Grouping Techniques for Scheduling Problems: Simpler and Faster[*†]

Aleksei V. Fishkin[1], Klaus Jansen[2] and Monaldo Mastrolilli[3]
[1]Siemens AG, Munich, Germany, alexey.fishkin@siemens.com
[2]Universität zu Kiel, Germany, kj@informatik.uni-kiel.de
[3]IDSIA Lugano, Switzerland, monaldo@idsia.ch

Abstract

In this paper we describe a general grouping technique to devise faster and simpler approximation schemes for several scheduling problems. We illustrate the technique on two different scheduling problems: scheduling on unrelated parallel machines with costs and the job shop scheduling problem. The time complexity of the resulting approximation schemes is always linear in the number n of jobs, and the multiplicative constant hidden in the $O(n)$ running time is reasonably small and independent of the error ε.

1 Introduction

Scheduling is one of the fundamental areas of combinatorial optimization. Most multiprocessor and shop scheduling problems are known to be hard to solve optimally. Thus the research focuses on giving efficient approximation algorithms that produce a solution close to the optimal one. Ideally, one hopes to obtain a family of polynomial algorithms such that for any given $\varepsilon > 0$ the corresponding algorithm is guaranteed to produce a solution with a value within a factor of $(1 + \varepsilon)$ of the optimum value; such a family is called a polynomial time approximation scheme (PTAS). While always being polynomial in the input size, the running time of a PTAS may also depend on $1/\varepsilon$: the better the approximation, the larger may be the running time. Fully polynomial approximation schemes (FPTAS) are PTASs having running time growing polynomially with the reciprocal of the error bound. Some problems are known not to have fully polynomial approximation schemes unless NP=P. This lower bound applies to strongly NP-hard problems [3].

The problem of scheduling jobs on machines such that the maximum completion time (*makespan*) is minimized has been extensively studied for various

[*]Supported by Swiss National Science Foundation project 200020-109854, “Approximation Algorithms for Machine scheduling Through Theory and Experiments II”.

[†]A preliminary version of this paper appeared in the Proceedings of ESA’01.

problem formulations. Several polynomial time approximation schemes have been found for various shop and multiprocessor scheduling problems [1, 2, 6, 7, 8, 10, 11, 13, 14, 20]: these include scheduling problems on a single machine with release dates and delivery times, scheduling on unrelated machines, multiprocessor tasks (e.g. dedicated, parallel, malleable tasks), and classical open, flow and job shops. Some of these results were extended in [12] by providing a polynomial time approximation scheme for a general multiprocessor job shop scheduling problem (containing as special cases some of the problem formulations above).

The main goal of this paper is to emphasize a fairly general idea for obtaining approximation schemes for makespan (and related) minimization problems. The resulting approach can be applied to speed up and significatively simplify all the previous schemes for the aforementioned scheduling problems. In this paper we provide two schemes that indeed follow the same pattern. More precisely, we focus our attention on two problems: scheduling on unrelated machines with costs and the classical job shop scheduling problem. Although these two problems are different in nature (the first is an assignment problem while the other is an ordering problem), the reader will recognize that the underlying ideas are very similar. Furthermore, the described techniques can also be applied to many other scheduling problems including the general multiprocessor job shop scheduling problem studied in [12] (see [17, 18] for other examples). In both cases the overall running time is $O(n) + C$, where the constant hidden in $O(n)$ is reasonably small and independent of the accuracy ε, whereas the additive constant C depends on the number of machines, the accuracy ε (and the number of operations for job shops).

The Basic Idea. The basic idea is to reduce the number of jobs to a constant and apply enumeration or dynamic programming afterwards. To accomplish this, we perform the following steps:

1. **Rounding and Profiling.** We first round the job data (processing times and costs) such that, in the resulting rounded instance, there are only a constant number of different *job profiles*. Jobs sharing the same profile may differ only by a different *scale factor*. It follows that each job is completely characterized by a job profile and a scale factor.

2. **Grouping.** Jobs with the same profile and having a sufficiently "small" scale factor are merged (*grouped*) to form new jobs. This way the number of jobs can be reduced in linear time to a constant number.

3. **Enumerating.** Finally, dynamic programming or enumeration yield a (fully) polynomial time approximation scheme.

The bound on the approximation error ε is proved by showing that two linear programming formulations (one for the original instance and one for the transformed instance) have a gap of at most εOPT, where OPT is the minimum objective value.

We remark that the general idea of rounding job data and merging jobs in order to reduce the total number of jobs is not new (see e.g. [19]). However, the proposed combination reduces the number of jobs immediately to a constant and leads to elegant and simpler approximation schemes for the two considered scheduling problems.

First Example: Unrelated Parallel Machines with Costs. We begin with the problem of scheduling a set $\mathcal{J} = \{J_1, ..., J_n\}$ of n independent jobs on a set $M = \{1, ..., m\}$ of m unrelated parallel machines. Each machine can process at most one job at a time, and each job has to be processed without interruption by exactly one machine. Processing job J_j on machine i requires $p_{ij} \geq 0$ time units and incurs a cost $c_{ij} \geq 0$, $i = 1, \ldots, m$, $j = 1, ..., n$. We consider the problem of minimizing the objective function that is a weighted sum of the maximum job completion time among all jobs (makespan) and the total cost.

When $c_{ij} = 0$ the problem turns into the classical makespan minimization form. Lenstra, Shmoys and Tardos [16] gave a polynomial-time 2-approximation algorithm for this problem; and this is the currently known best approximation ratio achieved in polynomial time. They also proved that for any positive $\varepsilon < 1/2$, no polynomial-time $(1 + \varepsilon)$-approximation algorithm exists, unless P=NP. Furthermore, Shmoys and Tardos [23] gave a polynomial-time 2-approximation algorithm for the general variant with cost. Since the problem is NP-hard even for $m = 2$, it is natural to ask how well the optimum can be approximated when there is 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 [9] proved that for any $\varepsilon > 0$, an ε-approximate solution can be computed in $O(nm(nm/\varepsilon)^{m-1})$ time, which is polynomial in both n and $1/\varepsilon$ if m is constant. Lenstra, Shmoys and Tardos [16] also gave an approximation scheme for the problem with running time bounded by the product of $(n + 1)^{m/\varepsilon}$ 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 [9]. Jansen and Porkolab [10] presented a fully polynomial-time approximation scheme for the problem whose running time is $n(m/\varepsilon)^{O(m)}$. They combine the previous (dynamic [9] and linear [16] programming) approaches. Their algorithm has to solve at least $(m^3/\varepsilon^2)^m$ many linear programs. In order to obtain a linear running time for the case when m is fixed, they use the price-directive decomposition method proposed by Grigoriadis and Khachiyan [5] for computing approximate solutions of block structured convex programs. The final ingredient is an intricate rounding technique based on the solution of a linear program and a partition of the job set.

