Approximating the Throughput of Multiple Machines in Real-Time Scheduling*

Amotz Bar-Noy† Sudipto Guha‡ Joseph (Seffi) Naor§ Baruch Schieber ¶

Abstract

We consider the following fundamental scheduling problem. The input to the problem consists of n jobs and k machines. Each of the jobs is associated with a release time, a deadline, a weight, and a processing time on each of the machines. The goal is to find a schedule that maximizes the weight of jobs that meet their deadline. We give constant factor approximation algorithms for four variants of the problem, depending on the type of the machines (identical vs. unrelated), and the weight of the jobs (identical vs. arbitrary). All these variants are known to be NP-Hard, and we observe that the two variants involving unrelated machines are also MAX-SNP hard. To the best of our knowledge, these are the first approximation algorithms for such problems in the non-preemptive off-line setting. The specific results obtained are:

- For identical job weights and unrelated machines: a greedy 2-approximation algorithm.
- For identical job weights and k identical machines: the same greedy algorithm achieves a tight $\frac{(1+1/k)^k}{(1+1/k)^k-1}$-approximation factor.
- For arbitrary job weights and a single machine: an LP formulation achieves a 2-approximation for polynomially bounded integral input and a 3-approximation for arbitrary input. For unrelated machines, the factors are 3 and 4 respectively.
- For arbitrary job weights and k identical machines: the LP based algorithm applied repeatedly achieves a $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor for polynomially bounded integral input and a $\frac{(1+1/2k)^k}{(1+1/2k)^k-1}$ approximation factor for arbitrary input.
- For arbitrary job weights and unrelated machines: a combinatorial $(3 + 2\sqrt{2} \approx 5.828)$-approximation algorithm.

*Part of this work was done while the first three authors visited IBM T.J. Watson Research Center.

†Electrical Engineering Department, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: *amotz@eng.tau.ac.il*.

‡Computer Science Department, Stanford University, Stanford, CA 94305. Supported by an IBM Cooperative fellowship, NSF Grant IIS-9811904 and NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. E-mail: *sudipto@cs.stanford.edu*.

§Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974. On leave from the Computer Science Department, Technion, Haifa 32000, Israel. E-mail: *naor@research.bell-labs.com*.

¶IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. E-mail: *sbar@watson.ibm.com*.

1 Introduction

We consider the following fundamental scheduling problem. The input to the problem consists of n jobs and k machines. Each of the jobs is associated with a release time, a deadline, a weight, and a processing time on each of the machines. The goal is to find a schedule that maximizes the weight of the jobs that meet their deadline. Such scheduling problems are frequently referred to as *real-time* scheduling problems, and the objective of maximizing the value of completed jobs is frequently referred to as *throughput*. We consider four variants of the problem depending on the type of the machines (identical vs. unrelated) and the weight of the jobs (identical vs. arbitrary).

Garey and Johnson [13] (see also [14]) show that even the simplest decision problem corresponding to this problem is already NP-Hard in the strong sense. In this decision problem the input consists of a set of n jobs with release time, deadline, and processing time. The goal is to decide whether all these jobs can be scheduled on a single machine; each within its time window. We show that the two variants involving unrelated machines are also MAX-SNP hard.

In this paper we give constant factor approximation algorithms for all four variants of the problem. To the best of our knowledge, this is the first paper that gives approximation algorithms with guaranteed performance (approximation factor) for these problems. We say that an algorithm has an approximation factor ρ for a maximization problem if the weight of its solution is at least $1/\rho \cdot \mathsf{OPT}$, where OPT is the weight of an optimal solution. (Note that we defined the approximation factor so that it would always be at least 1.)

The specific results obtained are listed below and summarized in the table given in Figure 1.

weight function	identical machines	unrelated machines
identical job weights	$(2, 1.8, 1.73, \ldots, \frac{(1+1/k)^k}{(1+1/k)^k-1}, \ldots, 1.58)$	$(2, 2, 2, \ldots, 2)$
arbitrary job weights integral, poly-size, input	$(2, 1.8, 1.73, \ldots, \frac{(1+1/k)^k}{(1+1/k)^k-1}, \ldots, 1.58)$	$(2, 3, 3, \ldots, 3)$
arbitrary job weights arbitrary input	$(3, 2.78, 2.7, \ldots, \frac{(1+1/2k)^k}{(1+1/2k)^k-1}, \ldots, 2.54)$	$(3, 4, 4, \ldots, 4)$

Figure 1: Each entry contains the approximation factors for $k = (1, 2, 3,$ general k, and $k \to \infty)$

- For identical job weights and unrelated machines, we give a greedy 2-approximation algorithm.

- For identical job weights and k identical machines, we show that the same greedy algorithm achieves a tight $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor.
- For arbitrary job weights, we round a fractional solution obtained from a linear programming relaxation of the problem. We distinguish between the case where the release times, deadlines, and processing times, are integral and polynomially bounded, and the case where they are arbitrary. In the former case, we achieve a 2-approximation factor for a single machine, and a 3-approximation factor for unrelated machines. In the latter case, we get a 3-approximation factor for a single machine, and a 4-approximation factor for unrelated machines.
- For arbitrary job weights and k identical machines, we achieve a $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor for polynomially bounded integral input, and a $\frac{(1+1/2k)^k}{(1+1/2k)^k-1}$ approximation factor for arbitrary input. Note that as k tends to infinity these factors tend to $\frac{e}{e-1} \approx 1.58198$, and $\frac{\sqrt{e}}{\sqrt{e}-1} \approx 2.54149$, respectively.
- For arbitrary job weights and unrelated machines we also present a *combinatorial* $(3+2\sqrt{2})$-approximation factor $(3+2\sqrt{2} \approx 5.828)$.

The computational difficulty of the problems considered here is due to the "slack time" available for scheduling the jobs. In general, the time window in which a job can be scheduled may be (much) larger than its processing time. Interestingly, the special case where there is no slack time can be solved optimally in polynomial time even for multiple machines [3]. Moreover, the problem can be solved on a single machine with the execution window less than twice the length of the job.

Another special case that was considered earlier in the literature is the case in which all jobs are released at the same time (or equivalently, the case in which all deadlines are the same). This special case remains NP-Hard even for a single machine. However, Sahni [24] gave a fully polynomial approximation scheme for this special case.

The problems considered here have many applications. Hall and Magazine [17] considered the single machine version of our problem in the context of maximizing the scientific, military or commercial value of a space mission. This means selecting and scheduling in advance a set of projects to be undertaken during the space mission, where an individual project is typically executable during only part of the mission. It is indicated in [17] that up to 25% of the budget of a space mission may be spent in making these decisions. Hall and Magazine [17] present eight heuristic procedures for finding an optimal solution together with computational experiments. However, they do not provide any performance guarantees on the solutions produced by their heuristics. They also mention the applicability of such problems to patient scheduling in hospitals. For more applications and related work in the scheduling literature see [8, 11] and the survey of [22].

The preemptive version of our problem for a single machine was studied by Lawler [21]. For identical job weights, Lawler showed how to apply dynamic programming techniques to solve the problem in polynomial time. He extended the same techniques to obtain a pseudo-polynomial algorithm for arbitrary weights as well ([21]). Lawler [20] also obtained polynomial time algorithms that solve the problem in two special cases: (i) the time windows in which jobs can be scheduled are nested; and (ii) the weights and processing times are in opposite order. Kise, Ibaraki and Mine [18] showed how to solve the special case where the release times and deadlines are similarly ordered.

