Improved Approximation Schemes for Scheduling Unrelated Parallel Machines

Klaus Jansen Lorant Porkolab

MPI–I–98–1-026 October 1998

Author's Address

Klaus Jansen
IDSIA Lugano
Corso Elvezia 36
6900 Lugano
Switzerland
`klaus@idsia.ch`

Lorant Porkolab
Max-Planck-Institute for Computer Science
Im Stadtwald
66123 Saarbrücken
Germany
`porkolab@mpi-sb.mpg.de`

Acknowledgements

Research supported in part by EU ESPRIT LTR Project No. 20244 (ALCOM-IT) and by the Swiss Office Fédéral de l'éducation et de la Science project n 97.0315 titled "Platform".

Abstract

We consider the problem of scheduling n independent jobs on m unrelated parallel machines. Each job has to be processed by exactly one machine, processing job j on machine i requires p_{ij} time units, and the objective is to minimize the makespan, i.e. the maximum job completion time. We focus on the case when m is fixed and develop a fully polynomial approximation scheme whose running time depends only linearly on n. In the second half of the paper we extend this result to a variant of the problem, where processing job j on machine i also incurs a cost of c_{ij}, and thus there are two optimization criteria: makespan and cost. We show that for any fixed m, there is a fully polynomial approximation scheme that, given values T and C, computes for any fixed $\epsilon > 0$ a schedule in $O(n)$ time with makespan at most $(1+\epsilon)T$ and cost at most $(1+\epsilon)C$, if there exists a schedule of makespan T and cost C.

1 Introduction

Let n and m denote the number of jobs and machines, respectively. Suppose that the execution time of job j on machine i is p_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$. The objective is to find a non-preemptive schedule with minimum makespan, i.e. a schedule where each job is processed without any interruption by exactly one machine and the largest completion time is the smallest possible. Lenstra, Shmoys and Tardos [7] gave a polynomial-time 2-approximation algorithm for the general problem, and this is the currently known best approximation ratio achieved in polynomial time. They also proved that for any $\epsilon < 1/2$, no polynomial-time $(1+\epsilon)$-approximation algorithm exists, unless P=NP. Since the problem is NP-hard even for $m = 2$, it is natural to ask how well the optimum can be approximated when there are only a constant number of machines.

In contrast to the previously mentioned inapproximability result for the general case, there exists a fully polynomial-time approximation scheme for the problem when m is fixed. Horowitz and Sahni [4] proved that for any $\epsilon > 0$, an ϵ-approximate solution can be computed in $O(nm(nm/\epsilon)^{m-1})$ time, which is polynomial in both n and $1/\epsilon$ if m is constant. Lenstra, Shmoys and Tardos [7] also gave an approximation scheme for the problem with a running time which is bounded by the product of $(n + 1)^{m/\epsilon}$ and a polynomial of the input size. Even though for fixed m their algorithm is not fully polynomial, it has a much smaller space complexity than the one in [4]. In this paper we present a new approximation scheme for the problem whose running time is $n(m/\epsilon)^{O(m)}$. If there are only a constant number of machines, it gives a fully polynomial-time approximation scheme which computes for any fixed $\epsilon > 0$ an ϵ-approximate solution in $O(n)$ time. This linear complexity bound is a substantial improvement in terms of n compared to the above mentioned results. We note that the existence of a linear-time approximation scheme was known before only for a more restricted version of the problem [6], where not only m, but also the ratio $\max_{i,j} p_{ij}/\min_{i,j} p_{ij}$ is bounded by a constant.

Recently other variants of the above problem, where processing job j on machine i incurs a cost of c_{ij}, were also considered [11, 8, 10]. Improving the results in [11, 8], Shmoys and Tardos [10] gave a polynomial-time algorithm, that given values C and T, finds a schedule with cost at most C and makespan at most $2T$, if a schedule of cost C and makespan at most T exists. This algorithm requires $O(mn^2 \log n)$ time (for $n \geq m$), and the expected running time of a randomized variant is $O(n^2 \log n)$. We extend our result on the single criteria minimization described above to this bicriteria problem. In particular, we show that there is an approximation scheme that, given values T and C, computes for any $\epsilon > 0$ a schedule in $n(m/\epsilon)^{O(m)}$ time with makespan at most $(1 + \epsilon)T$ and cost at most $(1 + \epsilon)C$, if there exists a schedule of makespan T and cost C. For constant m, this gives a fully polynomial-time approximation scheme that runs in $O(n)$ time for any fixed $\epsilon > 0$. The rounding procedure used in this algorithm seems to be fairly general and hence also applicable for other integer packing problems.

2 Makespan Minimization

In this section we consider the simpler, single criteria (makespan) minimization problem and propose an approximation scheme that computes for any $\epsilon > 0$ a feasible schedule whose makespan is at most $(1+\epsilon)$ times the optimum. First we discuss separately the different steps in details, then we state the main algorithm and summarize the results in Section 2.3. In the following we assume without loss of generality that $m \geq 2$ and $0 < \epsilon < 1$.

2.1 Selecting and Scheduling Long Jobs. The algorithm first computes for every job j the shortest execution time $d_j = \min_{1 \leq i \leq m} p_{ij}$, which clearly can be done in $O(mn)$ time. Let $D = \sum_{j=1}^{n} d_j$, and let OPT denote the makespan of an optimal schedule. It is easy to check that $OPT \leq D \leq m \cdot OPT$. Then the algorithm selects a constant(m, ϵ) number of jobs (the so called large/long jobs) according to these d_j values. This constant $k = k(m, \epsilon)$ is chosen to guarantee that a certain number (this will be specified later) of small jobs (i.e. some of those which were not selected) can be scheduled such that the time they require is at most $\mu \cdot OPT$ (where μ will be some fraction of ϵ). The following simple combinatorial statement will be used to determine the appropriate constant k.

Lemma 2.1. *Suppose $d_1 \geq d_2 \geq \ldots \geq d_n > 0$ is a sequence of real numbers and $D = \sum_{j=1}^{n} d_j$. Let p be a nonnegative integer, $\alpha > 0$, and assume that n is sufficiently large (i.e. all the indices of the d_i's in the statement are smaller than n; e.g. $n > \lceil \frac{1}{\alpha} \rceil \cdot p$ suffices). Then, there exists an integer $k = k(p, \alpha)$ such that $d_k + \ldots + d_{k+p-1} \leq \alpha \cdot D$ and $k \leq p \cdot (\lceil 1/\alpha \rceil - 1) + 1$.*

We prove the following slightly more general result.