In contrast to the previous approach [10], our algorithm (that works also for the general variant with cost) is extremely simple and follows the basic idea sketched before: first we preprocess the data to obtain a new instance with $\min\left\{n, (\log m/\varepsilon)^{O(m)}\right\}$ grouped jobs. The preprocessing step requires linear time. Then using dynamic programming we compute an approximate solution for the grouped jobs in $(\log m/\varepsilon)^{O(m^2)}$ time. Both steps together imply a fully polynomial-time approximation scheme that runs in $O(n) + C$ time where $C = (\log m/\varepsilon)^{O(m^2)}$. We remark that the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

Second Example: Makespan Minimization in Job Shops. In the job shop scheduling problem, there is a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs that must be processed on a given set $M = \{1, \ldots, m\}$ of m machines. Each job J_j consists of a sequence of μ operations $O_{1j}, O_{2j}, \ldots, O_{\mu j}$ that need to be processed in this order. Operation O_{ij} must be processed without interruption on machine $m_{ij} \in M$, during p_{ij} time units. Each machine can process at most one operation at a time, and each job may be processed by at most one machine at any time. For any given schedule, let C_{ij} be the completion time of operation O_{ij}. The objective is again to find a schedule that minimizes the maximum completion time $C_{\max} = \max_{ij} C_{ij})$.

The job shop problem is strongly NP-hard even if each job has at most three operations and there are only two machines [15]. Williamson et al. [24] proved that when the number of machines, jobs, and operations per job are part of the input there does not exist a polynomial time approximation algorithm with worst case bound smaller than $\frac{5}{4}$ unless $P = NP$. When m and μ are part of the input the best known result [4] is an approximation algorithm with worst case bound $O((\log(m\mu) \log(min(m\mu, p_{max}))/ \log\log(m\mu))^2)$, where p_{max} is the largest processing time among all operations. For those instances where m and μ are fixed (the restricted case we are focusing on in this paper), Shmoys et al. [22] gave approximation algorithms that compute $(2+\varepsilon)$-approximate solutions in polynomial time for any fixed $\varepsilon > 0$. This result has been improved by Jansen et al. [13] who have shown that $(1 + \varepsilon)$-approximate solutions of the problem can be computed in polynomial time. The main idea is to divide the set of jobs $\mathcal{J}$ into two groups $\mathcal{L}$ and $\mathcal{S}$ formed by jobs with "large" and "small" total processing time, respectively. The total number of large jobs is bounded by a constant exponentially in m, μ and ε. Then they construct all possible schedules for the large jobs. In any schedule for the large jobs, the starting and completion times of the jobs define a set of time intervals, into which the set of small jobs have to be scheduled. Then for every possible job ordering of large jobs, a linear program is used to assign small jobs to time intervals. The rounded solution of the linear program in combination with an algorithm by Sevastianov [21], gives an approximate schedule in time polynomial in n. In order to speed up the whole algorithm, in [14] is suggested a number of improvements: they use the logarithmic potential price decomposition method of Grigoriadis and Khachiyan [5] to compute an approximate solution of the linear program in linear time, and a novel rounding procedure to bring down to a constant the number of fractional assignments in any solution of the linear program. The overall running time is $O(n)$, where the multiplicative constant hidden in the

$O(n)$ running time is exponentially in m, μ and ε.

In contrast to the approximation schemes introduced in [13, 14], our algorithm is again extremely simple and even faster. We show that we can preprocess in linear time the input to obtain a new instance with a constant number of grouped jobs. This immediately gives a linear time approximation scheme with running time $O(n)+C$, where C is a constant that depends on m, ε and μ. Again, we remark that the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

Note. Our approach uses several transformations of the input instance which may potentially increase the objective function value by a factor of $1 + O(\varepsilon)$. Therefore we can perform a constant number of them while still staying within $1 + O(\varepsilon)$ of the original optimum. Throughout this paper, when we describe this type of transformation, we shall say it produces $1 + O(\varepsilon)$ *loss*.

2 Unrelated Parallel Machines with Costs

The problem can be stated by using the following integer linear program ILP that represents the problem of assigning jobs to machines ($x_{ij} = 1$ means that job J_j has been assigned to machine i, and μ is any given positive weight):

$$\begin{array}{lll} \min & T + \mu \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} c_{ij} & \\ s.t. & \sum_{j=1}^{n} x_{ij} p_{ij} \leq T, & i = 1, \ldots, m; \\ & \sum_{i=1}^{m} x_{ij} = 1, & j = 1, ..., n; \\ & x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, & j = 1, ..., n. \end{array}$$

The first set of constraints relates the makespan T to the processing time on each of the machines, while the second set ensures that every job gets assigned.

Let $0 < \varepsilon < 1$ be an arbitrary small rational number, and let $m \geq 2$ be an integral value. Throughout this section, the values ε and m are considered to be constants and not part of the input. The outline of the scheme is as follows.

Step 1 Round the processing times of the jobs (see Section 2.1).

Step 2 Merge jobs together of the same profile with small scale factor to obtain a constant number of jobs (see Section 2.2).

Step 3 Use a dynamic programming algorithm to compute an approximate solution for the job set obtained in Step 2 (see Section 2.3).

