Makespan minimization in job shops: a linear time approximation scheme *

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

Roberto Solis-Oba ‡
MPII Saarbrücken
Im Stadtwald
66123 Saarbrücken
Germany
solis@mpi-sb.mpg.de

Maxim Sviridenko
Sobolev Inst. of Mathematics
pr. Koptyuga 4
630090 Novosibirsk
Russia
svir@math.nsc.ru

Abstract

In this paper we present a linear time approximation scheme for the job shop scheduling problem with fixed number of machines and fixed number of operations per job. This improves on the previously best $2+\epsilon$, $\epsilon>0$, approximation algorithm for the problem by Shmoys, Stein, and Wein. Our approximation scheme is very general and it can be extended to the case of job shop scheduling problems with release and delivery times, multi-stage job shops, dag job shops, and preemptive variants of these problems.

1 Introduction

In the job shop scheduling problem there is a set $J=\{J_1,\ldots,J_n\}$ of n jobs that must be processed on a given set $M=\{M_1,\ldots,M_m\}$ of m machines. Each job J_j consists of a sequence of μ_j operations $O_{1j},\ldots,O_{\mu_j j}$ that need to be processed in this order. Operation O_{ij} must be processed without interruption on machine $M_{\pi_{ij}}$, during p_{ij} time units. A 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 a given schedule, let C_{ij} be the completion time of operation O_{ij}. The objective is to find a schedule that minimizes the maximum completion time, $C_{max}=\max_{ij} C_{ij}$. The value of C_{max} is also called the *makespan* or the *length* of the schedule.

For a given instance of the job shop scheduling problem, the value of the optimum makespan will be denoted as C^*_{max}. Let $P_t=\sum_{\pi_{ij}=t} p_{ij}$ be the total processing time of operations assigned to machine M_t. We call P_t the *load* of machine M_t. Let $P_{max}=\max\{P_1,\ldots,P_m\}$, be the maximum machine load. Clearly, $P_{max}\le C^*_{max}$. Let $l_j=\sum_{i=1}^{\mu_j} p_{ij}$ be the length of job J_j and let $\mu=\max_j\mu_j$ be the maximum number of operations in any job. We define $p_{max}=\max_{ij} p_{ij}$ to be the maximum operation length.

The job shop scheduling problem is considered to be one of the most difficult problems in combinatorial optimization, both from the theoretical and the practical points of view. Even very constrained versions of the problem are strongly NP-hard (see e.g. the survey paper by

*Preliminary versions of this paper appeared at the Proceedings of the 31th ACM Symposium on Theory of Computing (STOC'99) and the Proceedings of the Second Workshop on Approximation Algorithms (APPROX'99).

†This author was supported in part by the Swiss Office Fédéral de l'éducation et de la Science project no. 97.0315 titled "Platform".

‡This author was supported in part by EU ESPRIT LTR Project No. 20244 (ALCOM-IT).

Lawler et al. [6]). Two other widely studied shop scheduling problems are the flow shop and the open shop problems. In the flow shop problem every job has exactly one operation per machine, and the order of execution for the operations is the same for all jobs. In the open shop problem every job has also one operation per machine, but there is no specified order for the execution of the operations of a job. Williamson et al. [18] proved that for any $\rho < 5/4$, the existence of a ρ-approximation algorithm for any of the above shop scheduling problems when the number of machines is part of the input would imply that $P = NP$. This result holds even if every operation has integer processing time and each job has at most 4 operations.

Many papers about shop scheduling problems have been written recently. Several of them are based on the seminal work by Leighton, Maggs and Rao [7], on the acyclic job shop problem with unit length operations. In this problem every job has exactly one operation per machine. Their main result was to show that this problem always has a solution of length $O(P_{max} + l_{max})$, where $l_{max} = \max_j l_j$ is the maximum job length. This is not an algorithmic result since it relies on a non-constructive probabilistic argument (for a constructive version see [8]).

