\gamma(\lambda m/\beta + \lambda m/\alpha + \lceil \log n \rceil) = \epsilon\lambda + \epsilon^2\lambda/m + \epsilon^3/m^2 \leq \lambda(\epsilon+\epsilon^2)$. ∎

Theorem 4.4 *The algorithm described in the last section is a PTAS running in time $O(n^{c\epsilon^{-3}m^3 \log m})$, where c is some absolute constant.*

Proof: We are given some $\epsilon' > 0$ and want to find a solution with maximum load at most $\lambda(1+\epsilon')$. If $\epsilon' \geq 1$, we use Graham's List Scheduling. Otherwise we use the algorithm described above for $\epsilon = \epsilon'/6$. For an instance with optimal solution with maximum load λ, the algorithm yields a solution with maximum load at most $\lambda(1+\epsilon+\epsilon^3/4)(1+\epsilon)(1+\epsilon+\epsilon^2)^2 < \lambda(1+\epsilon')$.

Every layer in the graph stores all the possible assignments of jobs to machines. Since the smallest job is of weight γ, the maximum number of active jobs at a certain time is $\lambda m/\gamma$. So, the maximum number of edges in the graph and the running time of the algorithm is $nm^{\lambda m/\gamma} \leq nm^{2\epsilon^{-3}m^3\lceil \log n \rceil} = O(n^{1+2\epsilon^{-3}m^3 \log m})$. This yields the result, with the constant c bounded by $c < 2 \cdot 6^3 + 1 = 433$. ∎

5 The unrestricted number of machines case

In this section we show that in case the number of machines is given as part of the input, the problem cannot be approximated up to a factor of 3/2 in polynomial time unless $P = NP$.

Theorem 5.1 *For every $\rho < \frac{3}{2}$, there does not exist a polynomial time ρ-approximation algorithm for the temporary tasks assignment problem unless $P = NP$.*

Proof: The proof is by reduction from edge-coloring of cubic graphs (cubic means that all vertices have degree three): A *feasible* edge 3-coloring of a simple cubic graph $G = (V, E)$ is a coloring of E with the colors 1, 2 and 3, such that for every vertex the incident edges receive three distinct colors. Deciding whether a given cubic graph $G = (V, E)$ possesses a feasible edge 3-coloring is NP-complete [4]. Since G is cubic, $|V| = 2q$ and $|E| = 3q$ holds for some positive integer q. Moreover, since in a feasible edge 3-coloring of G every color class forms a perfect matching, every color will occur exactly q times.

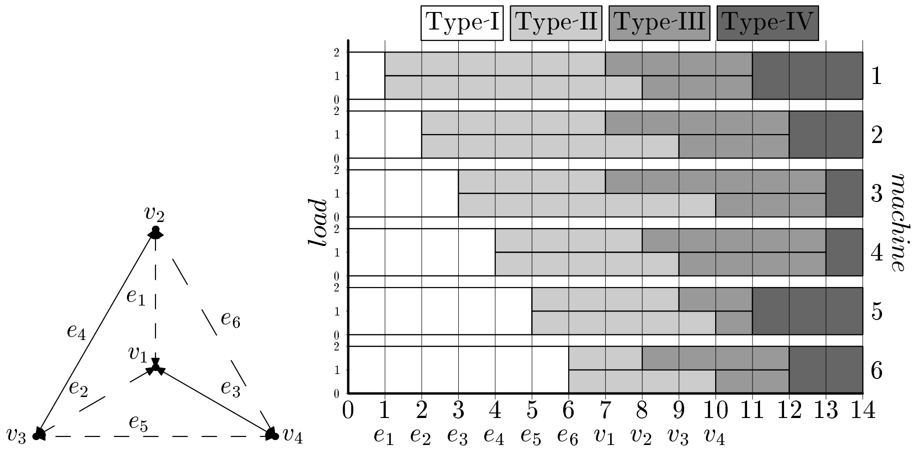

Figure 5: A 3-colorable cubic graph and the corresponding assignment

Given a graph G we describe an instance of the load balancing problem. Let $e_1, \ldots, e_{3q}$ be an arbitrary enumeration of the edges in E, and let $v_1, \ldots, v_{2q}$ be an arbitrary enumeration of the vertices in V. We construct an instance of the temporary task assignment problem with $m = 3q$ machines and $n = 18q$ jobs.