Optimal Value Bounds. We begin by computing some lower and upper bounds for the minimum objective value OPT that will be useful in the following. By multiplying each cost value by μ we may assume, w.l.o.g., that $\mu = 1$. Let

$$d_j = \min_{i=1,\ldots,m} (p_{ij} + c_{ij}),$$

and

$$D = \sum_{j=1}^{n} d_j.$$

Consider an optimal assignment (x^*_{ij}) of jobs to machines with makespan T^* and total cost C^*. Then,

$$\begin{aligned} D &= \sum_{j=1}^{n} d_j \leq \sum_{i=1}^{m}\sum_{j=1}^{n} x^*_{ij} c_{ij} + \sum_{i=1}^{m}\sum_{j=1}^{n} x^*_{ij} p_{ij} \\ &\leq C^* + m \cdot T^* \leq m \cdot OPT, \end{aligned}$$

where $OPT = T^* + C^*$. On the other hand, we can generate a feasible schedule according to the d_j values. Indeed, let $m_j \in \{1, ..., m\}$ denote any machine such that $d_j = p_{m_j,j} + c_{m_j,j}$. Assign every job J_j to machine m_j. The objective value of this schedule can be bounded by $\sum_{j \in J} c_{m_j,j} + \sum_{j \in J} p_{m_j,j} = D$. Therefore, $OPT \in [D/m, D]$, and by dividing all processing times and cost values by D/m, we get directly the following bounds for the optimum value:

$$1 \leq OPT \leq m.$$

2.1 Rounding and Profiling Jobs

For each job J_j, we define four sets of machines: the set of *fast*

$$\mathcal{F}_j = \left\{ i : p_{ij} \leq \frac{\varepsilon}{m} d_j \right\}$$

the set of *cheap*

$$\mathcal{C}_j = \left\{ i : c_{ij} \leq \frac{\varepsilon}{m} d_j \right\}$$

the set of *slow*

$$\mathcal{S}_j = \left\{ i : p_{ij} \geq \frac{m}{\varepsilon} d_j \right\}$$

and the set of *expensive*

$$\mathcal{E}_j = \{ i : c_{ij} \geq d_j/\varepsilon \}$$

machines, respectively.

Rounding the Input. According to the above definitions, the input data of any job J_j $(j = 1, \ldots, n)$ is rounded as follows.

- For any possible fast machine $i \in \mathcal{F}_j$ for job J_j, round the corresponding processing time to zero, i.e. set $p_{ij} := 0$. Similarly, for any $i \in \mathcal{C}_j$ set $c_{ij} := 0$.
- For any possible slow machine $i \in \mathcal{S}_j$ for job J_j, set the corresponding processing time p_{ij} to be a "large enough" number such that no reasonable algorithm would schedule that job on a slow machine ($p_{ij} \geq 2m$ suffices); let us write $p_{ij} := +\infty$. Similarly, for any $i \in \mathcal{E}_j$, set $c_{ij} := +\infty$. (Observe that, by definition, for all jobs J_j there exists always a machine i which is neither expensive nor slow.)

- For any other machine i of job J_j, round the processing time p_{ij} and cost c_{ij} down to the nearest lower value of $\frac{\varepsilon}{m} d_j (1+\varepsilon)^h$, for some $h \in \mathbb{N}$.

Lemma 1 *Rounding the input produces* $1+4\varepsilon$ *loss.*

Proof. Let us start by considering only the effect of rounding to zero the processing times and costs of jobs on fast and cheap machines, respectively. Let $A : \mathcal{J} \to M$ be an optimal assignment of jobs to machines for this modified instance. Clearly, the optimal value corresponding to A cannot be larger than OPT (we just reduced the processing times and costs). Let F and C denote the set of jobs which are processed on fast and cheap machines according to A. Now, if we replace the processing times and costs of the transformed instance by the original processing times and costs, we may potentially increase the value of A by at most

$$\sum_{J_j \in F} \frac{\varepsilon}{m} d_j + \sum_{J_j \in C} \frac{\varepsilon}{m} d_j \leq 2 \sum_{j=1}^{n} \frac{\varepsilon}{m} d_j = 2\varepsilon \frac{D}{m} = 2\varepsilon.$$

Now we show that there exists an approximate schedule where jobs are scheduled neither on slow nor on expensive machines. This allows us to set $p_{ij} = +\infty$, for $i \in \mathcal{S}_j$, and $c_{ij} = +\infty$, for $i \in \mathcal{E}_j$. Consider an optimal assignment $A : \mathcal{J} \to M$ of jobs to machines with T^* and C^* denoting the resulting makespan and cost, respectively. Let S and E represent, respectively, the set of jobs which are processed on slow and on expensive machines according to A. Then, assign every job $J_j \in S \cup E$ to machine m_j (recall that $m_j \in \{1, ..., m\}$ denote any machine such that $d_j = p_{m_j,j} + c_{m_j,j}$). Moving jobs $J_j \in S \cup E$ onto machines m_j may potentially increase the objective value by at most

$$\sum_{J_j \in S \cup E} d_j \quad \leq \quad \frac{\varepsilon}{m} \sum_{J_j \in S} p_{A(j),j} + \varepsilon \sum_{J_j \in E} c_{A(i),j} \leq \varepsilon T^* + \varepsilon C^*,$$

since $p_{A(j),j} \geq \frac{m}{\varepsilon} d_j$ for $J_j \in S$, and $c_{A(j),j} \geq d_j/\varepsilon$ for $J_j \in E$.

Observe that by the above arguments, all the positive costs c_{ij} and positive processing times p_{ij} are greater than $\frac{\varepsilon}{m} d_j$. The last change is such that every positive processing time p_{ij} and every positive cost c_{ij} is rounded down to the nearest lower value of $\frac{\varepsilon}{m} d_j (1+\varepsilon)^h$, for $h \in \mathbb{N}$, $i = 1, ..., m$ and $j = 1, ..., n$. Consider the optimal value of the rounded instance. Clearly, this value cannot be greater than OPT. It follows that by replacing the rounded values with the original ones we may increase each value by a factor $1+\varepsilon$, and consequently, the solution value potentially increases by the same factor $1+\varepsilon$. ■

Job Profiles and Scale Factors. Consider the input instance after the rounding step described above. We define the *execution profile* of job J_j to be an m-tuple $< \Pi_{1,j}, ..., \Pi_{m,j} >$ such that $p_{ij} = \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\Pi_{i,j}}$. We adopt the convention that $\Pi_{i,j} = +\infty$ if $p_{ij} = +\infty$, and $\Pi_{i,j} = -\infty$ if $p_{ij} = 0$. Likewise, we define the *cost profile* of job J_j to be an m-tuple $< \Gamma_{1,j}, ..., \Gamma_{m,j} >$ such that $c_{ij} = \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\Gamma_{i,j}}$. Again, we adopt the convention that $\Gamma_{i,j} = +\infty$ if $c_{ij} = +\infty$, and $\Gamma_{i,j} = -\infty$ if $c_{ij} = 0$. Let us say that two jobs have the same *job profile* iff they have the same execution and cost profile. Observe that the value of d_j plays the role of a *scale factor*, and two jobs with the same profile may differ only by their scale factors.