Shmoys, Stein and Wein [17] described an approximation algorithm for the job shop scheduling problem with $O(\log^2(m\mu))$ performance guarantee. This algorithm was later improved by Goldberg et al. [1] who designed an approximation algorithm with performance guarantee of $O(\log^2(m\mu)/(\log\log(m\mu))^2)$. When the number of machines m and the maximum number μ of operations per job are constant, Shmoys et al. [17] designed an approximation algorithm with performance guarantee $(2 + \varepsilon)$, for any fixed value $\varepsilon > 0$. Following the three-field notation scheme [6], we denote this problem as $Jm|op \leq \mu|C_{max}$.

There are only few theoretical results known for the preemptive version of the job shop scheduling problem. It is known that this problem is strongly NP-hard even in the case when there are 3 machines and every job has at most 3 operations (see survey paper [6]). On the positive side Sevastianov and Woeginger [14] designed a 3/2-approximation algorithm for the problem when the number of machines is 2.

A polynomial time approximation scheme (PTAS) for a (minimization) optimization problem is an algorithm that given any constant value $\varepsilon > 0$ finds in polynomial time a solution of value no larger than $1 + \varepsilon$ times the value of an optimum solution. A fully polynomial time approximation scheme is an approximation scheme that runs in time polynomial in the size of the input and $1/\varepsilon$.

When the number m of machines is fixed, there exist polynomial time approximation schemes for the flow shop [3] and the open shop [15] problems. But the $2 + \varepsilon$ approximation algorithm of Shmoys et al. [17] was the previously best known algorithm for the job shop problem with m and μ fixed.

In this work we describe a linear time approximation scheme for the job shop scheduling problem when m and μ are fixed. Our work is strongly based on ideas contained in some of the aforementioned papers. We use the idea by Sevastianov and Woeginger [15] of partitioning the set of jobs into three sets: big, small, and tiny jobs. The sets of big and small jobs have constant size. We construct all relative schedules for the big jobs, and since the number of big jobs is constant, the total number of their relative schedules is also constant. In any relative schedule for the big jobs, the starting and completion times of the jobs define a set of time intervals, into which we have to schedule the small and tiny jobs. We use linear programming to find a "compact" assignment of small and tiny jobs to these time intervals. Then we use a novel rounding technique to reduce the number of jobs that receive fractional assignments to a constant. Since only small and tiny jobs receive fractional assignments we can use a very simple rounding procedure for them to get a non-preemptive schedule without increasing the length of the solution by too much. This solution is not feasible, though, since in each interval there might be conflicts among the small and tiny jobs.

We find a feasible schedule for the small and tiny jobs in each time interval by using an algorithm by Sevastianov [12] (for a detailed presentation, in English, of the algorithm see [16]). Sevastianov's algorithm runs in $O((\mu mn)^2)$ time and for any instance of the job shop scheduling problem it finds a schedule of length at most $P_{max} + \varphi(m, \mu)p_{max}$, where $\varphi(m, \mu) = (m\mu^2 + 2\mu - 1)(\mu - 1) = O(m\mu^3)$. (See also [13] for survey and historical overview of geometric methods used in the design and analysis of approximation algorithms with absolute performance guarantee for scheduling problems.) By selecting properly the sets of big, small, and tiny jobs we can prove that the total length of the schedule computed by the algorithm is at most $1 + \epsilon$ times the length of an optimum solution.

All steps of the algorithm can be performed in linear time, except two of them: solving the linear program and running Sevastianov's algorithm. Since we do not solve exactly the job shop scheduling problem, we do not need to solve exactly the linear program, an approximate solution would suffice. We use an algorithm of Grigoriadis and Khachiyan [2] to find in linear time a $1+\varepsilon$ approximation to the solution of the linear program. Then we use an elegant idea of merging certain subsets of jobs together to form larger jobs to decrease the running time of Sevastianov's algorithm to $O(n)$. The overall complexity of our algorithm is linear in the number of jobs, but it is not polynomial in $1/\varepsilon$. This is not surprising since the problem is strongly NP-hard [6] and therefore no fully polynomial time approximation scheme for the problem can exist unless P=NP.