A closely related problem is considered Adler *et al.* [1] in the context of communication in linear networks. In this problem, messages with release times and deadlines have to be transmitted over a bus that has a unit bandwidth, and the goal is to maximize the number of messages delivered within their deadline. It turns out that our approximation algorithms for the case of arbitrary weights can be applied to the weighted version of the unbuffered case considered in [1] to obtain a constant factor approximation algorithm. No approximation algorithm is given in [1] for this version.

In the on-line version of our problems, the jobs appear one by one, and are not known in advance. Lipton and Tomkins [23] considered the non-preemptive version of the on-line problem, while Koren and Shasha [19] and Baruah *et al.* [7] considered the preemptive version. The special cases where the weight of a job is proportional to the processing time were considered in the on-line setting in several papers [5, 10, 12, 15, 16, 6]. Our combinatorial algorithm for arbitrary weights borrows some of the techniques used in the on-line case.

Some of our algorithms are based on rounding a fractional solution obtained from a linear programming (LP) relaxation of the problem. In the LP formulation for a single machine we have a variable for every feasible schedule of each of the jobs, a constraint for each job, and a constraint for each time point. A naive implementation of this approach would require an unbounded number of variables and constraints. To overcome this difficulty, we first assume that all release times, deadlines, and processing times are (polynomially bounded) integers. This yields a polynomial number of variables and constraints, allowing for the LP to be solved in polynomial time. For the case of arbitrary input, we show that we need not consider more than $O(n^2)$ variables and constraints for each of the n jobs. This yields a strongly polynomial running time at the expense of a minor degradation in the approximation factor. The rounding of the LP is done by reducing the problem to a graph coloring problem.

We extend our results from a single machine to multiple machines by applying the single machine algorithm repeatedly, machine by machine. We give a general analysis for this type of algorithms and, interestingly, prove that the approximation factor for the case of identical machines is superior to the approximation factor of the single machine algorithm which served as our starting point. A similar phenomenon (in a different context) has been observed by Cornuejols, Fisher and Nemhauser [9]. Our analysis in the unrelated machines case is similar to the one described (in a different context) by Awerbuch *et al.* [4]. Unlike the identical machines case, in the unrelated machines case the extension to multiple machines degrades the performance relative to a single machine.

Our algorithms (and specifically our LP based algorithms) can be applied to achieve approximation algorithms for other scheduling problems. For example, consider a problem where we can compute an estimate of the completion time of the jobs in an optimal (fractional) solution. Then, we can apply our algorithms using these estimated completion times as deadlines, and get a schedule where a constant fraction of the jobs indeed finish by these completion times. This observation has already been applied by Wein [25] to achieve constant factor approximation algorithms for various problems, among them the minimum flow-time problem.

2 Definitions and notations

Let the job system contain n jobs $\mathcal{J} = \langle J_1, \ldots, J_n \rangle$ and k machines $\mathcal{M} = \langle M_1, \ldots, M_k \rangle$. Each job J_i is characterized by the quadruple (r_i, d_i, L_i, w_i), where $L_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$. The interpretation is that job J_i is available at time r_i, the *release time*, it must be executed by time d_i, the *deadline*, its *processing time* on machine M_j is $\ell_{i,j}$, and w_i is the *weight* (or *profit*) associated with the job. We note that our techniques can also be extended to the more general case where the release time and deadline of each job differ on different machines. However, for simplicity, we consider only the case where the release time and deadline of each job are the same on all machines. The hardness results are also proved under the same assumption.

We refer to the case in which all job weights are the same as the *unweighted* model, and the case in which job weights are arbitrary as the *weighted* model. (In the unweighted case our goal is to maximize the cardinality of the set of scheduled jobs.) We refer to the case in which the processing times of the jobs on all the machines are the same as the *identical machines* model, and the case in which processing times differ as the *unrelated machines* model. In the unweighted jobs and identical machines model, job J_i is characterized by a triplet (r_i, d_i, ℓ_i) where $d_i \geq r_i + \ell_i$. Without loss of generality, we assume that the earliest release time is at time $t = 0$.

A feasible scheduling of job J_i on machine M_j at time t, $r_i \leq t \leq d_i - \ell_{i,j}$, is referred to as a *job instance*, denoted by $J_{i,j}(t)$. A job instance can also be represented by an *interval* on the time line $[0, \infty)$. We say that the interval $J_{i,j}(t) = [t, t + \ell_{i,j})$ *belongs* to job J_i. In general, many intervals may belong to a job. A set of job instances $J_{1,j}(t_1), \ldots, J_{h,j}(t_h)$ is a feasible schedule on machine M_j, if the corresponding intervals are independent, i.e., they do not overlap, and they belong to distinct jobs. The *weight* of a schedule is the sum of the weights of the jobs to which the intervals (job instances) belong. In the case of multiple machines, we need to find a feasible schedule of distinct jobs on each of the machines. The objective is to maximize the sum of the weights of all schedules.

We distinguish between the case where the release times, processing times, and deadlines are integers bounded by a polynomial in the number of jobs, and between the case of arbitrary inputs. The former case is referred to as *polynomially bounded integral* input and the latter case is referred to as *arbitrary* input.

3 Unweighted jobs

In this section we consider the unweighted model. We define a greedy algorithm and analyze its performance in both the unrelated and identical models. In the former model, we show that it is a 2-approximation algorithm, and in the latter model, we show that it is a $\rho(k)$-approximation algorithm, where

$$\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k} = \frac{(1+1/k)^k}{(1+1/k)^k - 1} .$$

3.1 The greedy algorithm

The greedy strategy for a single machine is as follows. At each time step t (starting at $t = 0$), the algorithm schedules the job instance that finishes first among all jobs that can be scheduled at t or later. Note that the greedy algorithm does not take into consideration the deadlines of the jobs, except for determining whether jobs are eligible for scheduling. The greedy algorithm for multiple machines just executes the greedy algorithm (for a single machine) machine by machine.

We now define the procedure $\mathsf{NEXT}(t, j, \mathcal{J})$. The procedure determines the job instance $J_{i,j}(t')$, $t' \geq t$, such that $t' + \ell_{i,j}$ is the earliest among all instances of jobs in $\mathcal{J}$ that start at time t or later on machine M_j. If no such interval exists, the procedure returns *null*, otherwise the procedure returns $J_{i,j}(t')$.

Algorithm 1-$\mathsf{GREEDY}(j, \mathcal{J})$ finds a feasible schedule on machine M_j among the jobs in $\mathcal{J}$. by calling Procedure NEXT repeatedly.

1. The first call is for $J_{i_1,j}(t_1) = \mathsf{NEXT}(0, j, \mathcal{J})$.

2. Assume the algorithm has already computed $J_{i_1,j}(t_1), \ldots, J_{i_h,j}(t_h)$. Let the current time be $t = t_h + \ell_{i_h,j}$ and let the current set of jobs be $\mathcal{J} := \mathcal{J} \setminus \{J_{i_1,j}, \ldots, J_{i_h,j}\}$.

3. The algorithm calls $\mathsf{NEXT}(t, j, \mathcal{J})$ that returns either $J_{i_{h+1},j}(t_{h+1})$ or *null*.

4. The algorithm terminates in round $r + 1$ when procedure NEXT returns *null*. It returns the set $\{J_{i_1,j}(t_1), \ldots, J_{i_r,j}(t_r)\}$.

Algorithm k-GREEDY($\mathcal{J}$) finds k schedules such that a job appears at most once in the schedules. It calls Algorithm 1-GREEDY machine by machine, each time updating the set $\mathcal{J}$ of jobs to be scheduled. Assume that the output of 1-GREEDY($j, \mathcal{J}$) in the first $i-1$ calls is $G_1, \ldots, G_{i-1}$, where G_j is a feasible schedule on machine M_j, for $1 \leq j \leq i-1$. Then, the algorithm calls 1-GREEDY($i, \mathcal{J} \setminus \cup_{j=1,\ldots,i-1} G_j$) to get schedule G_i.