Lemma 2 *The number of distinct profiles is at most* $\ell := \left(3 + 2\log_{1+\varepsilon} \frac{m}{\varepsilon}\right)^{2m}$.

Proof. Every profile is determined by $2m$ entries. Each entry can be either $\pm\infty$ or any nonnegative integer h with $\frac{\varepsilon}{m} d_j (1+\varepsilon)^h \leq \frac{m}{\varepsilon} d_j$. ■

2.2 Grouping Jobs

Consider the input instance rounded as described in Section 2.1. Let $\delta := \frac{\varepsilon}{m}$, and partition the set of jobs into two subsets $L = \{J_j : d_j > \delta\}$ and $S = \{J_j : d_j \leq \delta\}$ according to their scale factors. Let us say that L is the set of *large* jobs, while S the set of *small* jobs. We further partition the set of small jobs into subsets S_i of jobs having the same profile, for $i = 1, ..., \ell$. Let J_a and J_b be two jobs from S_i such that $d_a, d_b \leq \delta/2$. We "group" together these two jobs to form a composed job J_c in which the processing time (and cost) on machine i is equal to the sum of the processing times (and costs) of J_a and J_b on machine i, and let $d_c = d_a + d_b$. We repeat this process, by using the modified set of jobs, until at most one job $J_j \in S_i$ is left with $d_j \leq \delta/2$. At the end, all jobs J_j in group S_i have $d_j \leq \delta$. The same procedure is performed for all other subsets S_i. At the end of this process, there are at most ℓ jobs, one for each subset S_i, having $d_j \leq \delta/2$. All the other jobs, have processing times larger than $\delta/2$. Therefore, the number of jobs in the transformed instance is bounded by $\frac{2D}{\delta} + \ell \leq \frac{2m^2}{\delta} + \ell = (\log m/\varepsilon)^{O(m)}$. Note that the procedure runs in linear time, and a feasible schedule for the original set of jobs can be easily obtained from a feasible schedule for the grouped jobs. We motivate the described technique with the following result.

Lemma 3 *With* $1+\varepsilon$ *loss, the number of jobs can be reduced in linear time to be at most* $\min\{n, (\log m/\varepsilon)^{O(m)}\}$.

Proof. Consider the transformed instance I' right after the rounding step (but before grouping). Assume, w.l.o.g., that there exists an optimal solution SOL' for I' of value OPT'. It is sufficient to show that, by using the small jobs grouped as described previously, there exists a schedule of value $(1+\varepsilon)OPT'$.

Accordingly to SOL' we may assume that each machine executes the large jobs at the beginning of the schedule. Let c denote the total cost of large jobs when processed according to SOL', and let t_i denote the time at which machine i finishes to process large jobs, for $i = 1, ..., m$. Now, consider the following linear program LP_1:

$$\begin{array}{lllr}
\min & T + c + \sum_{i=1}^{m} \sum_{J_j \in S} x_{ij} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\Gamma_{i,j}} & & \\
s.t. & t_i + \sum_{J_j \in S} x_{ij} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\Pi_{i,j}} \leq\ T, & & i = 1, \ldots, m; \\
 & \sum_{i=1}^{m} x_{ij} =\ 1, & & J_j \in S; \\
 & x_{ij} \geq 0, & i = 1, \ldots, m, & J_j \in S.
\end{array}$$

Note that LP_1 formulates the integer relaxation of the original problem ILP for the subset of small jobs: we are assuming that machine i can start processing small jobs only at time t_i, and when the processing times and costs are structured as in Lemma 1.

For each S_ϕ, $\phi = 1, ..., \ell$, consider a set of decision variables $y_{\phi i} \in [0, 1]$ for $i = 1, ..., m$. The meaning of these variables is that $y_{\phi i}$ represents the fraction of jobs from S_ϕ processed on machine i. Consider the following linear program LP_2:

$$\begin{array}{lllr}
\min & T + c + \sum_{i=1}^{m} \sum_{\phi=1}^{\ell} y_{\phi i} \sum_{J_j \in S_\phi} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\Gamma_{i,j}} & & \\
s.t. & t_i + \sum_{\phi=1}^{\ell} y_{\phi i} \sum_{J_j \in S_\phi} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\Pi_{i,j}} \leq\ T, & & i = 1, \ldots, m; \\
 & \sum_{i=1}^{m} y_{\phi i} =\ 1, & & \phi = 1, \ldots, \ell; \\
 & y_{\phi i} \geq 0, & i = 1, \ldots, m, & \phi = 1, \ldots, \ell.
\end{array}$$

By setting

$$y_{\phi i} = \frac{\sum_{J_j \in S_\phi} \ x_{ij} d_j}{\sum_{J_j \in S_\phi} d_j},$$

it is easy to check that any feasible set of values (x_{ij}) for LP_1 gives a feasible set of values $(y_{\phi i})$ for LP_2 (recall that jobs belonging to any S_ϕ have the same profile). Furthermore, by these settings, the objective function value of LP_2 is equal to that of LP_1. But LP_1 is a relaxation of the original problem. Therefore, if we were able, by using the grouped small jobs, to get a feasible schedule of length at most $1 + \varepsilon$ times the optimal value of LP_2, we would be done. In the remainder we show how to generate such a schedule. This solution is obtained by using the solution of LP_2 and the small grouped jobs.