Lemma 2.2. *Suppose $d_1 \geq d_2 \geq \ldots d_n > 0$ is a sequence of real numbers and $D = \sum_{j=1}^{n} d_j$. Let p, q be nonnegative integers, $\alpha > 0$, and assume that n is sufficiently large (i.e. all the indices of the d_i's in the statement are smaller than n; e.g. $n > (\lceil \frac{1}{\alpha} \rceil p + 1)(q+1)^{\lceil \frac{1}{\alpha} \rceil}$ suffices). Then, there exists an integer $k = k(p, q, \alpha)$ such that*

$$d_k + \ldots + d_{k+p+qk-1} \leq \alpha \cdot D$$

and

$$k \leq (q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} + p[1 + (q+1) + \ldots + (q+1)^{\lceil \frac{1}{\alpha} \rceil - 2}]. \tag{2.1}$$

Proof. Decompose the sum $d_1 + \ldots + d_n$ into blocks $B_0 = d_1 + \ldots + d_{f(1)-1}$, $B_1 = d_{f(1)} + \ldots + d_{f(2)-1}$, $\ldots$, $B_i = d_{f(i)} + \ldots + d_{f(i+1)-1}$, where the function $f(i)$ is defined recursively by the following equation

$$f(i+1) = f(i) + p + q \cdot f(i), \quad f(0) = 1. \tag{2.2}$$

Since $\sum_{j=1}^{n} d_j = D$, at most $\lceil \frac{1}{\alpha} \rceil - 1$ blocks have size larger than $\alpha \cdot D$. Now let i be the smallest integer for which $B_i \leq \alpha \cdot D$. Then $i \leq \lceil \frac{1}{\alpha} \rceil - 1$, and $B_i = d_{f(i)} + \ldots + d_{f(i+1)-1} \leq$

$\alpha \cdot D$. This implies that there is an index $k \le f(i)$ such that $d_k + \ldots + d_{k+p+qk-1} \le \alpha \cdot D$. It follows from (2.2) that

$$f(i) = (q+1)^i + p[1 + (q+1) + \ldots + (q+1)^{i-1}]$$

which along with the bound on i implies (2.1). $\square$

If $q > 0$, then the bound on k in (2.1) simplifies to $[(p+q)(q+1)^{\lceil \frac{1}{\alpha} \rceil - 1} - p]/q$. It is clear that Lemma 2.1 can be obtained as a corollary of the previous result for $p = m$ and $q = 0$. We mention in passing that similar statements as Lemma 2.1 proved to be useful in designing approximation schemes for other scheduling problems as well, see e.g. [2, 5].

It will be shown later that during the process of scheduling all jobs the algorithm will encounter at most m small jobs which have to be scheduled at the end. To make sure that their execution requires at most $\mu \cdot OPT$ we use the previous result to select k, the number of long jobs, appropriately. Applying Lemma 2.1 with $p = m$ and $\alpha = \frac{\mu}{m}$, we obtain that there exists a (constant) $k \le m(\lceil \frac{m}{\mu} \rceil - 1)$ such that

$$d_{k+1} + \ldots + d_{k+m} \le \frac{\mu}{m} \cdot D \le \mu \cdot OPT.$$

This implies that any subset of jobs of cardinality at most m that contains none of the k longest jobs (with respect to the d_i's) can be scheduled with a total length of at most $\mu \cdot OPT$. We will use this fact in Section 2.3 to argue that the jobs with non-unique machine assignment (obtained from some linear program for short jobs) can be executed at the end (on the corresponding fastest machine) with a length bounded by $\mu \cdot OPT$. Selecting the $K = m\lceil \frac{m}{\mu} \rceil$ longest jobs and then finding the smallest k with the above property can be done in $O(Kn) = O(\frac{m^2}{\mu}n)$ time. Let T_l denote the set of the k longest jobs, and T_s the set of all others. Assume without loss of generality that $T_l = \{1, \ldots, k\}$. Then as we pointed out above, the total execution time for any subset of T_s of cardinality m can be bounded by $\mu \cdot OPT$.

Constructing all possible schedules of jobs $1, \ldots, k$ would require considering $m^k = m^{O(\frac{m^2}{\mu})}$ cases. To avoid the exponential dependence on $\frac{1}{\mu}$ (recall that our aim is to obtain a fully polynomial approximation scheme), we will not treat separately all of these schedules. First divide the interval $[0, D]$ into $N = \frac{mk}{\delta}$ equal parts $S_1, \ldots, S_N$ each of size at most $\frac{D\delta}{mk}$. Then by using a dynamic programming procedure called Algorithm L (see below) identify some of those schedules of $1, \ldots, k$ which do not differ too much. The same type of approach was used by Horowitz and Sahni [4] to compute exact and approximate solutions for various scheduling problems, however here we compute all of the substantially different schedules and not only the (approximately) best one. $T(j)$ denotes a set of m-tuples consisting the m completion times for different schedules of the first j jobs $1, \ldots, j$. In addition, we will also assume that there is a corresponding schedule available for every m-tuple in $T(j)$. Let E_{ij} be the $m \times k$ matrix all of whose entries are 0 except for the ij-entry which is 1. For $j = 1, \ldots, k$, let t_j denote the m-vector whose i-th coordinate is the processing time p_{ij} of job j on machine M_i. In the following description addition of m-tuples is done componentwise along with an update of the corresponding schedule, and the union is carried out such that if there are two m-tuples $(a_1, \ldots, a_m), (b_1, \ldots, b_m) \in \cup_{i=1}^{m} \{T(j-1) + E_{ij}^T t_j\}$ whose corresponding coordinate-pairs fall into the same intervals, i.e. $a_i, b_i \in S_{l_i}$ for some $l_i \in \{1, \ldots, N\}$ and for every $i = 1, \ldots, m$, then only one of them is retained. (It does not matter which one.)

Algorithm L

1. $T(0) \leftarrow \{(0, 0, \ldots, 0)\}$;
2. for $j = 1$ to k : $T(j) \leftarrow \bigcup_{i=1}^{m} \{ T(j-1) + E_{ij}^T t_j \}$;

Algorithm L, while constructing partial schedules, identifies those which on each machine i have finish times in the same subinterval S_{ℓ_i}, for some $\ell_i \in \{1, \ldots, N\}$. At each step when it proceeds from $(j-1)$ to j, the error it makes can be bounded by the length of the intervals, i.e. by $\frac{D\delta}{mk}$. Thus the total error is bounded by $k \cdot \frac{D\delta}{mk} = \frac{D\delta}{m} \le \delta \cdot OPT$. It is easy to check that Algorithm L can be implemented such that its running time is at most $O(kmN^m) = O(km(\frac{mk}{\delta})^m) = (\frac{m}{\delta})^{O(m)}$.