Our approach can be used to design linear time approximation schemes for more general problems like the so-called dag shop problem [12, 13, 17], in which only a partial order is specified for the ordering of execution of the operations of a job. This problem includes as a special case the open shop problem. Since the flow shop problem is a special case of the job shop problem, our result generalizes the results of Hall [3] and Sevastianov and Woeginger [15] in the sense that we prove the existence of PTAS for these two latter problems.

Our approximation scheme can be generalized also to the following problems when m and μ are fixed: multi-stage job shop, assembly scheduling problem, and job shop problems with release and delivery times. It is also possible to modify the schemes to design approximation algorithms for the preemptive versions of these problems.

The rest of the paper is organized in the following way. In Section 2 we describe a polynomial time approximation scheme for the non-preemptive job shop scheduling problem. Then in Section 3 we show how to reduce the time complexity of the algorithm to $O(n)$. In Section 4 we design a linear time approximation scheme for the preemptive version of the problem. Finally, in Section 5 we show how to handle other shop scheduling problems.

2 PTAS for the Job Shop Scheduling Problem

2.1 Restricted Job Shop Problem

Let $\varepsilon > 0$ be constant value. Let $m \geq 2$ and $\mu \geq 1$ be the number of machines and maximum number of operations per job, respectively. We assume that the values of ε, μ, and m are fixed and not part of the input. We partition the set of jobs into three subsets as follows. Let α be a real number such that

$$\varepsilon^{\lceil m/\varepsilon \rceil} \leq \alpha \leq \varepsilon.$$

We define three sets of jobs:

$$B = \{J_j | l_j \geq \alpha P_{max}\},$$
$$S = \{J_j | \alpha\varepsilon P_{max} < l_j < \alpha P_{max}\}, \text{ and}$$
$$T = \{J_j | l_j \leq \alpha\varepsilon P_{max}\}.$$

The jobs in B are called *big jobs*, the jobs in S are called *small jobs*, and the jobs in T are called *tiny jobs*. For the operations of big, small and tiny jobs we use a similar notation: the operations of big jobs are called *big operations*, while operations of small and tiny jobs are called *small* and *tiny operations*, respectively, independently of their actual sizes. The number of big jobs is at most $mP_{max}/(\alpha P_{max})$, and thus the size of B is bounded by a constant depending only on ε and m:

$$|B| \leq m/\alpha \leq m\varepsilon^{-m/\varepsilon}.$$

Sevastianov and Woeginger [15] show that the number α can be chosen so that

$$\sum_{J_j \in S} l_j \leq \varepsilon P_{max}. \tag{1}$$

This is done as follows. Define a sequence of numbers α_i, where i is a nonnegative integer, by $\alpha_i = \varepsilon^i$ and consider the sets S_i of small jobs with respect to α_i. Note that two sets S_i and S_j are disjoint for $i \neq j$. Since the total length of all jobs is at most mP_{max}, then there exists a value $k \leq m/\varepsilon$ for which $S = S_k$ satisfies inequality (1). We set $\alpha = \alpha_k$.

Our algorithm distinguishes only two kinds of jobs, B and $J \setminus B$. Differences between S and T will be used only in the analysis of the algorithm.