Let us denote by $y^*_{\phi i}$ the values of variables $y_{\phi i}$ according to the optimal solution of LP_2. For every positive value $y^*_{\phi i}$, schedule a subset of grouped jobs from S_ϕ on machine i until either (a) the jobs from S_ϕ are exhausted or (b) the total fraction of jobs assigned to i is equal to $y^*_{\phi i}$ (if necessary fractionalize one job to use up $y^*_{\phi i}$ exactly). We repeat this for the not yet assigned grouped small jobs and for every positive value $y^*_{\phi i}$. Note that if $y^*_{\phi i}$ is not fractional, then the jobs from S_ϕ are not preempted by the previous algorithm. In general, the number of preempted jobs from S_ϕ is at most $f_\phi - 1$, where $f_\phi = \left| \left\{ y^*_{\phi i} : y^*_{\phi i} > 0, i = 1, ..., m \right\} \right|$. Now remove all the preempted jobs J_j and schedule them at the end on machines m_j. This increases the makespan and the cost by at most $\Delta = \delta \cdot \sum_{\phi=1}^{\ell} (f_\phi - 1)$, since every grouped small job has cost plus processing time bounded by δ when processed on machine m_j. A basic feasible solution of LP_2 has the property that the number of positive variables is at most the number of rows in the constraint matrix, $m + \ell$, therefore $\sum_{\phi=1}^{\ell}(f_\phi - 1) \leq m$. In order to bound the total increase Δ by ε we have to choose δ such that $\delta \leq \frac{\varepsilon}{m}$, and the claim follows. ■

2.3 Dynamic Programming

The optimal solution for the transformed instance with $k = (\log m/\varepsilon)^{O(m)}$ jobs can be computed in $O(m^k)$ time by simply enumerating all possible assignments and retaining the smallest one. However, to avoid the exponential dependence on $1/\varepsilon$ (recall, our aim is to obtain a fully polynomial approximation scheme), we will not treat separately all of these schedules. Furthermore, for our problem we do not need to find an optimal solution for the transformed instance, an approximate solution suffices.

In the following, we show that an approximate schedule for the transformed instance can be computed in $(\log m/\varepsilon)^{O(m^2)}$ time. The first step is to round down every processing time and cost to the nearest lower value of $(\varepsilon/k)i$, for $i = 0, 1, \ldots, k/\varepsilon$; clearly this does not increase the objective function value. Then, find the optimal solution SOL of the resulting instance by using a dynamic programming approach. Finally, by replacing the rounded values with the originals, it is easy to check that we may potentially increase the cost and the makespan of SOL by at most ε, respectively. This results in a $(1 + 2\varepsilon)$-approximate solution for the transformed instance.

We now present the dynamic programming algorithm. Let $J_1, ..., J_k$ denote the k jobs of the transformed instance. Let a *schedule configuration* $\mathbf{s} =(t_1, ..., t_m, c)$ be defined as an $(m + 1)$-dimensional vector, where t_i denotes the completion time of machine i and c the total cost. For each job J_j let V_j denote the set of $(m + 1)$-dimensional vectors defined as follows: for each $i = 1, ..., m$, there is a vector $\mathbf{v} \in V_j$ whose entries are 0 except for the ith component which is p_{ij}, and the $(m + 1)$-st component which is c_{ij}. Let $T(j, \mathbf{s})$ denote the truth value of the statement: there is a schedule for jobs $J_1, .., J_j$ for which $\mathbf{s}$ is the corresponding schedule configuration. The values of all the $T(j, \mathbf{s})$ can be viewed as being arranged in a table, and the crux of the approach lies in the following very simple procedure that can be used for filling in the table entries:

$$\begin{aligned}
T(1, \mathbf{v}) &= \begin{cases} \text{true if } \mathbf{v} \in V_1 \\ \text{false if } \mathbf{v} \notin V_1 \end{cases} \\
T(j, \mathbf{s}) &= \bigvee_{\mathbf{v} \in V_j : \mathbf{v} \leq \mathbf{s}} T(j - 1, \mathbf{s} - \mathbf{v}) \text{ for } j = 2, \ldots, k.
\end{aligned}$$

The reader should have no difficulty to bound the time to fill in the table entries by $O(mN)$, where N is the number of table entries. By the way we have rounded the values, the number of different completion times for each machine and different costs is bounded by $mk/\varepsilon + 1$. It follows that the total number of different schedule configurations is at most $(mk/\varepsilon + 1)^{m+1}$. By observing that it is sufficient to store the minimum cost value for each different m-dimensional vector $(t_1, ..., t_m)$ of completion times, we have $N = k(mk/\varepsilon + 1)^m$. Therefore, the total running time of the dynamic program is $O(km(mk/\varepsilon + 1)^m) = (\log m/\varepsilon)^{O(m^2)}$. By Lemma 3, and by using the described dynamic programming approach, we have the following result.

Theorem 4 *For the problem of minimizing the weighted sum of the cost and the makespan in scheduling n jobs on m unrelated machines (m fixed), there exists a fully polynomial time approximation scheme that runs in* $O(n) + (\log m/\varepsilon)^{O(m^2)}$ *time.*

3 Makespan Minimization in Job Shops

Let $\varepsilon > 0$ be an arbitrary small rational number, and let $m \geq 2$ and $\mu \geq 1$ be integral values. Throughout this section, the values ε, m and μ are considered to be constants and not part of the input. For simplicity, we assume that $1/\varepsilon$ is integral. The outline of the scheme is as follows.

Step 1 Round the processing times of the jobs (see Section 3.1).

Step 2 Merge jobs together of the same profile with small scale factor to obtain a constant number of jobs (see Section 3.2).

Step 3 Use an enumeration algorithm to compute the optimum solution for the job set obtained in Step 2.

Optimal Value Bounds. We begin by providing some lower and upper bounds of the minimum makespan. For any given instance of the job shop scheduling problem, the optimal value will be denoted as OPT. Let

$$d_j = \sum_{i=1}^{\mu} p_{ij}$$

be the total processing time of job $J_j \in \mathcal{J}$, and let

$$D = \sum_{J_j \in \mathcal{J}} d_j.$$

Clearly, $D/m \leq OPT$ and a schedule of length at most D can be trivially obtained by scheduling one job after the other. Then the following bounds hold: $\frac{D}{m} \leq OPT \leq D$. By dividing every processing time p_{ij} by D/m, we will assume, w.l.o.g., that $D/m = 1$ and

$$1 \leq OPT \leq m.$$

3.1 Rounding and Profiling Jobs

For each job J_j $(j = 1, \dots, n)$, let

$$\mathcal{N}_j = \left\{ O_{ij} : i = 1, \dots, \mu \text{ and } p_{ij} \le \frac{\varepsilon}{\mu m} d_j \right\}$$

denote the set of *negligible* operations.