2.2 Scheduling Short Jobs by Linear Programming. For every m-tuple $t = (t_1, \ldots, t_m) \in T(k)$, consider the following integer linear program $ILP(t)$.

$$\begin{array}{lllll} \text{Min} & \lambda & & & \\ \text{s.t.} & \sum_{j=k+1}^{n} p_{ij} x_{ij} + t_i & \le & \lambda, & i = 1, \ldots, m, \\ & \sum_{i=1}^{m} x_{ij} & = & 1, & j = k+1, \ldots, n, \\ & x_{ij} \in \{0, 1\}, & & & j = k+1, \ldots, n, \ i = 1, \ldots, m. \end{array} \tag{2.3}$$

It formulates the original scheduling problem for the subset T_s of short jobs assuming (due to the schedule of long jobs corresponding to t) that machine i can start processing jobs only at time t_i. In any feasible solution of $ILP(t)$, $x_{ij} = 1$ indicates that job j is scheduled on machine i. Let $LP(t)$ denote the linear programming relaxation of $ILP(t)$ which is obtained by replacing every $0-1$ constraint ($x_{ij} \in \{0, 1\}$) with the weaker non-negativity condition ($x_{ij} \ge 0$). $LP(t)$ can also be viewed as a block optimization problem, where the blocks are m-dimensional simplicies,

$$B_j = \{x_j = (x_{1j}, \ldots, x_{mj}) \in R^m \mid \sum_{i=1}^{m} x_{ij} = 1, \ x_{ij} \ge 0, \ i = 1, \ldots, m\}, \quad j = k+1, \ldots, n,$$

and the coupling constraints are the linear inequalities $\sum_{j=k+1}^{n} p_{ij} x_{ij} + t_i \le \lambda$, $i = 1, \ldots, m$, corresponding to the m machines. By using the Logarithmic Potential Price Directive Decomposition Method [3] developed by Grigoriadis and Khachiyan for a large class of problems with block angular structure, we can compute a ρ-approximate solution $x(t)$ of $LP(t)$ in $m(\ln m + \rho^{-2}\ln\rho^{-1})$ iterations, where each iteration requires $O(m\ln\ln(m\rho^{-1}))$ operations and $(n-k)$ block optimizations performed to a relative accuracy of $O(\rho)$. In our case each block optimization is the minimization of a given linear function over an m-dimensional simplex which can be done (not only approximately, but even) exactly in $O(m)$ time. Therefore the overall running time of their procedure on $LP(t)$ is $O(n(\frac{m}{\rho})^2\ln\frac{m}{\rho})$. Let the length of a ρ-approximate solution $x(t)$ be

$$\lambda(x(t)) = \max\{\sum_{j=k+1}^{n} p_{ij}x(t)_{ij} + t_i \mid i = 1,\ldots,m\}.$$

The solutions of $LP(t)$ can be considered as fractional assignments between jobs and machines. A fractional assignment x can be represented by a bipartite graph $G = (V_1, V_2, E)$, where V_1 and V_2 correspond to machines and jobs, respectively, and $(i,j) \in E$ if and only if $x_{ij} > 0$. Any assignment x represented by a bipartite graph G of length d can be converted in $O(|E|m) = O(nm^2)$ time into another (fractional) assignment of length at most d represented by a forest [9] (see Lemma 5.1). Therefore assume that $x(t)$ is a ρ-approximate solution of $LP(t)$ represented by a forest. For a fractional assignment x, a job j has a non-unique machine assignment if there are at least two machines M_i and $M_{i'}$, $i \neq i'$, such that $x_{ij} > 0$ and $x_{i'j} > 0$. Given a fractional assignment x represented by a forest G, the number of jobs with non-unique job assignments can be bounded by $m-1$. To see this consider a connected component C of G with ℓ machines. In C we can have at most $\ell - 1$ jobs with non-unique job assignment, otherwise we would have at least 2ℓ edges in an induced subgraph of C or G with 2ℓ vertices.

After computing a ρ-approximate solution $x(t)$ for every $LP(t)$, $t \in T(k)$ we choose one with the smallest $\lambda(x(t))$ value. Assume it is attained for $t^* \in T(k)$. Let x^* denote the assignment vector for the (partial) schedule corresponding to t^* and those (integer) components of $x(t^*)$ which define a unique machine-job assignment. For a partial or complete assignment vector x, let $S(x)$ denote the makespan of the schedule determined by x.

Lemma 2.3. *$S(x^*) \le (1+\delta)(1+\rho)\cdot OPT$.*

Proof. For any $t \in T(k)$, let $OPT_{ILP}(t)$ and $OPT_{LP}(t)$ be the optimums of $ILP(t)$ and $LP(t)$, respectively. Assume that $\min_{t\in T(k)} OPT_{LP}(t) = OPT_{LP}(\tilde{t})$. Then

$$S(x^*) \le \lambda(x(t^*)) \le \lambda(x(\tilde{t})) \le (1+\rho)\cdot OPT_{LP}(\tilde{t}) \le (1+\rho)\cdot \min_{t\in T(k)} OPT_{ILP}(t). \quad (2.4)$$

The value $\min_{t\in T(k)} OPT_{ILP}(t)$ would be equal with OPT, if we had considered in $T(k)$ all possible schedules of the selected k longest jobs. Even though we do not have all possible schedules listed in $T(k)$, the error we may make by considering a schedule of $1,\ldots,k$ which is only approximately the same as the one corresponding to an optimal schedule of $1, \ldots, n$ can be bounded. According to the construction in Section 2.1, each of the components of the t-vector for an optimal schedule differ only by at most $\delta \cdot OPT$ from the components of a t-vector listed in $T(k)$. Therefore $\min_{t \in T(k)} OPT_{ILP}(t) \leq OPT + \delta \cdot OPT = (1+\delta) \cdot OPT$. Combining this inequality with (2.4) implies the lemma. $\square$

2.3 The Main Algorithm. Now we state the main algorithm and then summarize the results we have obtained so far.

Algorithm U

1. Compute $d_j = \min_{1 \leq i \leq m} p_{ij}$, $j = 1, \ldots, n$. Set $D = \sum_{j=1}^{n} d_j$, $\mu = \delta = \rho = \epsilon/4$, and $K = m\lceil \frac{m}{\mu} \rceil$.
2. Select the K longest jobs with respect to the d_j's. Assume $d_1 \geq d_2 \geq \ldots \geq d_K$. Find the smallest $k \leq K - m$ such that $d_{k+1} + \ldots + d_{k+m} \leq \frac{\mu}{m} \cdot D$. Partition the set of jobs into two sets T_l and T_s such that T_l contains the k longest jobs according to the d_j's.
3. Construct the set $T(k)$ of all substantially different (up to an accuracy of δ) schedules of the jobs in T_l by using Algorithm L. [See Section 2.1.]
4. For each $t \in T(k)$, compute a ρ-approximate solution of $LP(t)$, and then select from them one with the smallest objective function value. Assume that it is attained for $t^* \in T(k)$. [See Section 2.2.]
5. Convert this solution into another ρ-approximate solution $x(t^*)$ which can be represented by a forest.
6. Compute a partial schedule by using t^* and those components of $x(t^*)$ which specify a unique machine-job assignment.
7. Process those jobs which have fractional components in $x(t^*)$ at the end on those machines where their execution time is the smallest (i.e. according to the d_j's). [See below.]