It is not difficult to see that mP_{max} is an upper bound on the length of an optimal schedule. So we partition the time interval from 0 to mP_{max} into $\lceil m/(\alpha\varepsilon)\rceil$ equal intervals of length at most $\alpha\varepsilon P_{max}$. These intervals are called *intervals of the first type*. We consider only schedules in which every big operation starts processing at the beginning of some interval of the first type. This restriction does not increase the length of the optimal schedule considerably. Indeed, let us consider the first big operation in an optimal schedule that does not start at the beginning of some interval of the first type. We can simply shift this big operation to the right, so that it starts at the beginning of the next interval. All operations starting after this big operation are also shifted to the right by the same length. Then we do the same thing with the remaining big operations. The overall increase in the length of the optimum schedule is bounded by $\mu|B|\alpha\varepsilon P_{max} \leq \mu m\varepsilon P_{max} \leq \mu m\varepsilon C^*_{max}$. Let $\widetilde{C}^*_{max}$ be the length of an optimal schedule in which the big operations start at the beginning of some interval of the first type. As was noted above $\widetilde{C}^*_{max} \leq (1 + \mu m\varepsilon)C^*_{max}$.

In the rest of this section we consider only this restricted job shop scheduling problem in which every big operation must start at the beginning of some interval of the first type. Since the number of big operations is constant and since the number of intervals of the first type is constant, we conclude that the number of different schedules with fixed starting times for the big operations is constant too. For each schedule for the big operations we assign the starting times for the small and tiny operations within some interval of the second type by solving a linear program as described below.

2.2 Scheduling Small and Tiny Operations

Fix some feasible restricted schedule for the big operations within the time interval $[0, (1 + \mu m\varepsilon)mP_{max}]$. Let S_{ij} and C_{ij} be the starting and completion times of big operation O_{ij}, respectively. Let $A = \{a_k | a_k = S_{ij} \text{ or } a_k = C_{ij} \text{ for some big operation } O_{ij}\}$, be the set of starting and completion times of big operations. Notice that

$$|A| \leq 2\mu|B| \leq 2\mu m/\alpha. \tag{2}$$

Assume that the elements in A are indexed so that $a_1 \leq a_2 \leq \ldots \leq a_{|A|}$. We define two new elements $a_0 = 0$ and $a_{|A|+1} = C \in [a_{|A|}, (1+\mu m\varepsilon)mP_{max}]$ (the exact value of C will be specified later), and partition the time interval from 0 to C into $|A|+1$ intervals $[a_k, a_{k+1}), k = 0, \ldots, |A|$. We call these intervals, *intervals of the second type.* Let Δ_k be the length of the kth interval, i.e. $\Delta_k = a_{k+1} - a_k$. Define $\Delta_{tk} = 0$ if some big operation is processed in interval k on machine M_t, and $\Delta_{tk} = \Delta_k$ otherwise. So Δ_{tk} is the amount of time that machine M_t can be used during interval k to process small and tiny operations.

For every job $J_j \in T \cup S$ let

$$K_j = \{K = (k_1, k_2, \ldots, k_{\mu_j}) \in Z^{\mu_j} \mid 0 \leq k_1 \leq k_2 \leq \ldots \leq k_{\mu_j} \leq |A|\}$$

be the set of all feasible assignments of operations of job J_j to intervals of the second type. A tuple $(k_1, k_2, \ldots, k_{\mu_j}) \in K_j$ means that the ith operation of job J_j, $1 \leq i \leq \mu_j$, is processed in interval (of the second type) k_i.

Now we use a linear program to schedule small and tiny operations. We define variables x_{jK}, $K = (k_1, k_2, \ldots, k_{\mu_j}) \in K_j$, $J_j \in T \cup S$, where x_{jK} has value f, $0 \leq f \leq 1$, if and only if a fraction f of the first operation of job J_j is processed in interval k_1 on machine $M_{\pi_{1j}}$, a fraction f of the second operation is processed in interval k_2 on $M_{\pi_{2j}}$, and so on.The linear program is the following. (We assume that we have already chosen the value of the length C of the schedule, so we are interested only in knowing whether the jobs can be scheduled within the time interval $[0, C]$; we show below how to choose C.)

$$\sum_{K \in K_j} x_{jK} = 1, \quad J_j \in T \cup S, \tag{3}$$

$$\sum_{J_j \in T \cup S} \sum_{K \in K_j} \sum_{k_i = k, \pi_{ij} = t} p_{ij}\, x_{jK} \leq \Delta_{tk}, \quad t = 1, \ldots, m, k = 0, \ldots, |A| \tag{4}$$

$$x_{jK} \geq 0, \quad K \in K_j, J_j \in T \cup S \tag{5}$$

Constraint 3 ensures that job J_j is completely scheduled, while constraint (4) means that the total length of operations assigned to interval k on machine M_t does not exceed the length of the interval.