Rounding the Input. The input data of any job J_j $(j = 1, \dots, n)$ is rounded as follows.

- Round the processing time of any negligible operation to zero, i.e. set $p_{ij} := 0$ for any $O_{ij} \in \mathcal{N}_j$.
- For any other operation $O_{ij} \notin \mathcal{N}_j$, round p_{ij} down to the nearest lower value of $\frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^h$, for some $h \in \mathbb{N}$.

Lemma 5 *Rounding the input produces* $1 + 2\varepsilon$ *loss.*

Proof. Clearly, by setting $p_{ij} := 0$, for every $O_{ij} \in \mathcal{N}_j$ and $j = 1, \dots, n$, the corresponding optimal makespan cannot be larger than OPT. Furthermore, if we replace the zero processing times of the negligible operations with the original processing times, we may potentially increase the solution value by at most

$$\sum_{O_{ij} \in \mathcal{N}} \frac{\varepsilon}{\mu m} d_j = \sum_{j=1}^{n} \sum_{O_{ij} \in \mathcal{N}_j} \frac{\varepsilon}{\mu m} d_j \le \varepsilon \frac{D}{m} = \varepsilon.$$

Any non negligible operation O_{ij} has processing times p_{ij} greater than $\frac{\varepsilon}{\mu m} d_j$. We rounded each p_{ij} down to the nearest lower value of $\frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^h$, where $h \in \mathbb{N}$ and $O_{ij} \notin \mathcal{N}_j$. By replacing these rounded values with the original ones we may potentially increase the makespan by at most a factor of $1 + \varepsilon$. ■

Job Profiles and Scale Factors. Consider the input instance after the rounding step. We define the *profile* of a job J_j to be a $(\mu \cdot m)$-tuple

$$< \Pi_{1,1,j}, \dots, \Pi_{1,m,j}, \Pi_{2,1,j}, \dots, \Pi_{2,m,j}, \dots, \Pi_{\mu,1,j}, \dots, \Pi_{\mu,m,j} >$$

such that the following holds for every operation O_{ij}.

- If $w = m_{ij}$ and $p_{ij} \ne 0$, then $p_{ij} = \frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^{\Pi_{i,w,j}}$.
- Else if $w = m_{ij}$ $(p_{ij} = 0)$ then $\Pi_{i,w,j} = -\infty$.
- Else $(w \ne m_{ij})$ we assume $\Pi_{i,w,j} = +\infty$.

Again, observe that any job J_j is completely defined by its job profile and the scale factor d_j.

Lemma 6 *The number of distinct profiles is at most* $\ell := m^{\mu} \cdot \big(2 + \log_{1+\varepsilon} \frac{\mu m}{\varepsilon}\big)^{\mu}$.

Proof. Every profile is determined by μm entries. Recall that every operation can be processed by one machine. It follows that all the different profiles can be generated by first deciding the machine for each operation (there are m^{μ} possibilities). Then, for the chosen machines, the entries can be either $-\infty$ or any nonnegative integer h with $\frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^h \leq d_j$. For the remaining (not chosen) machines the entries are equal to $+\infty$. ■

3.2 Grouping jobs

Consider the input instance rounded as described in the previous Section 3.1. Let $\delta := m(\frac{\varepsilon}{6\mu^4 m^2})^{m/\varepsilon}$, and partition the set of jobs into two subsets of *large* jobs $L = \{J_j : d_j > \delta\}$ and *small* jobs $S = \{J_j : d_j \leq \delta\}$ according to their scale factors. Again, we further partition the set of small jobs into subsets S_i of jobs having the same profile, for $i = 1, \ldots, \ell$. For each subset S_i, we group jobs from S_i as described for the unrelated parallel machines scheduling problem, but with the following difference: here grouping two jobs, J_a and J_b, means to form a composed job for which the processing time of the i-th operation is equal to the sum of the processing times of the i-th operations of J_a and J_b.

Lemma 7 *With* $1 + 2\varepsilon$ *loss, the number of jobs can be reduced in linear time to be at most* $\min\{n, \frac{2m}{\delta} + \ell\}$.

Proof. Consider the rounded input instance I' as described in Section 3.1 (before grouping small jobs). Assume, w.l.o.g., that there exists a solution SOL' for I' of value OPT'. It is sufficient to show that, by using the small jobs grouped as described previously, there exists a schedule of value $(1 + 2\varepsilon)OPT'$.

Partition the set L of large jobs into three subsets L_1, L_2, L_3 as follows. Set $\rho = \frac{\varepsilon}{6\mu^4 m^2}$ and let α denote a constant integer defined later such that

$$\alpha = 0, 1, \ldots, m/\varepsilon - 1.$$

Following a technique that was used in [20], L is partitioned into the following three subsets:

$$\begin{aligned} L_1 &= \{J_j : m\rho^{\alpha} < d_j\}, \\ L_2 &= \{J_j : m\rho^{\alpha+1} < d_j \leq m\rho^{\alpha}\}, \\ L_3 &= \{J_j : m\rho^{m/\varepsilon} < d_j \leq m\rho^{\alpha+1}\}. \end{aligned}$$

Note that each set size is bounded by a constant, and sets L_1, L_2, L_3 and S establish a partition of all jobs. The number α can be chosen such that

$$\sum_{J_j \in L_2} d_j \leq \varepsilon. \qquad (1)$$

This is done as follows. Starting from $\alpha := 0$, check each time whether the set L_2 corresponding to the current value of α satisfies inequality (1); if it is not the case, set $\alpha := \alpha + 1$ and repeat. Note that for different α-values, the corresponding L_2 sets are disjoint. The total length of all jobs is m, and so there exists a value $\alpha' \leq m/\varepsilon - 1$ for which the corresponding set L_2 satisfies inequality (1). We set $\alpha := \alpha'$.

In the following we consider an artificial situation that we use as a tool. Focus on solution SOL', remove from SOL' all the jobs except those from L_1. Clearly, this creates gaps in the resulting schedule σ. The starting and finishing times of the operations from L_1 divide the time into intervals: $t_1, t_2, \ldots, t_g$, where t_1 is the interval whose right boundary is the starting time of the first operation according to σ, and t_g is the interval with left boundary defined by the finishing time of the jobs from L_1. Furthermore, let l_v denote the length of interval t_v, $1 \leq v \leq g$. It follows that $OPT' = \sum_{s=1}^{g} l_s$. Note that the number g of intervals is bounded by $2\mu|L_1| + 1 \leq 3\mu/\rho^{\alpha}$. Let G be the set of pairs (v, i) such that no job from L_1 is processed in interval v and by machine i, for $v = 1, \ldots, g$ and $i = 1, \ldots, m$ (each pair (v, i) represents a gap of schedule σ).