Let $\{i_1, \ldots, i_p\}$ be the set of those jobs for which we have a non-unique machine assignment in $x(t^*)$. The argument in the previous section implies that $p \leq m$. Therefore by the selection of k, $d_{i_1} + \ldots + d_{i_p} \leq \mu \cdot OPT$. Thus these jobs can be executed at the end sequentially according to the d_i's (i.e. each of them is assigned to the machine where its execution time is the smallest) with a total length of at most $\mu \cdot OPT$. Let now $\overline{x}$ denote the assignment vector of the complete (final) schedule we have obtained. By combining the above bounds we obtain that

$$S(\overline{x}) \leq S(x^*) + \mu \cdot OPT \leq (1+\delta)(1+\rho) \cdot OPT + \mu \cdot OPT.$$

In order to guarantee that the makespan of the final schedule is bounded by $(1+\epsilon)\cdot OPT$, we only have to choose μ, δ, and ρ appropriately. For example setting $\mu = \rho = \delta = \epsilon/4$ suffices. Since all of the steps of Algorithm U requires at most $n(m/\epsilon)^{O(m)}$ time, we have shown the following result.

THEOREM 2.1. *There is an ϵ-approximation scheme for the non-preemptive minimum makespan scheduling problem with n jobs and m unrelated parallel machines that runs in $n(m/\epsilon)^{O(m)}$ time.*

For any fixed m, this result gives a fully polynomial approximation scheme whose running time depends only linearly on n.

3 Makespan and Cost

In this section we consider scheduling unrelated parallel machines with costs and extend the above result to the bicriteria problem in the following sense. We show that there is an approximation scheme that, given values T and C, computes for any $\epsilon > 0$ a schedule with makespan at most $(1+\epsilon)T$ and cost at most $(1+\epsilon)C$, if there exists a schedule of makespan T and cost C. The algorithm runs in $n(m/\epsilon)^{O(m)}$ time and thus it solves the problem in linear time for any fixed m and $\epsilon > 0$. Most of the steps of Algorithm U can be easily modified for the cost version of the problem, the only exception is the rounding procedure. This is discussed in details in Section 3.2.

3.1 Modifications in the Main Algorithm. Suppose that there exists a schedule with makespan at most T and cost at most C, and also assume without loss of generality that $T = C = 1$. Focusing on the case when m is constant, the objective is to compute an approximate solution whose makespan and cost are at most $1+\epsilon$. In the following we describe how the steps of the Main Algorithm stated in Section 2.3 should be modified in order to obtain an algorithm for the cost variant of the problem.

Step 1. Compute for each job $j \in J$ the value $d_j = \min_{i=1,\ldots,m}[p_{ij} + c_{ij}]$, and assume that $d_j = p_{m_j,j} + c_{m_j,j}$, for some $m_j \in \{1,\ldots,m\}$. Then $D = \sum_{j\in J} d_j \le m+1$, which can be easily verified as follows. Let $o_j \in \{1,\ldots,m\}$ be the assigned machine for job $j \in J$ in a schedule with makespan and cost at most 1. Furthermore, let $m_j \in \{1,\ldots,m\}$ denote a machine such that $d_j = p_{m_j,j} + c_{m_j,j}$. Since the load of each machine and the cost of the schedule is at most 1 (given by the machine assignment $f(j) = o_j$ for $j \in J$), the sum $D = \sum_{j\in J} d_j$ can be bounded by $\sum_{j\in J}[p_{o_j j} + c_{o_j j}] \le m+1$. It will be shown later that the number of small jobs which have to be rescheduled at the end can be bounded by m^2. By assuming that $d_1 \ge d_2 \ge \ldots \ge d_n > 0$ and applying Lemma 2.1 with $p = m^2$ and $\alpha = \epsilon/4(m+1)$, we obtain that there is an index $k \le m^2(\lceil 4(m+1)/\epsilon\rceil - 1)$ such that $d_{k+1} + d_{k+2} + \ldots + d_{k+m^2} < \epsilon/4$. Therefore set $K = m^2\lceil 4(m+1)/\epsilon\rceil$.

Step 2. It remains the same: determine k and the sets $T_l = \{1,\ldots,k\}$ and $T_s = \{k+1,\ldots,n\}$. Then for any subset $S \subseteq T_s$ of cardinality at most m^2, it holds that $\sum_{j\in S} p_{m_j,j} < \epsilon/4$, and $\sum_{j\in S} c_{m_j,j} < \epsilon/4$.

Step 3. Construct the set $T(k)$ of $m+1$ tuples $(t_1, \ldots, t_m, c)$ that correspond to substantially different schedules of jobs in T_l, but here also taking into account the cost in the $(m+1)$-st component. Clearly, Algorithm L can be easily modified to compute $T(k)$.

Step 4. For each $(m+1)$-tuple $(t_1, \ldots, t_m, c) \in T(k)$ we consider an integer linear program $ILP(t)$ which is almost identical with (2.3), the only difference is that it also includes the following inequality on costs: $\sum_{i=1}^{m} \sum_{j=k+1}^{n} c_{ij} x_{ij} + c \leq \lambda$. Thus the linear programming relaxation of $ILP(t)$ can be solved in the same way as before.

Step 6 and *Step 7* can also be easily extended to the cost variant of the problem with the same order of running time.

Step 5 is the only step that requires a substantially different approach. In this step the fractional assignment vector x (obtained as an approximate solution of the linear programming relaxation of $ILP(t)$) has to be rounded. Previously (see Section 2.2) this step was done by converting the fractional assignment vector to another one represented by a forest whose length is the same as the original one. Since in the bicriteria problem there is an additional inequality for the costs, this approach is not applicable any more (see Lemma 5.1 in [9]). Therefore considering the general case, Shmoys and Tardos [10] introduced a new rounding technique. They constructed a bipartite graph $B(x) = (V, E)$ with weights on the edges and used a minimum cost integer matching algorithm to get an integer solution with the same cost value as that of the original fractional solution while increasing the makespan by a factor of at most two. More precisely, assuming that the length of the fractional assignment computed in Step 4 is 1, the following holds for the schedule obtained by the rounding procedure in [10]: the total processing time on machine i is bounded by $1 + p_i^{max}$, where $p_i^{max} = \max\{p_{ij} : j \in J, x_{ij} > 0\}$. Let j_i be a job assigned to machine i by the minimum cost matching algorithm with largest processing time. Then there exists a schedule for $J' = J \setminus \{j_i | 1 \leq i \leq m\}$ whose makespan and cost are bounded by 1. In other words, for each machine $i \in \{1, \ldots, m\}$ we have at most one job j_i that increases the makespan, which would give all together at most m small jobs in our case. Computing the minimum cost matching in a bipartite graph $G = (V, E)$ requires at most $O(|V||E| + |V|^2 \log |V|)$ time [1]. The bipartite graph $B(x)$ has at most $|V| \leq 2n + m$ vertices and at most $|E| \leq 2nm$ edges. In general, this gives a $O(n^2 m + m^2 n + n^2 \log n + m^2 \log m)$ running time. For any fixed m, both $|V|$ and $|E|$ in $B(x)$ are bounded by $O(n)$. Therefore in this case rounding the fractional solution x by using the bipartite graph $B(x)$ can be done in $O(n^2 \log n)$ time.