The following property of Algorithm 1-GREEDY is used in the analysis of the approximation factors of our algorithms.

Proposition 3.1 *Let the set of jobs found by* 1-GREEDY($j, \mathcal{J}$) *for a job system $\mathcal{J}$ be G. Let H be any feasible schedule on machine M_j among the jobs in $\mathcal{J} \setminus G$. Then, $|H| \leq |G|$.*

Proof: For each interval (job instance) in H there exists an interval in G that overlaps with it and terminates earlier. Otherwise, 1-GREEDY would have chosen this interval. The proposition follows from the feasibility of H, since at most one interval in H can overlap with the end point of any interval in G. □

3.2 Unrelated machines

Based on Proposition 3.1, the following theorem states the performance of the k-GREEDY algorithm in the unweighted jobs and unrelated machines model.

Theorem 3.2 *Algorithm k-GREEDY achieves a 2 approximation factor in the unweighted jobs and unrelated machines model.*

Proof: Let $G(k) = G_1 \cup \cdots \cup G_k$ be the output of k-GREEDY and let $OPT(k) = O_1 \cup \cdots \cup O_k$ be the sets of intervals scheduled on the k machines by an optimal solution OPT. (We note that these sets will be considered as jobs and job instances interchangeably.) Let $H_j = O_j \setminus G(k)$ be the set of all the jobs scheduled by OPT on machine M_j that k-GREEDY did not schedule on any machine and let $H = H_1 \cup \cdots \cup H_k$. Let $OG = OPT(k) \cap G(k)$ be the set of jobs taken by both k-GREEDY and OPT. It follows that $OPT(k) = OG \cup H$.

Proposition 3.1 implies that $|H_j| \leq |G_j|$. This is true since H_j is a feasible schedule on machine M_j among the jobs that were not picked by k-GREEDY while constructing the schedule for machine M_j. Since the sets H_j are mutually disjoint and the same holds for the sets G_j, $|H| \leq |G(k)|$. Since $|OG| \leq |G(k)|$, we get that $|OPT(k)| \leq 2|G(k)|$ and the theorem follows. □

3.3 Identical machines

In this section we analyze the k-GREEDY algorithm for the unweighted jobs and identical machines model. We show that the approximation factor in this case is

$$\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k} = \frac{(1+1/k)^k}{(1+1/k)^k - 1} .$$

For $k = 1, 2$ we get $\rho(1) = 2$ and $\rho(2) = 9/5$, and for $k \longrightarrow \infty$ we have $\rho(k) \longrightarrow e/(e-1) \approx 1.58198$.

The analysis below is quite general and just uses the facts that the algorithm is applied sequentially machine by machine, and that the machines are identical. Let $OPT(k)$ be an optimal schedule for k identical machines. Let $\mathcal{A}$ be any algorithm for one machine. Define by $\alpha_{\mathcal{A}}(k)$ (or by $\alpha(k)$ when $\mathcal{A}$ is known) the approximation factor of $\mathcal{A}$ compared with $OPT(k)$. Note that the comparison is done between an algorithm that uses one machine and an optimal schedule that uses k machines. Let $\mathcal{A}(k)$ be the algorithm that applies algorithm $\mathcal{A}$, machine by machine, k times. In the next theorem we bound the performance of $\mathcal{A}(k)$ using $\alpha(k)$.

Theorem 3.3 *Algorithm $\mathcal{A}(k)$ achieves an $\frac{\alpha(k)^k}{\alpha(k)^k - (\alpha(k)-1)^k}$ approximation factor for k identical machines.*

Proof: Let A_i be the set of jobs chosen by $\mathcal{A}(k)$ for the i-th machine. Suppose that the algorithm has already chosen the sets of jobs $A_1, \ldots, A_{i-1}$. Consider the schedule given by removing from $OPT(k)$ all the jobs in $A_1, \ldots, A_{i-1}$ that were also chosen by the optimal solution. Clearly, this is still a feasible schedule of cardinality at least $|OPT(k)| - \sum_{j=1}^{i-1} |A_j|$. Therefore, by the definition of $\alpha(k)$, the set A_i satisfies $|A_i| \geq (1/\alpha(k))(|OPT(k)| - \sum_{j=1}^{i-1} |A_j|)$. Rearranging the terms gives us the equation,

$$\sum_{j=1}^{i} |A_j| \geq \frac{|OPT(k)|}{\alpha(k)} + \frac{\alpha(k)-1}{\alpha(k)} \sum_{j=1}^{i-1} |A_j| . \tag{1}$$

We prove by induction on i that

$$\sum_{j=1}^{i} |A_j| \geq \frac{\alpha(k)^i - (\alpha(k)-1)^i}{\alpha(k)^i} |OPT(k)| .$$

When $i = 1$, by definition, $|A_1| \geq \frac{|OPT(k)|}{\alpha(k)}$. Assume the claim holds for $i-1$. Applying the induction hypothesis to Equation (1) we get,

$$\sum_{j=1}^{i} |A_j| \geq \frac{|OPT(k)|}{\alpha(k)} + \frac{\alpha(k)-1}{\alpha(k)} \cdot \frac{\alpha(k)^{i-1} - (\alpha(k)-1)^{i-1}}{\alpha(k)^{i-1}} |OPT(k)| .$$

Rearranging terms yields the inductive claim. Setting $i = k$ proves the theorem, namely,

$$\sum_{j=1}^{k} |A_j| \geq \frac{\alpha(k)^k - (\alpha(k)-1)^k}{\alpha(k)^k} |OPT(k)| .$$

□

We now apply the above theorem to algorithm k-GREEDY. We compute the value of $\alpha(k)$ for algorithm 1-GREEDY, and observe that algorithm k-GREEDY indeed applies algorithm 1-GREEDY k times, as assumed by Theorem 3.3.

Theorem 3.4 *The approximation factor of k-GREEDY is $\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k}$, in the unweighted jobs and identical machines model.*

Proof: Recall that algorithm 1-GREEDY scans all the intervals ordered by their end points and picks the first possible interval belonging to a job that was not picked before. Suppose this greedy strategy picks set G, and consider the schedule of $H = OPT(k) - G$ on machines numbered $1, \ldots, k$. Similar to the arguments of Proposition 3.1, in each of the machines, if a particular job of H was not chosen, then there must be a job in progress in G. Also this job must finish before the particular job in H finishes. Thus, the number of jobs in H executed on any single machine by the optimal schedule has to be at most $|G|$. Since the jobs executed by the optimal schedule on different machines are disjoint, we get $|H| \leq k|G|$. Consequently, $|OPT(k)| \leq (k+1)|G|$ and $\alpha(k) = k+1$. The theorem follows by setting this value for $\alpha(k)$ in Theorem 3.3. □

3.4 Tight bounds for GREEDY

In this subsection we construct an instance for which our bounds in the unweighted model for algorithm GREEDY are tight. We first show that for one machine (where the unrelated and identical models coincide) the 2-approximation is tight. Next, we generalize this construction for the unrelated model, and prove the tight bound of 2 for $k > 1$ machines. Finally, we generalize the construction for one machine to $k > 1$ identical machines and prove the tight bound of $\rho(k)$.

Recall that in the unweighted model each job is characterized by a triplet (r_i, d_i, ℓ_i) in the identical machines model and by a triplet (r_i, d_i, L_i), where $L_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$, in the unrelated machines model.