Since the total length of jobs from L_2 is at most ε, we can get rid of these jobs by assuming that they are processed at the end of the schedule one after the other; this increases the schedule by at most ε.

Consider the problem of placing jobs from $L_3 \cup S$ into the gaps of σ such that the length of the resulting schedule is OPT'. In the following we first describe a linear program LP_1 which is a relaxation of this problem. Then we propose another linear program LP_2 which is a relaxation of LP_1. By using the solution of LP_2, we show that the jobs from L_3 and the grouped small jobs, can be scheduled within the gaps without increasing too much the length of the gaps. More precisely, we show that the total increase of the length can be bounded by ε, and the claim follows. The overall approach is similar to that used in the proof of Lemma 3.

We formulate LP_1 as follows. Consider the set S of (not grouped) small jobs. For each job $J_j \in S \cup L_3$ we use a set of decision variables $x_{j,\tau} \in [0, 1]$ for tuples $\tau = (\tau_1, \ldots, \tau_\mu) \in A$, where

$$A = \{(\tau_1, \ldots, \tau_\mu) | 1 \leq \tau_1 \leq \tau_2 \leq \ldots \leq \tau_\mu \leq g\}.$$

The meaning of these variables is that $x_{j,\tau}$ represents the fraction of job J_j whose operations are processed according to $\tau = (\tau_1, \ldots, \tau_\mu)$, i.e., the i-th operation is scheduled in interval τ_k for each $1 \leq k \leq \mu$. Note that by the way in which we numbered the operations, any tuple $(\tau_1, \ldots, \tau_\mu) \in A$ represents a valid ordering for the operations. Let the load $L_{v,h}$ on machine h in interval v be defined as the total processing time of operations from the jobs in $S \cup L_3$, that are executed by machine h during interval v, i.e.,

$$L_{v,h} = \sum_{J_j \in S \cup L_3} \sum_{\tau \in A} \sum\nolimits_{k=1,\ldots,\mu | \tau_k = v, m_{kj} = h} \; x_{j,\tau} p_{kj}.$$

Let us write the load $L_{v,h}$ as the sum of $L^3_{v,h} + L^S_{v,h}$, where $L^3_{v,h}$ is the load of the jobs from L_3, while $L^S_{v,h}$ is the load of jobs from S. By Lemma 5, we have

that
$$L^S_{v,h} = \sum_{\tau \in A} \sum\nolimits_{k=1,\ldots,\mu|\tau_k=v} \sum_{J_j \in S} x_{j,\tau} \frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^{\Pi_{k,h,j}}.$$

Then LP_1 is the following

$$\begin{array}{lll} (1) & L^3_{v,h} + L^S_{v,h} \leq \; l_v, & (v,h) \in G; \\ (2) & \sum_{\tau \in A} x_{j\tau} = \; 1, & J_j \in S \cup L_3; \\ (3) & x_{j\tau} \geq 0, & \tau \in A, \; J_j \in S \cup L_3. \end{array}$$

Constraint (1) ensures that the total length of operations assigned to gap (v,h) does not exceed the length of the interval, while constraint (2) ensures that job J_j is completely scheduled.

Let S_ϕ denote the set of small jobs having the same profile, where $\phi = 1,\ldots,\ell$. For each S_ϕ ($\phi = 1,\ldots,\ell$) we use a set of decision variables $y_{\phi\tau} \in [0,1]$ for tuples $\tau = (\tau_1,\ldots,\tau_\mu) \in A$. The meaning of these variables is that $y_{\phi\tau}$ represents the fraction of jobs from S_ϕ whose operations are processed according to $\tau = (\tau_1,\ldots,\tau_\mu)$, i.e., the i-th operation is scheduled in interval τ_k for each $1 \leq k \leq \mu$. Let

$$L^*_{v,h} = \sum_{\tau \in A} \sum\nolimits_{k=1,\ldots,\mu|\tau_k=v} \sum_{\phi=1}^{\ell} y_{\phi\tau} \sum_{J_j \in S_\phi} \frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^{\Pi_{k,h,j}}.$$

Then LP_2 is the following

$$\begin{array}{lll} (1) & L^3_{v,h} + L^*_{v,h} \leq \; t_s, & (v,h) \in G; \\ (2) & \sum_{\tau \in A} x_{j\tau} = \; 1, & J_j \in L_3; \\ (3) & \sum_{\tau \in A} y_{\phi\tau} = \; 1, & \phi = 1,\ldots,\ell; \\ (4) & x_{j\tau} \geq 0, & \tau \in A, \; J_j \in S; \\ (5) & y_{\phi\tau} \geq 0, & \tau \in A, \; \phi = 1,\ldots,\ell. \end{array}$$

By setting
$$y_{\phi\tau} = \frac{\sum_{J_j \in S_\phi} \; x_{j,\tau} d_j}{\sum_{J_j \in S_\phi} d_j}$$
it is easy to check that any feasible set of values $(x_{j,\tau})$ for LP_1 gives a feasible set of values $(y_{\phi\tau})$ for LP_2. Since by construction a feasible solution for LP_1 exists, a feasible solution for LP_2 exists as well. We show now that by using the optimal solution of LP_2 we can derive a schedule without increasing too much the makespan.