3.2 Rounding in Linear Time. In this section, we show how the $O(n^2 \log n)$ running time for the rounding procedure described above can be improved to $O(n)$. Let x be the fractional assignment vector and consider two arbitrary machines, say 1 and 2 for simplicity. Let $J'_{\{1,2\}} = \{j \in J \mid x_{1j} > 0, x_{2j} > 0\}$. The goal is to modify the solution such that afterwards either $x_{1j} = 0$ or $x_{2j} = 0$ for every $j \in J'_{\{1,2\}}$, i.e. the number of fractional components decreases. Let $C_{\{1,2\}} = \sum_{j \in J'_{\{1,2\}}} (c_{1j} x_{1j} + c_{2j} x_{2j})$ and $D_i = \sum_{j \in J'_{\{1,2\}}} p_{ij} x_{ij}$, $i = 1, 2$, be the cost and the load on machine i, respectively, induced by the jobs in

$J'_{\{1,2\}}$. In the following the index $\{1,2\}$ of J' and C is omitted, since there will be no other machine pairs considered in the description below.

By using the transformations $x'_{ij} = \frac{1}{f_j} x_{ij}$, $p'_{ij} = f_j p_{ij}$ and $c'_{ij} = f_j c_{ij}$, where $f_j = x_{1j} + x_{2j} \in (0,1]$, we can assume that $x'_{1j} + x'_{2j} = 1$ for every job $j \in J'$. Note that the cost C and loads D_i, $i = 1,2$, are invariant with respect to these transformations. By dividing the processing times by D_i and the costs by C, we can also assume that both the cost C' and the loads D'_i, $i = 1,2$, are 1. (It is clear that a solution with loads and cost $1+\gamma$ for the transformed problem gives us a solution with loads $(1+\gamma)D_i$, $i = 1,2$, and cost $(1+\gamma)C$ for the original problem.) By using the matching approach one can compute an (integral) solution $\tilde{x}'$ for the transformed variant of the restricted problem where either $\tilde{x}'_{1j} = 1$ or $\tilde{x}'_{2j} = 1$, for every job $j \in J'$, which also provides a solution for the original problem where either $\tilde{x}_{1j} = f_j$ or $\tilde{x}_{2j} = f_j$. It is easy to check that replacing x_{ij} with $\tilde{x}_{ij}$ for each $j \in J'$, and repeating the above steps for each machine pair (M_1, M_2) with $M_1 \neq M_2$ result in an integral assignment. Therefore in order to obtain a linear-time rounding procedure for the general problem with any fixed number of machines it suffices to show that such an algorithm exists for two machines. The argument below provides a proof for this.

For each $j \in J'$, let $k_j = x'_{1j}(p'_{1j}+c'_{1j}) + x'_{2j}(p'_{2j}+c'_{2j})$, and $p'_{\ell_j j}+c'_{\ell_j j} = \min\{p'_{ij}+c'_{ij} \mid i = 1,2\}$, which along with $x'_{1j} + x'_{2j} = 1$ imply the following inequalities.

$$p'_{\ell_j j} + c'_{\ell_j j} \;\leq\; k_j \;\leq\; \max\{(p_{1j}+c_{1j}), (p_{2j}+c_{2j})\}, \quad j \in J'. \tag{3.5}$$

$$\sum_{j \in J'} k_j \;\leq\; 3. \tag{3.6}$$

In the following J' will be partitioned according to the k_j values into subsets $J'_1, \ldots J'_7$, and the jobs in the different subsets (along with their fractional componenets in x) will be considered separately. For every subset J'_ℓ, $\ell = 1, \ldots, 7$, let $C'(J'_\ell)$ and $D'_i(J'_\ell)$ denote the cost and the finishing time (or load) on machine i, $i = 1,2$, respectively, induced by the jobs in J'_ℓ. In the rounding procedure below μ is some kind of accuracy which will be specified later such that $\frac{1}{\mu} = O(\frac{m^2}{\epsilon})$. We can assume that $\mu n^{1/2-4\alpha} > 1$, since otherwise $n \leq (1/\mu)^{5/2} = (\frac{m}{\epsilon})^{O(1)}$, and thus the matching algorithm (applied to the general problem with m machines) provides a valid rounding in $(\frac{m}{\epsilon})^{O(1)}$ time.

Case 1: $J'_1 = \{j \in J' \mid k_j \leq \frac{\mu}{n}\}$.

Schedule each job $j \in J'_1$ on machine ℓ_j. By (3.5), $D'_i(J'_1) \leq \sum_{j \in J'_1 : \ell_j = i} p'_{ij} \leq |J'_1| \cdot \frac{\mu}{n} \leq \mu$. Similarly, the total cost of processing the jobs in J'_1 is bounded by μ.

Case 2: $J'_2 = \{j \in J' \mid k_j > \frac{\mu}{n^{1/2+\alpha}}\}$, where $\alpha > 0$ is a small constant.

By (3.6), $|J'_2| \leq 3n^{1/2-\alpha}/\mu$. Then one can compute an integral assignment for J'_2 and machines $\{1,2\}$ by the matching algorithm in $O(n^{1-2\alpha}\log n) = O(n)$ time. By removing the jobs j_1 and j_2 as described above, one can get an integral assignment for $J'_2 \setminus \{j_1, j_2\}$ with finishing times at most $D'_i(J'_2)$, $i = 1,2$, and cost at most $C'(J'_2)$.

Case 3: $J'_3 = \{j \in J' \mid \frac{\mu}{n^{2/3-\alpha}} \leq k_j \leq \frac{\mu}{n^{1/2+\alpha}}\}$, where $\alpha > 0$ is a small constant.