Lemma 1 *For $C = \widetilde{C}^*_{max}$, the linear program (3)-(5) has a feasible solution for some restricted schedule for the big operations.*

Proof: Consider an optimum schedule S^* of the restricted job shop problem. Assume that some small or tiny operation O_{ij} is processed in consecutive time intervals $b_{ij}, b_{ij}+1, \ldots, e_{ij}$ on machine $M_{\pi_{ij}}$, where b_{ij} might be equal to e_{ij} (corresponding to the case when the operation is completely scheduled in a single interval). Let $f_{ij}(k)$ be the fraction of operation O_{ij} that is scheduled in interval k.

We assign values to the variables x_{jK}, $K = (b_{1j}, b_{2j}, \ldots, b_{\mu_j, j})$, as follows. Set $x_{jK} = f$, where $f = \min\{f_{ij}(b_{ij}) \mid 1 \leq i \leq \mu_j\}$ is the smallest fraction of an operation of job J_j that is scheduled in the first interval assigned to it in S^*. Next we assign values to the other variables

x_{jK} to cover the remaining $1-f$ fraction of each operation. To do this, for every operation O_{ij}, we make $f_{ij}(b_{ij}) = f_{ij}(b_{ij}) - f$. Clearly for at least one operation O_{ij} the new value of $f_{ij}(b_{ij})$ will be set to zero. For those operations with $f_{ij}(b_{ij}) = 0$ we set $b_{ij} = b_{ij} + 1$ since the first interval for the rest of the operation O_{ij} is interval $b_{ij} + 1$. Then we assign value to the new variable $x_{jK}, K = (b_{1j}, b_{2j}, \ldots, b_{\mu_j,j})$ as above, and repeat the process until $f = 0$. Note that each iteration of this process assigns a value to a different variable x_{jK} since from one iteration to the next at least one interval b_{ij} is redefined. This assignment of values to variables x_{jK} is a feasible solution for the linear program. □

Let C_{min} be the smallest value C such that linear program (3)-(5) has a feasible solution for some schedule for the big operations. By Lemma 1, for $C = \widetilde{C}^*_{max}$ the linear program has a feasible solution, so $C_{min} \leq \widetilde{C}^*_{max}$. For any fixed value $\delta \geq 0$, we can find a value C satisfying $C_{min} \leq C \leq C_{min} + \delta P_{max}$, by using binary search. Thus we must solve linear program (3)-(5) at most $\lceil \log_2((1+\mu m\varepsilon)m/\delta)\rceil$ times. Since C_{min} is a lower bound for $\widetilde{C}^*_{max}$, then $C \leq \widetilde{C}^*_{max} + \delta P_{max} \leq C^*_{max} + 2\mu m\varepsilon P_{max}$, for $\delta = \mu m\varepsilon$.

The linear program has $|T \cup S| + m(|A|+1)$ constraints and $|T \cup S|(|A|+1)^{\mu}$ variables. Therefore, a basic feasible solution is guaranteed to have at most $|T \cup S| + m(|A|+1)$ nonzero variables. This solution can have at most $m(|A|+1)$ fractional variables, since by constraint (3) every job must have at least one positive variable associated with it. (This kind of argument was first made and exploited by Potts [10] in the context of parallel machine scheduling.) We now describe a simple rounding procedure to obtain an integral (and possibly infeasible) solution for the linear program. If job J_j has more than one nonzero variable associated with it we set one of them to 1 and the others to 0 in an arbitrary manner. In this solution the small and tiny operations have a unique assignment to intervals of the second type. Let $D(k)$ be the total processing time of small and tiny operations assigned to interval k such that the jobs corresponding to these operations received fractional assignments from the linear program. Notice, that by (1) and (2), $\sum_{k=0}^{|A|} D(k) \leq m(|A|+1)\alpha\varepsilon P_{max} + \sum_{J_j \in S} l_j = O(m^2\mu)\varepsilon P_{max}$. Thus this rounding procedure only slightly increases the length of the solution.