3.4.1 A single machine

For a single machine the system contains two jobs: $G_1 = (0, 3, 1)$ and $H_1 = (0, 2, 2)$. Algorithm 1-GREEDY schedules the instance $G_1(0)$ of job G_1 and cannot schedule any instance of H_1. An optimal solution schedules the instances $H_1(0)$ and $G_1(2)$. Clearly, the ratio is 2. We could repeat this pattern on the time axis to obtain this ratio for any number of jobs.

This construction demonstrates the limitation of the approach of Algorithm 1-GREEDY. This approach ignores the deadlines and therefore does not capitalize on the urgency in scheduling job H_1 in order not to miss its deadline. We generalize this idea further for k machines.

3.4.2 Unrelated machines

For $k \geq 1$ machines the job system contains $2k$ jobs: $G_1, \ldots, G_k$ and $H_1, \ldots, H_k$. The release time of all jobs is 0. The deadline of all the G-type jobs is 3 and the deadline of all the H-type jobs is 2. The length of job G_i on machine M_i is 1 and it is 4 on all other machines. The length of job H_i on machine M_i is 2 and it is 3 on all other machines.

Note that only jobs G_i and H_i can be scheduled on machine M_i, since all other jobs are too long to meet their deadline. Hence, Algorithm k-GREEDY considers only these two jobs while constructing the schedule for machine M_i. As a result, k-GREEDY selects the instance $G_i(0)$ of job G_i to be scheduled on machine M_i and cannot schedule any of the H-type jobs. On the other hand, an optimal solution schedules the instances $H_i(0)$ and $G_i(2)$ on machine M_i.

Overall, Algorithm k-GREEDY schedules k jobs while an optimal algorithm schedules all $2k$ jobs. This yields a tight approximation factor of 2 in the unweighted jobs and unrelated machines model.

3.4.3 Identical machines

We define job systems $\mathcal{J}(k)$ for any given $k \geq 1$. We show that on $\mathcal{J}(k)$ the performance of k-GREEDY($\mathcal{J}(k)$) is no more than $(1/\rho(k)) \cdot \mathsf{OPT}(\mathcal{J}(k))$. The $\mathcal{J}(1)$ system is the one defined in Subsection 3.4.1. The $\mathcal{J}(2)$ job system contains 18 jobs: 6 jobs of type $G_1 = (0, d_1, \ell)$, 4 jobs of type $G_2 = (0, d_2, \ell + 1)$, and 8 jobs of type $H = (0, d, \ell + 2)$. If we set $\ell = 10$, $d_1 = 100$, $d_2 = 70$, and $d = 48$, we force Algorithm 2-GREEDY to make the following selections:

- On the first machine, 2-GREEDY schedules all the 6 jobs of type G_1. This is true since these jobs are of length less than the lengths of the jobs of type G_2 and the jobs of type H. The last G_1-type interval terminates at time 60. Hence, there is no room for a G_2-type (H-type) interval, the deadline of which is 70 (48), and the length of which is 11 (12).
- On the second machine, 2-GREEDY schedules all the 4 jobs of type G_2 since they are shorter than the jobs of type H. The last G_2-type job terminates at time 44 which leaves no room for another job of type H.

Overall, 2-GREEDY schedules only 10 jobs. We show now an optimal solution that schedules all 18 jobs. It schedules 9 jobs on each machine as follows:

$$H(0), H(12), H(24), H(36), G_2(48), G_2(59), G_1(70), G_1(80), G_1(90) \ .$$

Note that all the instances terminate before their deadlines. As a result we get a ratio $\rho(2) = 9/5$.

We are ready to define $\mathcal{J}(k)$ for any $k \geq 1$. The job system contains $k(k+1)^k$ jobs. Algorithm k-GREEDY is able to schedule only $k(k+1)^k - k^{k+1}$ out of them and there exists an optimal solution that schedules all of them. As a result we get the ratio

$$\frac{k(k+1)^k}{k(k+1)^k - k^{k+1}} = \frac{(1+1/k)^k}{(1+1/k)^k - 1} = \rho(k) \ .$$

The $\mathcal{J}(k)$ system is composed of $k+1$ types of jobs: $G_1, G_2, \ldots, G_k, H$. There are $(k+1)^{k-i} \cdot k^i$ jobs $(0, d_i, \ell + i - 1)$ in G_i and k^{k+1} jobs $(0, d, \ell + k)$ in H. By setting ℓ as a large enough number and then fixing $d_1, \ldots, d_k$ and d, we force Algorithm k-GREEDY to select for machine i all the jobs of type G_i but no other jobs. Thus k-GREEDY does not schedule any of the H-type jobs. On the other hand, an optimal solution is able to construct the same schedule for all the k machines. It starts by scheduling $1/k$ of the H-type jobs with their first possible instance. Then, it schedules in turn $1/k$ of the jobs from $G_k, G_{k-1}, \ldots, G_1$ in this order. The values of $d, d_k, \ldots, d_1$ allow for such a schedule.

We omit the details of how to set ℓ and how to determine the deadlines. We just remark that to validate the optimal schedule, we get lower bounds for the deadlines and to force the k-GREEDY schedule we get upper bounds for the deadlines. If ℓ is large enough, then these upper bounds are larger than the lower bounds.

For $k = 3$, one could check the following values: $\ell = 111$, $d = 27 \cdot (111 + 3)$, $d_3 = 9 \cdot (111 + 2) + 27 \cdot (111 + 3) = 36 \cdot 111 + 99$, $d_2 = 12 \cdot (111 + 1) + 9 \cdot (111 + 2) + 27 \cdot (111 + 3) = 48 \cdot 111 + 111$, and $d_1 = 16 \cdot 111 + 12 \cdot (111 + 1) + 9 \cdot (111 + 2) + 27 \cdot (111 + 3) = 64 \cdot 111 + 111$.

4 Weighted jobs

In this section we present approximation algorithms for weighted jobs. We first present algorithms for a single machine and for unrelated machines that are based on rounding a linear programming relaxation of the problem. Then, we re-apply the analysis of Theorem 3.3 to get better approximation factors for the identical machines model. We conclude with a combinatorial algorithm for unrelated machines which is efficient and easy to implement. However, it achieves a weaker approximation guarantee.

4.1 Approximation via linear programming

In this subsection we describe a linear programming based approximation algorithm. We first describe the algorithm for the case of a single machine, and then generalize it to the case of multiple machines. Our linear programming formulation is based on discretizing time. Suppose that the time axis is divided into N time slots. The complexity of our algorithms depends on N.

However, we assume for now that N is part of the input, and that the discretization of time is fine enough so as to represent any feasible schedule (up to small shifts). Later, we show how to get rid of these assumptions at the expense of a slight increase in the approximation factor.

The linear program relaxes the scheduling problem in the following way. A fractional feasible solution is one which distributes the processing of a job between the job instances or intervals belonging to it with the restriction that at any given point of time t, the sum of the fractions assigned to all the intervals at t (belonging to all jobs) does not exceed 1. To this end, for each job $J_i \in \mathcal{J}$, define a variable x_{it} for each interval $[t, t+\ell_i)$ belonging to it, i.e., for which $t \geq r_i$ and $t + \ell_i \leq d_i$. It would be convenient to assume that $x_{it} = 0$ for any other value of t between 1 and N. The linear program is as follows.