By (3.6), $|J_3'| \leq 3n^{2/3-\alpha}/\mu$. Partition J_3' into $\lfloor n^{1/3-\alpha}/2 \rfloor$ subsets such that each contains $O(n^{1/3})$ jobs, and optimize each block by the matching algorithm in $O(n^{2/3} \log n)$ time. This gives for all of the subsets a total running time of $O(n^{1-\alpha} \log n) = O(n)$. Except for at most two jobs per subset, one obtains an (integral) assignment for the jobs in J_3' with load at most $D_i'(J_3')$ and cost at most $C'(J_3')$. Every job j from these "exceptional" jobs can be rescheduled on machine ℓ_j, and by (3.5) the additional load and cost of all of these jobs can be bounded by $n^{1/3-\alpha}\mu/n^{1/2+\alpha} \leq \mu/n^{1/2-1/3+2\alpha} \leq \mu$.

Case 4: $J_4' = \{j \in J' \mid \frac{\mu}{n^{5/6-\alpha}} \leq k_j \leq \frac{\mu}{n^{2/3-\alpha}}\}$, where $\alpha > 0$ is a small constant.

This case can be analysed analogously as Case 3.

Case 5: $J_5' = \{j \in J' \mid \frac{\mu}{n} \leq k_j \leq \frac{\mu}{n^{5/6-\alpha}}$ and $p_{1,j}', p_{2j}' \in [0, \frac{\mu}{n^{5/6-\alpha}}]\}$.

Let $\tilde{p}_{ij} = \lfloor \frac{n}{\mu} p_{ij}' \rfloor$, $i = 1, 2$, $j \in J_5'$. By this rounding one allows an error of $\frac{\mu}{n}$ for each job, which will give a total additive factor of at most μ for all jobs in J_5'. For each pair $(s_1, s_2) \in \{0, \ldots, \lfloor n^{1/6+\alpha} \rfloor\} \times \{0, \ldots, \lfloor n^{1/6+\alpha} \rfloor\}$, consider the following subset of J_5'

$$J_5'(s_1, s_2) = \{j \in J_5' \mid \tilde{p}_{1j} = s_1, \tilde{p}_{2j} = s_2\}.$$

Then in the modified problem with the scaled and rounded p_{ij} values, all jobs in $J_5'(s_1, s_2)$ have the same execution times on machines 1 and 2. Furthermore, for these jobs in the original problem, the difference of execution times on the same machine is at most $\frac{\mu}{n}$. In the following, we solve the optimization problem for one set $J_5'(s_1, s_2)$ and show that the total additional loads and cost for all (s_1, s_2) pairs are at most 2μ and μ, respectively.

Optimization for one set $J_5'(s_1, s_2)$: Consider the scaled and rounded execution times $\tilde{p}_{ij}$. The finishing time of the jobs in $J_5'(s_1, s_2)$ on machine i, $i = 1, 2$, is $\tilde{D}_i(s_1, s_2) = \sum_{j \in J_5'(s_1,s_2)} x_{ij}' \tilde{p}_{ij} = s_i \sum_{j \in J_5'(s_1,s_2)} x_{ij}'$, and the total cost is $\tilde{C}(s_1, s_2) = \sum_{j \in J_5'(s_1,s_2)} x_{ij}' c'_{ij}$. Let $A_i = \sum_{j \in J_5'(s_1,s_2)} x_{ij}'$, $i = 1, 2$, let $A = A_1 + A_2 = |J_5'(s_1, s_2)|$. By the matching algorithm there exists an assignment for almost all jobs in $J_5'(s_1, s_2)$ with finishing times at most $\tilde{D}_i(s_1, s_2)$ and cost at most $\tilde{C}(s_1, s_2)$. We have only two jobs (one on machine 1 and one on machine 2) that can increase the finishing times. Since for each machine the execution times of all jobs are the same, one has to consider only three cases to find a good solution with cost at most $\tilde{C}(s_1, s_2)$:

(1) $\lfloor A_1 \rfloor + 1$ jobs on machine 1, $A - \lfloor A_1 \rfloor - 1$ jobs on machine 2.

(2) $\lfloor A_1 \rfloor$ jobs on machine 1, $A - \lfloor A_1 \rfloor$ jobs on machine 2.

(3) $\lfloor A_1 \rfloor - 1$ jobs on machine 1, $A - \lfloor A_1 \rfloor + 1$ jobs on machine 2.

Note that if A_1 is not integral, only the first two cases have to be considered. For each of these subcases, an approximate solution is computed and then a solution with minimum cost is selected.

Optimization for one subcase: One has to find a minimum cost solution with a jobs executed on machine 1 and $A - a$ jobs executed on machine 2. Let $c_1^{(1)}, c_2^{(1)}, \ldots, c_A^{(1)}$ be the cost of the jobs on machine 1, and let $c_1^{(2)}, c_2^{(2)}, \ldots, c_A^{(2)}$ be the cost of the jobs on machine 2. A solution is then a subset $I \subset \{1, \ldots, A\}$ of size $|I| = a$ with minimum cost $\sum_{j \in I} c_j^{(1)} + \sum_{j \notin I} c_j^{(2)}$. In other words, jobs in I are executed on the machine 1 and jobs in $\{1, \ldots, A\} \setminus I$ are executed on the machine 2. Let $C_i = \sum_{j=1}^{A} c_j^{(i)}$ for $i = 1, 2$. Then, the sum above is equal to $C_2 + \sum_{j \in I}(c_j^{(1)} - c_j^{(2)}) = \sum_{j \in I}(c_j^{(1)} - c_j^{(2)} + \frac{C_2}{a})$ (for $|I| = a$). Now let $d_j = c_j^{(1)} - c_j^{(2)} + \frac{C_2}{a}$. One can obtain a minimum cost solution with $|I| = a$ by sorting the d_j values in $O(n \log n)$ time and choosing the smallest a elements. Since the total cost is at most 1, it can be assumed that $d_j \leq 1$. By scaling and rounding with $\bar{d}_j = \lfloor \frac{|J_5'(s_1,s_2)|}{\bar{\mu}} d_j \rfloor$, one gets only integral values between 0 and $\frac{|J_5'(s_1,s_2)|}{\bar{\mu}}$. The $\bar{d}_j$ values, $j \in J_5'(s_1, s_2)$, can be sorted in $O(|J_5'(s_1, s_2)|)$ time. Furthermore, the error for each job in the original problem is at most $\frac{\bar{\mu}}{|J_5'(s_1,s_2)|}$.

Estimation of the cost: In total, for small constant $\bar{\mu}$, the scaling approximation algorithm gives us a solution $I \subset \{1, \ldots, A\}$ with $|I| = a$ and minimum cost multiplied with $(1 + \bar{\mu})$. We may choose $\bar{\mu} = \mu$. Repeating the previous computation for each of the three (or two) subcases above and selecting the best solution gives an assignment for the jobs in $J_5'(s_1, s_2)$ with cost $\tilde{C}(s_1, s_2)(1 + \mu)$. For all (s_1, s_2) pairs, this implies an additive factor of at most μ in the cost.