2.3 Finding a Feasible Schedule

Consider some interval $[a_k, a_{k+1})$ of the second type. Let $p_{max}(k)$ be the length of the longest small or tiny operation assigned to this interval. By construction, in the rounded solution the total length of operations assigned to this interval on each machine is at most $a_{k+1} - a_k + D(k)$. We consider now the problem of scheduling the small and tiny operations within the interval. This is simply a smaller instance of the job shop problem, and by using Sevastianov's algorithm [12] it is possible to find a feasible schedule of length at most $a_{k+1} - a_k + D(k) + O(m\mu^3)p_{max}(k)$.

Note that we can schedule the small and tiny operations in each interval $[a_k, a_{k+1})$ independently from the operations in any other interval $[a_i, a_{i+1})$. Moreover, if we add $D(k) + O(m\mu^3)p_{max}(k)$ to the length of each interval, the union of the schedules for these intervals yields a feasible solution for the original constrained job shop problem (where we assume that all big operations start at the beginning of some interval of the first type). The makespan of this schedule is at most

$$C + \sum_{k=0}^{|A|}\Big(D(k) + O(m\mu^3)p_{max}(k)\Big) \leq C + O(m^2\mu)\varepsilon P_{max} + O(m\mu^3)\left(\sum_{J_j \in S} l_j + (|A|+1)\alpha\varepsilon P_{max}\right) \leq$$

$$C + O(m^2\mu^4)\varepsilon P_{max} \leq C^*_{max} + O(m^2\mu^4)\varepsilon P_{max}.$$

Since m and μ are both constants and ε is an arbitrary rational number then our algorithm can find in polynomial time a solution of length at most $1+\epsilon$ times the optimum for any value $\epsilon > 0$.

Theorem 1 *The above algorithm is a polynomial time approximation scheme for the job shop scheduling problem when m and μ are fixed.*

3 Speed Up to Linear Time

In the PTAS that we have just described there are two steps that seem to require more than linear time: finding a basic feasible solution for the linear program and running Sevastianov's algorithm. In the next two sections we show how to perform these steps in linear time.

Since we do not solve exactly the job shop scheduling problem, we do not need to solve the linear program exactly either. An approximate solution would suffice. To find an approximate solution for the linear program, let us replace condition (4) of the linear program by the following:

$$\frac{1}{\Delta_{tk}} \sum_{J_j \in T \cup S} \sum_{K \in K_j} \sum_{k_i = k, \pi_{ij} = t} p_{ij}\, x_{jK} \leq \lambda, \quad t = 1, \ldots, m, k = 0, \ldots, |A|, \Delta_{tk} \neq 0, \tag{6}$$

where λ is a non-negative variable. If for some pair t, k, $\Delta_{tk} = 0$ we can remove the condition from the linear program and set the corresponding variables x_{jK} to zero. We call this new linear program LP$'$.

This linear program has a special *block angular* structure that we now describe (for a survey see [2, 9]). For each small and tiny job J_j let x_{jK_J} be the $(|A|+1)^{\mu_j}$-dimensional vector whose components are the different variables x_{jK} of job J_j. For job J_j we define the set $B_j = \{x_{jK_j} \mid$ conditions (3) and (5) are satisfied$\}$. This set is a *simplex* of constant dimension. Linear inequalities (6) form a set of so-called *coupling constraints*. Note that the left hand side of each inequality (6) is non-negative. A solution for the linear program LP$'$ is a set of points that belong to the above simplicies and that satisfies the coupling constraints.