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{t=r_i}^{d_i-\ell_i} w_i \cdot x_{it}$$

subject to:

$$\begin{array}{lrcl}
\text{For each time slot } t,\ 1 \leq t \leq N: & \displaystyle\sum_{i=1}^{n} \sum_{t'=t-\ell_i+1}^{t} x_{it'} & \leq & 1 \\
\text{For each job } i,\ 1 \leq i \leq n: & \displaystyle\sum_{t=r_i}^{d_i-\ell_i} x_{it} & \leq & 1 \\
\text{For all } i \text{ and } t: & 0 \leq x_{it} & \leq & 1
\end{array}$$

It is easy to see that any feasible schedule defines a feasible integral solution to the linear program and vice versa. Therefore, the value of an optimal (fractional) solution to the linear program is an upper bound on the value of an optimal integral solution.

We compute an optimal solution to the linear program and denote the value of variable x_{it} in this solution by q_{it}. Denote the value of the objective function in an optimal solution by OPT. We now show how to round an optimal solution to the linear program to an integral solution.

To show how the linear program is used we define a coloring of intervals. The collection of all intervals belonging to a set of jobs $\mathcal{J}$ can be regarded as an interval representation of an interval graph $\mathcal{I}$. We define a set of intervals in $\mathcal{I}$ to be *independent* if: (i) No two intervals in the set overlap; (ii) No two intervals in the set belong to the same job. (Note that this definition is more restrictive than the regular independence relation in interval graphs.) Clearly, an independent set of intervals defines a feasible schedule. The weight of an independent set P, $w(P)$, is defined to be the sum of the weights of the jobs to which the intervals belong.

Our goal is to color intervals in $\mathcal{I}$ such that each color class induces an independent set. We note that not all intervals are required to be colored and that an interval may receive more than one color. Suppose that a collection of color classes (independent sets) $P_1, \ldots, P_m$ with non-negative coefficients $\alpha_1, \ldots, \alpha_m$ satisfies (i) $\sum_{i=1}^{m} \alpha_i \leq 2$, and, (ii) $\sum_{i=1}^{m} w(P_i) \cdot \alpha_i \geq \mathsf{OPT}$. By convexity, there exists a color class P_i, $1 \leq i \leq m$, for which $w(P_i) \geq \mathsf{OPT}/2$. This color class is defined to be our approximate solution, and the approximation factor is 2. It remains to show how to obtain the desired coloring.

We now take a short detour and define the *group constrained interval coloring problem.* Let $Q = \langle Q_1, \ldots, Q_p \rangle$ be an interval representation in which the maximum number of mutually overlapping intervals is t_1. Suppose that the intervals are partitioned into disjoint groups $g_1, \ldots, g_r$, where a group contains at most t_2 intervals. A legal group constrained coloring of the intervals in Q is a coloring in which: (i) Overlapping intervals are not allowed to get the same color; (ii) Intervals belonging to the same group are not allowed to get the same color.

Theorem 4.1 *There exists a legal group constrained coloring of the intervals in Q that uses at most $t_1 + t_2 - 1$ colors.*

Proof: We use a greedy algorithm to obtain a legal coloring using at most $t_1 + t_2 - 1$ colors. Sort the intervals in Q by their left endpoint and color the intervals from left to right with respect to this ordering. When an interval is considered by the algorithm it is colored by any one of the free colors available at that time. We show by induction that when the algorithm considers an interval, there is always a free color.

This is true initially. When the algorithm considers interval Q_i, the colors that cannot be used for Q_i are occupied by either intervals that overlap with Q_i, or by intervals that belong to the same group as Q_i. Since we are considering the intervals sorted by their left endpoint, all intervals overlapping with Q_i also overlap with each other, and hence there are at most $t_1 - 1$ such intervals. There can be at most $t_2 - 1$ intervals that belong to the same group as Q_i. Since the number of available colors is $t_1 + t_2 - 1$, there is always a free color. □

We are now back to the problem of coloring the intervals in $\mathcal{I}$. Let $N' = N^2 \cdot n^2$. We can round each fraction q_{it} in the optimal solution to the closest fraction of the form a/N', where $1 \leq a \leq N'$. This incurs a negligible error (of at most $1/(Nn)$ factor) in the value of the objective function. We now generate an interval graph $\mathcal{I}'$ from $\mathcal{I}$ by replacing each interval $J_i(t) \in \mathcal{I}$ by $q_{it} \cdot N'$ "parallel" intervals. Define a group constrained coloring problem on $\mathcal{I}'$, where group g_i, $1 \leq i \leq n$, contains all instances of job J_i. Note that in $\mathcal{I}'$, the maximum number of mutually overlapping intervals is bounded by N', and the maximum number of intervals belonging to a group is also N'.

By Theorem 4.1, there exists a group constrained coloring of $\mathcal{I}'$ that uses at most $2N' - 1$ colors. Attach a coefficient of $1/N'$ to each color class. Clearly, the sum of the coefficients is less than 2. Also, by our construction, the sum of the weights of the intervals in all the color classes, multiplied by the coefficient $1/N'$, is OPT. We conclude,

Theorem 4.2 *The approximation factor of the algorithm that rounds an optimal fractional solution* *is* 2.

We note that the technique of rounding a fractional solution by decomposing it into a convex combination of integral solutions was also used by Albers *et al.* [2]

4.1.1 Strongly polynomial bounds

The difficulty with the linear programming formulation and the rounding algorithm is that the complexity of the algorithm depends on N, the number of time slots. We now show how we choose N to be a polynomial in the number of jobs, n, at the expense of losing a bit in the approximation factor.

First, we note that in case the release times, deadlines, and processing times are integral, we may assume without loss of generality that each job is scheduled at an integral point of time. If, in addition, they are restricted to integers of polynomial size, then the number of variables and constraints is bounded by polynomial.

We now turn our attention to the case of arbitrary inputs. Let $p(n)$ be a $\Theta(n^2)$ polynomial. Partition the jobs in $\mathcal{J}$ into two classes:

- Big slack jobs: $J_i \in \mathcal{J}$ for which $d_i - r_i > p(n) \cdot \ell_i$.
- Small slack jobs: $J_i \in \mathcal{J}$ for which $d_i - r_i \leq p(n) \cdot \ell_i$.

We obtain a fractional solution separately for the big slack jobs and small slack jobs. We first explain how to obtain a fractional solution for the big slack jobs. For each big slack job $J_i \in \mathcal{J}$, find $p(n)$ non-overlapping job instances and assign a value of $1/p(n)$ to each such interval. Note that this many non-overlapping intervals can be found since $d_i - r_i$ is large enough. We claim that this assignment can be ignored when computing the solution (via LP) for the small slack jobs. This is true because at any point of time t, the sum of the fractions assigned to intervals at t belonging to big slack jobs can be at most $n/p(n)$, and thus their effect on any fractional solution is negligible. (In the worst case, scale down all fractions corresponding to small slack jobs by a factor of $(1 - n/p(n))$.) Nevertheless, a big slack job contributes all of its weight to the fractional objective function.

We now restrict our attention to the set of small slack jobs and explain how to compute a fractional solution for them. For this we solve an LP. To bound the number of variables and constraints in the LP we partition time into at most $n \cdot (p(n) + 1)$ time slots. Instead of having a variable for each job instance we consider at most $n^2 \cdot (p(n)+1)$ variables, where for each job $J_i \in \mathcal{J}$, there are at most $n \cdot (p(n) + 1)$ variables, and the j-th variable "represents" all the job instances of J_i that start in the j-th time slot. Similarly, we consider at most $n \cdot (p(n) + 1)$ constraints, where the j-th constraint "covers" the j-th time slot. For each small slack job $J_i \in \mathcal{J}$, define

$p(n)+1$ "dividers" along the time axis at points $r_i + j\frac{d_i - r_i}{p(n)}$, for $j = 0, \ldots, p(n)$. After defining all the $n \cdot (p(n)+1)$ dividers, the time slots are determined by the adjacent dividers. The main observation is that for each small slack job J_i, no interval can be fully contained in a time slot, i.e., between two consecutive dividers.