Estimation of the finishing times: The total finishing time on machine i for the jobs in $J_5'(s_1, s_2)$ can be bounded by:

$$\frac{\mu}{n} \cdot (\tilde{D}_i(s_1, s_2) + s_i) + \frac{\mu}{n} \cdot |J_5'(s_1, s_2)| \;\leq\; \bar{D}_i(s_1, s_2) + \frac{\mu}{n^{5/6-\alpha}} + \frac{\mu}{n} \cdot |J_5'(s_1, s_2)|,$$

where $\bar{D}_i(s_1, s_2)$ is the finishing time of the (fractional) jobs on machine i in the original problem instance. For all (s_1, s_2) pairs, we get a finishing time of at most

$$D_i'(J_5') + n^{2/6+2\alpha} \frac{\mu}{n^{5/6-\alpha}} + \mu \;\leq\; D_i'(J_5') + \frac{\mu}{n^{1/2-3\alpha}} + \mu.$$

If α is small enough (i.e. $\alpha < 1/6$), we get an additive factor of 2μ.

Case 6: $J_6' = \{j \in J' \mid \frac{\mu}{n} \leq k_j \leq \frac{\mu}{n^{5/6-\alpha}},\, p_{\ell j}' \leq \frac{\mu}{n^{5/6-\alpha}},\, p_{\ell' j}' > \frac{1}{n^{5/6-\alpha}},\ \text{for } 1 \leq \ell \neq \ell' \leq 2\}$. Schedule each job j on machine ℓ_j. It follows from (3.5) that in this case $\ell = \ell_j$.

Case 6.1: $J_6' \leq n^{5/6-\alpha}$.

Clearly, the total cost and finishing times for jobs in J_6' can be bounded by μ.

Case 6.2: $J_6' \leq n^{5/6-\alpha}$.

By using the matching algorithm, there is a schedule for the jobs in J_6' with cost at most $C'(J_6')$ and finishing times at most $D_i'(J_6') + p_i^{max}$ where now p_i^{max} is the maximum execution time of jobs in J_6' on machine i. Since $D_i'(J_6') \leq 1$, such a schedule can contain at most $2n^{5/6-\alpha}$ jobs assigned to a machine ℓ'. By scheduling all jobs in J_6' on machine ℓ, in almost all cases the correct matching edge is choosen in the bipartite graph. The finishing time of the other jobs can be bounded by $2n^{5/6-\alpha} \cdot \frac{\mu}{n^{5/6-\alpha}} = 2\mu$. By (3.5), $c_{\ell j}' \leq \frac{\mu}{n^{5/6-\alpha}}$, and therefore the total cost is increased by an additive factor of at most $2n^{5/6-\alpha} \cdot \frac{\mu}{n^{5/6-\alpha}} = 2\mu$.

Case 7: $J'_7 = \{j \in J' \mid \frac{\mu}{n} \leq k_j \leq \frac{\mu}{n^{5/6-\alpha}},\ p'_{\ell j} \leq \frac{\mu}{n^{5/6-\alpha}},\ \frac{\mu}{n^{1/3+3\alpha}} \geq p'_{\ell' j} > \frac{\mu}{n^{5/6-\alpha}},\ 1 \leq \ell \neq \ell' \leq 2\}$.
Assume without loss of generality that $\ell = 1$ and $\ell' = 2$. Let again $\tilde{p}_{ij} = \lfloor \frac{n}{\mu} p'_{ij} \rfloor$ with integral values between 0 and $\lfloor n^{1/6+\alpha} \rfloor$ for $\ell = 1$ and between $\lfloor n^{1/6+\alpha} \rfloor$ and $\lfloor n^{2/3-3\alpha} \rfloor$ for $\ell' = 2$. By this rounding, one allows an error of $\frac{\mu}{n}$ for each job executed on machine 1 and 2. For each pair $(s_1, s_2) \in \{0, \ldots, \lfloor n^{1/6+\alpha} \rfloor\} \times \{\lfloor n^{1/6+\alpha} \rfloor, \ldots, \lfloor n^{2/3-3\alpha} \rfloor\}$, consider the following subset of jobs

$$J'_7(s_1, s_2) = \{j \in J'_7 \mid \tilde{p}_{1j} = s_1, \tilde{p}_{2j} = s_2\}.$$

The number of (s_1, s_2) pairs is bounded by $n^{5/6-2\alpha}$. By using a similar method as in case 4 for the optimization of one job set $J'_7(s_1, s_2)$, one obtains for each pair (s_1, s_2) one additional job per machine, say job j on machine 1 and job j' on machine 2. But now place job j' also on machine 1 (to avoid increasing the finishing time on machine 2). Then, the finishing time on machine 1 is bounded by $\frac{\mu}{n} \cdot (\tilde{D}_1(s_1, s_2) + 2s_1) + \frac{\mu}{n} \cdot |J'_7(s_1, s_2)|$, and the one on machine 2 is at most $\frac{\mu}{n} \cdot (\tilde{D}_2(s_1, s_2)) + \frac{\mu}{n} \cdot |J'_7(s_1, s_2)|$. Thus the total finishing time for all (s_1, s_2) pairs on machine 2 is bounded by $D_2(J'_7) + \mu$, while the overall finishing time on machine 1 is at most $D_1(J'_7) + 2n^{5/6-2\alpha} \frac{\mu}{n^{5/6-\alpha}} + \mu \leq D_1(J'_7) + \frac{2\mu}{n^{\alpha}} + \mu$. For $\alpha > 0$, this gives an additive factor of 3μ. Similarly, the additional cost can be bounded by at most $\mu + n^{5/6-2\alpha} \frac{\mu}{n^{5/6-\alpha}} \leq 2\mu$. By considering both cases with $\ell = 1, 2$, one obtains additive factors of at most 4μ for the cost and finishing times on both machines.

Case 8: $J'_8 = \{j \in J' \mid \frac{\mu}{n} \leq k_j \leq \frac{\mu}{n^{5/6-\alpha}},\ p'_{\ell j} \leq \frac{\mu}{n^{5/6-\alpha}},\ \frac{1}{n^{5/6-\alpha}} \geq p'_{\ell' j} > \frac{\mu}{n^{1/3+3\alpha}},\ 1 \leq \ell \neq \ell' \leq 2\}$.
Case 9: $J'_9 = \{j \in J' \mid \frac{\mu}{n} \leq k_j \leq \frac{\mu}{n^{5/6-\alpha}}, p'_{\ell j} > \frac{\mu}{n^{5/6-\alpha}}, \ell = 1, 2\}$.
It follows easily from (3.5) and the assumption that $\mu n^{1/2-4\alpha} > 1$, that $J'_8 = \emptyset$ and $J'_9 = \emptyset$.