- For every edge e_i ($i = 1, \ldots, 3q$), there is a corresponding job of weight 2 starting at time 0 and ending at time i.

- Let v_j ($j = 1, \ldots, 2q$) be a vertex, and let e_x, e_y and e_z be the three edges incident to v_j. Then there are six jobs $J_{j,x}$, $J_{j,y}$, $J_{j,z}$ and $K_{j,1}$, $K_{j,2}$, $K_{j,3}$ that correspond to v_j and that all have weight 1. The jobs $J_{j,x}$, $J_{j,y}$, $J_{j,z}$ start at times x, y, and z, respectively, and all end at time $3q + j$. The jobs $K_{j,1}$, $K_{j,2}$, $K_{j,3}$ start at time $3q + j$ and end at times $5q + 1$, $5q + 2$, and $5q + 3$, respectively.

- Finally, there are $3q$ dummy jobs that all have weight 2. The dummy jobs are divided into three classes of q jobs. The q dummy jobs in class c ($c = 1, 2, 3$) start at time $5q + c$ and end at time $5q + 4$.

This completes the description of the scheduling instance. We claim that this scheduling instance possesses a schedule with maximum load 2 if and only if the graph G possesses a feasible edge 3-coloring.

Proof of the "if" part. Suppose that the graph G possesses a feasible edge 3-coloring. Let $e_i = [v_j, v_k]$ be an edge in E that receives color $c \in \{1, 2, 3\}$ in this coloring. The following jobs are processed on machine i: From time 0 to time i, process the job that corresponds to edge e_i. From time i to time $3q + j$ process job $J_{j,i}$, and from time $3q + j$ to time $5q + c$, process job $K_{j,c}$. Analogously, from time i to time $3q + k$ process job $J_{k,i}$, and from time $3q + k$ to time $5q + c$, process job $K_{k,c}$. Finally, from time $5q + c$ to time $5q + 4$ process a dummy job from class c. It is easy to see that in this schedule all jobs are processed, and at any moment in time every machine has load at most 2.

Proof of the "only if" part. Now assume that there is a schedule with maximum load 2. Note that at any moment τ in time, $0 \leq \tau \leq 5q + 4$, the total available load is exactly $6q = 2m$. Hence, in a schedule with maximum load 2 every machine must be busy all the time. Without loss of generality we assume that for $1 \leq i \leq 3q$, machine i processes the job corresponding to edge e_i. Moreover, at time $5q+4$ machine i completes one of the dummy jobs. We color edge e_i by the class c of this dummy job.

We claim that in the resulting coloring, every vertex is incident to three differently colored edges: Suppose otherwise. Then there exist two edges $e_x = [v_j, v_k]$ and $e_y = [v_j, v_\ell]$ that receive the same color c. Consider machine

x at time x in the schedule. The job corresponding to e_x ends at time x, and the only available jobs are $J_{j,x}$ and $J_{k,x}$. Since machine x is busy all the time, it must process these two jobs. Consider machine x at time $3q + j$ in the schedule. The processing of job $J_{j,x}$ ends, and the machine must process some job from time $3q+j$ to time $5q+c$; the only possible job for this is job $K_{j,c}$. By similar arguments we get that job $K_{j,c}$ is simultaneously processed on machine y, a contradiction. Hence, the constructed edge-coloring indeed is a feasible edge 3-coloring. ∎

References

[1] 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.

[2] 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.

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

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, San Francisco, 1979.

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

[6] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:263–269, 1969.

[7] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. of the ACM*, 34(1):144–162, January 1987.

[8] R.M. Karp. *Reducibility among Combinatorial Problems, R.E. Miller and J.W. Thatcher (eds.), Complexity of Computer Computations.* Plenum Press, 1972.

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

[10] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23:116–127, 1976.