The LP formulation for the modified variables and constraints is slightly different from the original formulation. To see why, consider a feasible schedule. As mentioned above, a job instance cannot be fully contained in a time slot t. However, the schedule we are considering may consist of two instances of jobs such that one terminates within time slot t and the other starts within t. If we keep the constraints that stipulate that the sum of the variables corresponding to intervals that intersect a time slot is bounded by 1, then we would not be able to represent such a schedule in our formulation. To overcome this problem, we relax the linear program, and allow that at every time slot t, the sum of the fractions assigned to the intervals that intersect t can be at most 2. The relaxed linear program is the following.

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{t=r_i}^{d_i-\ell_i} w_i \cdot x_{it}$$

subject to:

$$\begin{array}{lrcl}
\text{For each time slot } t: & \displaystyle\sum_{i=1}^{n} \sum_{t'=t-\ell_i+1}^{t} x_{it'} & \leq & 2 \\
\text{For each job } i,\ 1 \leq i \leq n: & \displaystyle\sum_{t=r_i}^{d_i-\ell_i} x_{it} & \leq & 1 \\
\text{For all } i \text{ and } t: & 0 \leq x_{it} & \leq & 1
\end{array}$$

It is easy to see that our relaxation guarantees that the value of the objective function in the above linear program is at least as big as the value of an optimal schedule. We round an optimal fractional solution in the same way as in the previous section. Since we relaxed our constraints, we note that when we run the group constrained interval coloring algorithm, the number of mutually overlapping intervals can be at most twice the number of intervals in each group. Therefore, when we generate the color classes $P_1, \ldots, P_m$ with coefficients $\alpha_1, \ldots, \alpha_m$, we can only guarantee that: (i) $\sum_{i=1}^{m} \alpha_i < 3$; and, (ii) $\sum_{i=1}^{m} w(P_i) \cdot \alpha_i = \mathsf{OPT}$, yielding an approximation factor of 3. We conclude,

Theorem 4.3 *The approximation factor of the strongly polynomial algorithm that rounds a fractional solution is* 3.

4.1.2 Unrelated machines

In this section we consider the case of k unrelated machines. We first present a linear programming formulation. For clarity, we give the LP formulation for polynomially bounded integral inputs. However, the construction given above that achieves a strongly polynomial algorithm for arbitrary inputs can be applied here as well. Assume that there are N time slots. For each job $J_i \in \mathcal{J}$, machine j, define a variable x_{itj} for each instance $[t, t+\ell_{i,j}]$ of J_i.

$$\text{maximize} \quad \sum_{j=1}^{k}\sum_{i=1}^{n}\sum_{t=r_i}^{d_i-\ell_{i,j}} w_i \cdot x_{itj}$$

subject to:

$$\begin{array}{lrcl}
\text{For each time slot } t \text{ and machine } j: & \displaystyle\sum_{i=1}^{n}\sum_{t'=t-\ell_{i,j}+1}^{t} x_{it'j} & \leq & 1 \\
\text{For each job } i,\ 1 \leq i \leq n: & \displaystyle\sum_{j=1}^{k}\sum_{t=r_i}^{d_i-\ell_{i,j}} x_{itj} & \leq & 1 \\
\text{For all } i,\ j, \text{ and } t: & 0 \leq x_{itj} & \leq & 1
\end{array}$$

The algorithm rounds the fractional solution machine by machine. Let $S = \{S_1, \ldots, S_k\}$ denote the rounded solution. When rounding machine i, we first discard from its fractional solution all intervals belonging to jobs chosen to $S_1, \ldots, S_{i-1}$. Let c denote the approximation factor that can be achieved when rounding a single machine. Namely, $c = 2$ for integral polynomial size inputs and $c = 3$ for arbitrary inputs.

Theorem 4.4 *The approximation factor of the algorithm that rounds a k-machine solution is $(c+1)$.*

Proof: Let F_i, $1 \leq i \leq k$, denote the fractional solution of machine i, and let $w(F_i)$ denote its value. Denote by F'_i the fractional solution of machine i after discarding all intervals belonging to jobs chosen to $S_1, \ldots, S_{i-1}$.

We know that for all i, $1 \leq i \leq k$,

$$w(S_i) \geq \frac{1}{c} \cdot w(F'_i)$$

Adding up all the inequalities, since the sets S_i (F'_i) are mutually disjoint, we get that,

$$w(S) \geq \frac{1}{c} \cdot \sum_{i=1}^{k} w(F'_i)$$

Recall that for each job i, the sum of the values of the fractional solution assigned to the intervals belonging to it in all the machines does not exceed 1. Therefore,

$$\sum_{i=1}^{k} w(F_i') \geq \sum_{i=1}^{k} w(F_i) - w(S).$$

Yielding that

$$w(S) \geq \frac{\sum_{i=1}^{k} w(F_i)}{c+1}.$$

□

4.1.3 Identical machines

In this subsection we apply Theorem 3.3 for the case of weighted jobs and identical machines. We distinguish between the cases of polynomially bounded integral input and arbitrary input.

Theorem 4.5 *There exists an algorithm for the weighted jobs and identical machines case that achieves an approximation factor of* $\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k}$*, for polynomially bounded integral input, and* $\rho'(k) = \frac{(2k+1)^k}{(2k+1)^k-(2k)^k}$*, for arbitrary input.*

Proof: As shown above, a linear program can be formulated such that the value of its optimal solution is at least as big as the value of an optimal schedule. Let N' be chosen in the same way as in the discussion preceding Theorem 4.2. We claim that using our rounding scheme, this feasible solution defines an interval graph that can be colored by $(k+1)N'-1$ colors for integral polynomial size inputs and by $(2k+1)N'-1$ colors for arbitrary inputs.

Consider first the case of integral polynomial size input. In the interval graph that is induced by the solution of the LP, there are at most N' intervals (that correspond to the same job) in the same group, and at most kN' intervals mutually overlap at any point of time. Applying our group constrained interval coloring, we get a valid coloring with $(k+1)N'-1$ colors. Similarly, for arbitrary inputs, in the interval graph which is induced by the solution of the LP, there are at most N' intervals (that correspond to the same job) in the same group, and at most $2kN'$ intervals mutually overlap. Applying our group constrained interval coloring, we get a valid coloring with $(2k+1)N'-1$ colors.

This implies that $\alpha(k) = k+1$ for integral polynomial size input and $\alpha(k) = 2k+1$ for arbitrary input. In other words, this is the approximation factor that can be achieved with a single machine when compared to an optimal algorithm that uses k identical machines. Setting these values of $\alpha(k)$ in our paradigm for transforming an algorithm for a single machine to an algorithm for k

identical machines, yields the claimed approximation factors. □

Remark: Note that as k tends to infinity, the approximation factor is $\frac{e}{e-1} \approx 1.58192$ for both unweighted jobs and for weighted jobs with integral polynomial size inputs. For arbitrary input, the approximation factor is $\frac{\sqrt{e}}{\sqrt{e}-1} \approx 2.54149$. Setting $k = 1$ we get that these bounds coincide with the bounds for a single machine. For every $k > 2$ and for both cases these bounds improve upon the bounds for unrelated machines (of 3 and 4).

4.2 A combinatorial algorithm

In this section we present a combinatorial algorithm for the weighted machines model. We first present an algorithm for the single-machine version and then we show how to extend it to the case where there are $k > 1$ machines, even in the unrelated machines model.