Estimation of the total increase in cost and makespan. All together there is an increase of at most 11μ both in the cost and the finishing times, and in addition there are at most 2 jobs which have to be rescheduled at the end (this happens only in Case 2). Recall that the above steps have to be repeated for each machine pair (M_1, M_2) with $M_1 \neq M_2$. Therefore after the $\binom{m}{2}$ iterations the cost and finishing times are increased by a multiplicative factor of at most $(1 + 11\mu)^{m^2}$, and there will be at most m^2 jobs to be rescheduled. By the selection of k, the latter jobs can be scheduled (on the corresponding fastest machine) such that both the total cost and the execution times are bounded by $\frac{\epsilon}{4}$. By solving approximately the linear programming relaxation of $ILP(t)$ for every $t \in T(k)$ and selecting the best solution, a fractional assignment x can be computed for which both the cost and the finishing times are bounded by $(1 + \delta)(1 + \rho)$. After the $\binom{m}{2} \leq m^2$ iterations of the new rounding procedure one obtains an integral assignment whose cost and finishing times are at most $(1 + \delta)(1 + \rho)(1 + 11\mu)^{m^2}$. It is easy to check that by selecting $\delta = \rho = \epsilon/9$ and $\mu = \epsilon/48m^2$ the multiplicative factor for the increase is bounded by $(1+\frac{\epsilon}{4})^2$. Taking also into account those jobs which had to be rescheduled, we obtain that the overall increase is at most $(1+\epsilon)$ in both the cost and the makespan.

THEOREM 3.1. *For any $\epsilon > 0$, one can compute in $n(m/\epsilon)^{O(m)}$ time a schedule with cost at most $(1+\epsilon)C$ and makespan at most $(1+\epsilon)T$, if one of cost C and makespan T exists.*

For any fixed m, this result gives a fully polynomial-time approximation scheme whose running time depends only linearly on n. Finally we state one further extension of the previous results to a problem which was also discussed in [10].

THEOREM 3.2. *There is an ϵ-approximation scheme of running time $n(m/\epsilon)^{O(m)}$ for minimizing a weighted sum of the makespan and the operating cost in non-preemptive scheduling of unrelated parallel machines.*

Proof. Consider the problem of minimizing the objective function consisting of a weighted sum of the makespan and operating costs:

$$\min \quad \mu T + \sum_{i,j} c'_{ij} x_{ij}$$

for some parameter $\mu > 0$, where the desired makespan is no longer part of the input. By using normalization ($c_{ij} = c'_{ij}/\mu$) and solving (approximately) the linear program for all jobs, we may assume that $\mu = 1$ and that the optimum $OPT \in [\frac{1}{3}, 1]$. Now, one can subdivide the square $[0,1] \times [0,1]$ into a discrete number of small squares and use the previous algorithm with accuracy $\bar{\epsilon}$ for each pair $(\alpha\delta, \beta\delta)$, where $1 > \delta > 0$ is a value specified later and $\alpha, \beta \in \{0, 1, \ldots, \lceil \frac{1}{\delta} \rceil\}$. The algorithm above tests all these pairs (S, T) and takes a feasible schedule with minimum objective value. If there is a schedule with makespan at most S and cost at most T, then the algorithm finds one with objective value at most $(S+T)(1+\bar{\epsilon})$. One of the small subsquares containes the optimum (T_{OPT}, C_{OPT}), and a pair $(\alpha'\delta, \beta'\delta)$ with the properties $\alpha'\delta \le T_{OPT} + \delta$, $\beta'\delta \le C_{OPT} + \delta$ is tested. Furthermore, we know that there exists a schedule with makespan at most $T = \alpha'\delta$ and cost at most $C = \beta'\delta$. Then the algorithm generates a schedule with makespan at most $T(1+\bar{\epsilon}) \le (T_{OPT} + \delta)(1+\bar{\epsilon})$ and cost at most $C(1+\bar{\epsilon}) \le (C_{OPT} + \delta)(1+\bar{\epsilon})$, which implies that the objective value of this schedule is bounded by

$$(C_{OPT} + T_{OPT} + 2\delta)(1+\bar{\epsilon}) = OPT(1+\bar{\epsilon}) + 2\delta(1+\bar{\epsilon}).$$

To show that this value is at most $(1+\epsilon)OPT$, it suffices to set $\bar{\epsilon} = \epsilon/2$, $\delta = \frac{1}{12} \cdot \frac{\epsilon}{1+\epsilon/2}$, and use the fact that $\frac{1}{3} \le OPT$. The number of calls to our previous algorithm is $O(\frac{1}{\delta^2}) = O(\frac{1}{\epsilon^2})$, which can also be improved by using binary search. Thus the theorem follows. □

4 Conclusions

We mention in closing that by using similar ideas linear time approximation schemes can also be obtained for the following related problems with a fixed number of machines: preemptive scheduling of unrelated parallel machines, flow and open shop scheduling, preemptive and non-preemptive malleable and/or non-malleable parallel task scheduling. The last problem was studied in [5], while the others along with some further extensions of the above results will be addressed in a subsequent paper.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network Flows, Prentice Hall, Englewood Cliffs, 1993.

[2] A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, Scheduling independent multiprocessor tasks, *Proc. 5th Annual European Symp. on Algorithms*, LNCS 1284, 1-12, 1997.

[3] M.D. Grigoriadis and L.G. Khachiyan, Coordination complexity of parallel price-directive decomposition, *Mathematics of Operations Research* 21 (1996), 321-340.

[4] E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *Journal of the Association for Computing Machinery* 23 (1976), 317-327.

[5] K. Jansen and L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, *Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms*, to appear, 1999.

[6] Y. Kopidakis, D. Fayard, and V. Zissimopoulos, Linear time approximation schemes for parallel processor scheduling, *Proc. 8th IEEE Symp. on Parallel and Distributed Processing*, 482-485, 1996.

[7] J.K. Lenstra, D.B. Shmoys, and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming* 46 (1990), 259-271.

[8] J.-H. Lin and J.S. Vitter, ϵ-approximations with minimum packing constraint violation, *Proc. 24th ACM Symp. on the Theory of Computing*, 771-782, 1992.

[9] S.A. Plotkin, D.B. Shmoys, and E. Tardos, Fast approximation algorithms for fractional packing and covering problems, *Mathematics of Operations Research* 20 (1995), 257-301.

[10] D.B. Shmoys and E. Tardos, An approximation algorithm for the generalized assignment problem, *Mathematical Programming* 62 (1993), 461-474.

[11] M.A. Trick, Scheduling multiple variable-speed machines, *Proc. 1st Integer Programming and Combinatorial Optimization Conference*, 485-494, 1990.