Let $y^*_{\phi\tau}$ ($x^*_{j\tau}$) denote the values of variables $y_{\phi\tau}$ ($x_{j\tau}$) according to the optimal solution of LP_2. For every positive value $y^*_{\phi\tau}$, schedule a subset $H_{\phi\tau}$ of grouped jobs from S_ϕ on machine i until either (a) the jobs from S_ϕ are exhausted or (b) the total fraction (i.e. $\frac{\sum_{J_j \in H_{\phi\tau}} d_j}{\sum_{J_j \in S_\phi} d_j}$) of jobs assigned to i is equal to $y^*_{\phi\tau}$ (if necessary fractionalize one job to use up $y^*_{\phi\tau}$ exactly). We repeat this for the not yet assigned grouped small jobs and for every positive value $y^*_{\phi\tau}$. Note that if $y^*_{\phi\tau}$ is not fractional, then the jobs from S_ϕ are not preempted by the previous algorithm. In general, the number of preempted jobs from S_ϕ is at most $f_\phi - 1$, where $f_\phi = \left|\left\{y^*_{\phi\tau} : y^*_{\phi\tau} > 0, \tau \in A\right\}\right|$. According to the optimal solution of LP_2, let us say that job $J_j \in L_3$ is preempted if the corresponding $(x^*_{j\tau})$-values are fractional. Let $f_j = \left|\left\{x^*_{j\tau} : x^*_{j\tau} > 0, \tau \in A\right\}\right|$, for $J_j \in L_3$, then we have that the number of preemptions of job $J_j \in L_3$ is $f_j - 1$. Therefore, the total number of preemptions is $f = \sum_{\phi=1}^{\ell}(f_\phi - 1) + \sum_{J_j \in L_3}(f_j - 1)$, and this gives also an upper bound on the number of preempted jobs.

Now remove all the preempted jobs from $S \cup L_3$, and schedule these set of jobs at the end of the schedule, one after the other. Since every job from S has a smaller total processing time than any job from L_3, we can bound the total increase of the schedule length by $\Delta = f \cdot m\rho^{\alpha+1}$. A basic feasible solution of LP_2 has the property that the number of positive variables is at most the number of rows in the constraint matrix, $mg + \ell + |L_3|$, therefore $f \le mg \le 3m\mu/\rho^\alpha$, and $\Delta = f \cdot m\rho^{\alpha+1} \le 3m^2\mu\rho$. By the previous algorithm, we have assigned all the jobs from $S \cup L_3$ to gaps with a total increase of the schedule length of $3m^2\mu\rho$. Now we consider the problem of schedule jobs from $S \cup L_3$ within each interval. This is simply a smaller instance of the job shop problem, and by using Sevastianov's algorithm [21] it is possible to find a feasible schedule for each interval t_v of length at most $l_v + \mu^3 m \cdot m\rho^{\alpha+1}$; this increases the total length by at most $\mu^3 mg \cdot m\rho^{\alpha+1} \le 3m^2\mu^4\rho$. Therefore, the total increase is $3m^2\mu\rho + 3m^2\mu^4\rho \le 6m^2\mu^4\rho$, and by setting $\rho = \frac{\varepsilon}{6m^2\mu^4}$ the claim follows. ■

By the previous lemmas a $(1 + O(\varepsilon))$-approximate schedule can be obtained by finding the optimal schedule for the reduced set of jobs, a task that can be performed in constant time.

Theorem 8 *There exists a linear time PTAS for the job shop scheduling problem whose multiplicative constant hidden in the $O(n)$ running time is reasonably small and does not depend on the error ε, whereas the additive constant is exponential in m, μ and $1/\varepsilon$.*

4 Conclusion

In this paper we have proposed a new grouping technique which improves the running times of several known approximation schemes. Interestingly, the resulting approximation schemes are much simpler than the previous ones. This new technique allowed us also to avoid the quite involved linear programming technique by Grigoriadis and Khachiyan [5] as well as the rounding techniques [10, 14], and the vector summation algorithm by Sevastianov [21] (that were used as key ingredients in the approximation scheme for the job shop scheduling problem). Clearly, the technique can be also used as a preprocessing step in other algorithms (like branch and bound or cutting plane algorithms). Finally, we expect that the proposed idea will be also useful for other scheduling problems.

References

[1] A. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. *Algorithmica*, 32:247–261, 2002.

[2] J. Chen and A. Miranda. A polynomial time approximation scheme for general multiprocessor job scheduling. *SIAM Journal on Computing*, 31:1–17, 2001.

[3] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[4] L. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk. Better approximation guarantees for job-shop scheduling. *SIAM Journal on Discrete Mathematics*, 14(1):67–92, 2001.

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

[6] L. Hall. Approximability of flow shop scheduling. *Mathematical Programming*, 82:175–190, 1998.

[7] L. Hall and D. Shmoys. Approximation algorithms for constrained scheduling problems. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 134–139, 1989.

[8] L. Hall and D. Shmoys. Near-optimal sequencing with precedence constraints. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, pages 249–260. University of Waterloo Press, 1990.

[9] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327, 1976.

[10] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Math. Oper. Res.*, 26(2):324–338, 2001.

[11] K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.

[12] K. Jansen and L. Porkolab. Polynomial time approximation schemes for general multiprocessor job shop scheduling. *J. Algorithms*, 45(2):167–191, 2002.

[13] K. Jansen, R. Solis-Oba, and M. Sviridenko. Makespan minimization in job shops: a polynomial time approximation scheme. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC 99)*, pages 394–399, 1999.

[14] K. Jansen, R. Solis-Oba, and M. Sviridenko. Makespan minimization in job shops: A linear time approximation scheme. *SIAM J. Discrete Math.*, 16(2):288–300, 2003.

[15] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbook in Operations Research and Management Science*, 4:445–522, 1993.

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

[17] M. Mastrolilli. Efficient approximation schemes for scheduling problems with release dates and delivery times. *Journal of Scheduling*, 6:521–531, 2003.

[18] M. Mastrolilli. A linear time approximation schemes for the single machine scheduling problem with controllable processing times. *Journal of Algorithms*, 59:37–52, 2006.

[19] P. Schuurman and G. Woeginger. Approximation schemes - a tutorial. In R. Moehring, C. Potts, A. Schulz, G. Woeginger, and L. Wolsey, editors, *Lectures on Scheduling.* To appear.

[20] S. Sevastianov and G. J. Woeginger. Makespan minimization in open shops: a polynomial time approximation scheme. *Mathematical Programming*, 82:191–198, 1998.

[21] S. V. Sevastianov. On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics*, 55:59–82, 1994.

[22] D. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23:617–632, 1994.

[23] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[24] D. Williamson, L. Hall, J. Hoogeveen, C. Hurkens, J. Lenstra, S. Sevastianov, and D. Shmoys. Short shop schedules. *Operations Research*, 45:288–294, 1997.