4.2.1 A single machine

The algorithm is inspired by on-line call admission algorithms (see [12, 7]). We scan the jobs instances (or intervals) one by one. For each job instance, we either accept it, or reject it. We note that rejection is an irrevocable decision, where as acceptance can be temporary, i.e., an accepted job may still be rejected at a later point of time. We remark that in the case of non-preemptive on-line call admission, a constant competitive factor cannot be achieved by such an algorithm. The reason is that due to the on-line nature of the problem jobs must be considered in the order of their release time. Our algorithm has the freedom to order the jobs in a different way, yielding a constant approximation factor.

We now outline the algorithm. All feasible intervals of all jobs are scanned from left to right (on the time axis) sorted by their end points. The algorithm maintains a set $\mathcal{A}$ of currently accepted intervals. When a new interval, I, is considered according to the sorted order, it is immediately rejected if it belongs to a job that already has an instance in $\mathcal{A}$, and immediately accepted if it does not overlap with any other interval in $\mathcal{A}$. In case of acceptance, interval I is added to $\mathcal{A}$. If I overlaps with one or more intervals in $\mathcal{A}$, it is accepted only if its weight is more than β (to be determined later) times the sum of the weights of all overlapping intervals. In this case, we say that I "preempts" these overlapping intervals. We add I to $\mathcal{A}$ and discard all the overlapping intervals from $\mathcal{A}$. The process ends when there are no more intervals to scan.

A more formal description of our algorithm, called Algorithm ADMISSION is given in Figure 2.

The main difficulty in implementing the above algorithm is in scanning an infinite number of intervals. After proving the performance of the algorithm we show how to overcome this difficulty.

Informally, we say that an interval I "caused" the rejection or preemption of another interval J,

Algorithm ADMISSION:

1. Let $\mathcal{A}$ be the set of accepted job instances.
   Initially, $\mathcal{A} = \emptyset$.
2. Let $\mathcal{I}$ be the set of the yet unprocessed job instances.
   Initially, $\mathcal{I}$ is the set of all feasible job instances.
3. While $\mathcal{I}$ is not empty repeat the following procedure:
   Let $I \in J_i$ be the job instance that terminates earliest among all instances in $\mathcal{I}$ and let w be its weight.
   Let W be the sum of the weights of all instances $I_1, \ldots, I_h$ in $\mathcal{A}$ that overlap I.
   (a) $\mathcal{I} := \mathcal{I} \setminus \{I\}$.
   (b) If $J_i \cap \mathcal{A} \neq \emptyset$ then **reject** I.
   (c) Else if $W = 0$ then **accept** I;
   $\mathcal{A} := \mathcal{A} \cup \{I\}$.
   (d) Else if $\frac{w}{W} > \beta$ then **accept** I and **preempt** $I_1, \ldots, I_h$;
   $\mathcal{A} := \mathcal{A} \cup \{I\} \setminus \{I_1, \ldots, I_h\}$.
   (e) Otherwise (i.e., $\frac{w}{W} \leq \beta$) then **reject** I.

end ADMISSION

Figure 2: Algorithm ADMISSION

if either interval I directly rejected or preempted interval J, or if it preempted another interval that caused the rejection or preemption of interval J. (Note that this relation is defined recursively, as an interval I may preempt another interval, that preempted other intervals, which in turn rejected other intervals. In this case interval I caused the rejection or preemption of all these intervals.) Fix an interval I that was accepted by the algorithm, and consider all the intervals chosen by the optimal solution, the rejection or preemption of which was caused by interval I. We prove that the total weight of these intervals is at most $f(\beta)$ times the weight of the accepted interval I, for some function f. Optimizing β, we get the $3 + 2\sqrt{2} \approx 5.828$ bound.

Theorem 4.6 *The approximation factor of Algorithm* ADMISSION *is* $3 + 2\sqrt{2}$.

Proof: Let $\mathcal{O}$ be the set of intervals chosen by an optimal algorithm OPT. Let the set of intervals accepted by Algorithm ADMISSION be denoted by $\mathcal{A}$. For each interval $I \in \mathcal{A}$ we define a set $R(I)$ of all the intervals in $\mathcal{O}$ that are "accounted for" by I. This set consists of I in case $I \in \mathcal{O}$, and of all the intervals in $\mathcal{O}$ the rejection or preemption of which was caused by I. More formally:

- Assume I is accepted by rule 3(c). Then, the set $R(I)$ is initialized to be I in case $I \in \mathcal{O}$ and the empty set $\emptyset$, otherwise.

- Assume I is accepted by rule 3(d). Then $R(I)$ is initialized to contain all those intervals from $\mathcal{O}$ that were directly preempted by I and the union of the sets $R(I')$ of all the intervals I' that were preempted by I. In addition, $R(I)$ contains I in case $I \in \mathcal{O}$.

- Assume $J \in \mathcal{O}$ is rejected by rule 3(b). Let $I \in \mathcal{A}$ be the interval that caused the rejection of J. Note that both I and J belong to the same job. In this case add J to $R(I)$.

- Assume $J \in \mathcal{O}$ was rejected by rule 3(e) and let $I_1, \ldots, I_h$ be the intervals in $\mathcal{A}$ that overlapped with J at the time of rejection. Let w be the weight of J and let w_j be the weight of I_j for $1 \le j \le h$. We view J as h imaginary intervals $J_1, \ldots, J_k$, where the weight of J_j is $\frac{w_j \cdot w}{\sum_{j=1}^{h} w_j}$ for $1 \le j \le h$. Set $R(I_j) := R(I_j) \cup \{J_j\}$. Note that due to the rejection rule it follows that the weight of J_j is no more than β times the weight of I_j.

It is not hard to see that each interval from $\mathcal{O}$, or a portion of it if we use rule 3(e), belongs exactly to one set $R(I)$ for some $I \in \mathcal{A}$. Thus, the union of all sets $R(I)$ for $I \in \mathcal{A}$ covers $\mathcal{O}$.

We now fix an interval $I \in \mathcal{A}$. Let w be the weight of I and let W be the sum of weights of all intervals in $R(I)$. Define $\rho = \frac{W}{w}$. Our goal is to bound ρ from above.

Interval I may directly reject at most one interval from $\mathcal{O}$. Let w_r be the weight of (the portion of) the interval $I_r \in \mathcal{O} \cap R(I)$ that was directly rejected by I, if such exists. Otherwise, let $w_r = 0$. Observe that $w_r \le \beta w$, since otherwise I_r would not have been rejected. Let $I' \in \mathcal{O}$ be the interval that belongs to the same job as the one to which I belongs (it maybe I itself), if such exists. By definition, the weight of I' is w. Let $W' = W - w - w_r$ be the sum of the weights of the rest of the intervals in $R(I)$. Define $\alpha = \frac{W'}{w}$. It follow that $\rho = \alpha + \beta + 1$.

We now assume inductively that the ρ bound is valid for intervals with earlier end point than the end point of I. Since the overall weight of the jobs that I directly preempted is at most w/β, we get that $\frac{w}{\beta} \cdot \rho = \alpha \cdot w$. This implies that $\frac{\alpha+\beta+1}{\beta} = \alpha$. Or equivalently, $\alpha = \frac{\beta+1}{\beta-1} = 1 + \frac{2}{\beta-1}$. Therefore, $\rho = 2 + \beta + \frac{2}{\beta-1}$. This equation is minimized for $\beta = 1 + \sqrt{2}$ which implies that $\rho = 3 + 2\sqrt{2}$. Finally, since the ρ bound holds for all the intervals in $\mathcal{A}$ and since the union of all $R(I)$ sets covers all the interval taken by OPT, we get that the value of OPT is at most ρ times the value of $\mathcal{A}$. Hence, the approximation factor is $3 + 2\sqrt{2}$. $\square$

Implementation: Observe that Step (3) of the algorithm has to be invoked only when there is a "status" change, i.e., either a new job becomes available (n times) or a job in the schedule ends (n times). Each time Step (3) is invoked the total number of jobs instances that have to be examined is at most n (at most one for each job). To implement the algorithm we employ a priority queue that holds intervals according to their endpoint. At any point of time it is enough to hold at most one job instance for each job in the priority queue. It turns out that the total number of operations for retrieving the next instance is $O(n \log n)$, totalling to $O(n^2 \log n)$ operations.

4.2.2 Unrelated machines

If the number of unrelated machines is $k > 1$, we call Algorithm ADMISSION k times, machine by machine, in an arbitrary order, where the set of jobs considered in the i-th call does not contain the jobs already scheduled on machines $M_1, \ldots, M_{i-1}$. The analysis that shows how the $3 + 2\sqrt{2} \approx 5.828$ bound carries over to the case of unrelated machines is very similar to the analysis presented in the proof of Theorem 4.6. The main difference is in the definition of $R(I)$. For each interval $I \in \mathcal{A}$ that was executed on machine M_i, we define the set $R(I)$ to consist of I in case $I \in \mathcal{O}$, and of all the intervals that (i) were executed on machine M_i in the optimal schedule, and (ii) the rejection or preemption of these jobs was caused by I.

5 The MAX-SNP hardness

We show that the problem of scheduling unweighted jobs on unrelated machines is MAX-SNP Hard. This is done by reducing a variant of Max-2SAT, in which each variable occurs at most three times, to this problem. In this variant of Max-2SAT, we are given a collection of clauses, each consisting of two (Boolean) variables, with the additional constraint that each variable occurs at most three times, and the goal is to find an assignment of values to these variables that would maximize the number of clauses that are satisfied (i.e., contain at least one literal that has a "true" value). This problem is known to be MAX-SNP Hard (cf. [26]).

Given an instance of the Max-2SAT problem we show how to construct an instance of the problem of unweighted jobs, unrelated machines, such that the value of the Max-2SAT problem is equal to the value of the scheduling problem. Each variable x_i is associated with a machine M_i. Each clause C_j is associated with a job. The release time of any job is 0 and its deadline is 3. The job can be executed only on the two machines corresponding to the variables the clause C_j contains. (In other words, the processing time of the job in the rest of the machines is infinite.)

Suppose that clause C_j contains a variable x_i as a positive (negative) literal. The processing time of the job corresponding to C_j on M_i is $3/k$, where $k \in \{1, 2, 3\}$ is the number of occurrences of variable x_i as a positive (negative) literal. Note that in case variable x_i occurs in both positive and negative forms, it occurs exactly once in one of the forms since a variable x_i occurs at most three times overall. It follows that in any feasible schedule, machine M_i cannot execute both a job that corresponds to a positive literal occurrence and a job that corresponds to a negative literal occurrence.

We conclude that if m jobs can be scheduled, then m clauses can be satisfied. In the other direction, it is not hard to verify that if m clauses can be satisfied, then m jobs can be scheduled. Since Max-2SAT with the restriction that each variable occurs at most three times is MAX-SNP Hard, the unweighted jobs and unrelated machines case is MAX-SNP Hard as well.

Acknowledgment

We are indebted to Joel Wein for many helpful discussions, and especially for his suggestion to consider the general case of maximizing the throughput of jobs with release times and deadlines.

References

[1] M. Adler, A.L. Rosenberg, R.K. Sitaraman and W. Unger, Scheduling time-constrained communication in linear networks, *Proc. 10th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 269-278, 1998.

[2] S. Albers, N. Garg, and S. Leonardi, Minimizing stall time in single and parallel disk systems, *Proc. 30th Annual ACM Symposium on Theory of Computing*, pp. 454–462, 1998.

[3] E.M. Arkin and E.B. Silverberg, Scheduling jobs with fixed start and end times, *Discrete Applied Mathematics*, Vol.18, pp. 1–8, 1987.

[4] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi and A. Rosén, On-line competitive algorithms for call admission in optical networks, *Proc. 4th Annual European Symposium on Algorithms*, Lecture Notes in Computer Science, Vol. 1136, pp. 431–444, 1996.

[5] B. Awerbuch, Y. Bartal, A. Fiat and A. Rosén, Competitive non-preemptive call control, *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 312–320, 1994.

[6] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour and B. Schieber, Competitive Bandwidth Allocation with Preemption, To appear, *SIAM Journal on Computing*, Vol. 28, 1999.

[7] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha and F. Wang, On the competitiveness of on-line real-time task scheduling, *Real-Time Systems*, Vol. 4, pp. 125–144, 1992.

[8] J. Blazewicz, K.H. Ecker, G. Schmidt and J. Weglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, 1994.

[9] G. Cornuejols, M. Fisher and G. Nemhauser, Location of bank accounts to optimize float, *Management Science*, Vol. 23, pp. 789–810, 1977.

[10] U. Faigle and W.M. Nawijn, Note on scheduling intervals on-line, *Discrete Applied Mathematics*, Vol. 58, pp. 13–17, 1995.

[11] M. Fischetti, S. Martello and P. Toth, The fixed job schedule problem with spread-time constraints, *Operations Research*, Vol. 35, pp. 849–858, 1987.

[12] J.A. Garay, I.S. Gopal, S. Kutten, Y. Mansour and M. Yung, Efficient on-line call control algorithms, *Journal of Algorithms*, Vol. 23, pp. 180–194, 1997.

[13] M.R. Garey and D.S. Johnson, Two processor scheduling with start times and deadlines, *SIAM J. on Computing*, Vol. 6, pp. 416–426, 1977.

[14] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[15] S.A. Goldman, J. Parwatikar and S. Suri, On-line scheduling with hard deadlines, *Proc. 5th International Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science, Vol. 1272, pp. 258–271, 1997.

[16] M.H. Goldwasser, Patience is a Virtue: The Effect of Slack on Competitiveness for Admission Control, *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 396–405, 1999.

[17] N.G. Hall and M.J. Magazine, Maximizing the value of a space mission, *European Journal of Operational Research*, Vol. 78, pp. 224–241, 1994.

[18] H. Kise, T. Ibaraki and H. Mine, A solvable case of one machine scheduling problem with ready and due dates, *Operation research*, Vol. 26, pp. 121–126, 1978.

[19] G. Koren and D. Shasha, An optimal on-line scheduling algorithm for overloaded real-time systems. *SIAM Journal on Computing*, Vol. 24, pages 318–339, 1995.

[20] E.L. Lawler, Sequencing to minimize the weighted number of tardy jobs, *Recherche Operationnel*, Vol. 10, pp. 27–33, 1976.

[21] E.L. Lawler, A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs, *Annals of Operations Research*, Vol. 26, pp. 125–133, 1990.

[22] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity", in *Handbooks in Operations Research, Vol.4: Logistics for Production and Inventory (Eds. S.C. Graves, A.H.G. Rinnooy Kan and P.H. Zipkin)*, North Holland, 1993.

[23] R.J. Lipton and A. Tomkins, Online interval scheduling, *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 302–311, 1994.

[24] S. Sahni, Algorithms for scheduling independent tasks, *Journal of the ACM*, Vol. 23, pp. 116–127, 1976.

[25] J. Wein, *Personal Communication*, 1999.

[26] M. Yannakakis, On the approximation of maximum satisfiability, *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1–9, 1992.