The Logarithmic Potential Price Directive Decomposition Method developed by Grigoriadis and Khachiyan [2] can be used to either determine that the linear program LP$'$ is infeasible, or to find a $(1+\varepsilon)$-approximation to the smallest value λ for which LP$'$ has a feasible solution. This procedure runs in linear time ([2], Theorem 3). Since by choosing $C = \widetilde{C}^*_{max}$, LP$'$ has a feasible solution for $\lambda = 1$, then we can find in linear time a solution of the linear program in which the length of each interval Δ_{tk} is enlarged to $\Delta_{tk}(1+\varepsilon)$. The length of this solution is no more than $(1+\varepsilon)$ times larger than the length of a solution for the original linear program.

The Logarithmic Potential Price Directive Decomposition Method finds a feasible solution for the linear program, but not necessarily a basic feasible solution. So we need a linear time rounding procedure which given a feasible solution of the linear program LP$'$, finds a solution with at most $O(|A|)$ fractional variables where the hidden constants depend on m and μ only.

3.1 Rounding Procedure

In this subsection we show how to round any feasible solution for the linear program (3),(5),(6) to get a new feasible solution in which all but a constant number of variables x_{jK} have value 0 or 1. Moreover we can do this rounding procedure in linear time.

First we write the linear program in matrix form as $Bx = b$, $x \geq 0$, where B is the constraint matrix. The key observation that allows us to perform the rounding in linear time is to note that matrix B is sparse. We will show that there exists a constant size subset B' of columns of

B in which the number of non-zero rows is smaller than the number of columns. The non-zero entries of B' induce a singular matrix of constant size, so we can find a non-zero vector y in the null space of this matrix, i.e. $B'y = 0$.

Let $\delta > 0$ be the smallest value such that some component of the vector $x + \delta y$ is either zero or one (if the dimension of y is smaller than the dimension of x we augment it by adding an appropriate number of zero entries). Note that the vector $x + \delta y$ is a feasible solution of the linear program. Let $x^0$and x^1 be respectively the zero and one components of vector $x + \delta y$. We update the linear program by making $x = x + \delta y$ and then removing from x all variables in x^0 and x^1 and all columns of B corresponding to such variables. If $x^1 \neq \emptyset$ then vector b is set to $b - \sum_{i \in x^1} B[*, i]$, where $B[*, i]$ is the column of B corresponding to variable i.

This process rounds the value of at least one variable x_{jK} to either 0 or 1. We note that the value of δ can be found in constant time since vector y has constant number of non-zero coordinates. We repeat this process until only a constant number of variables x_{jK} have fractional values. Since there is a linear number of these variables then the overall time is linear.

Now we describe the rounding algorithm in more detail. Let us assume that the columns of B are indexed so that the columns corresponding to variables $x_{jK}, K \in K_j$ for each job J_j appear in adjacent positions. We might assume that at all times during the rounding procedure each job J_j has associated at least two columns in B. This assumption can be made since if job J_j has only one associated column, then the corresponding variable x_{jK} must have value either zero or one. Let B' be the set formed by the first $2m(|A| + 1) + 2$ columns of B. Note that at most $2m(|A| + 1) + 1$ rows of B' have non-zero entries. To see this observe that at most $m(|A| + 1) + 1$ of these entries come from constraint (3) and by the above assumption on the number of columns for each job, while at most $m(|A| + 1)$ non-zero entries come from constraint (6).

To avoid introducing more notation let B' be the matrix induced by the non-zero rows. Since B' has at most $2m(|A| + 1) + 1$ rows and exactly $2m(|A| + 1) + 2$ columns then B' is singular and hence there exists at least one non-zero vector y such that $B'y = 0$. Since the size of B' is constant, vector y can be found in constant time by using simple linear algebra.

After updating x, B, and b as described above, the procedure is repeated. This is done until there are at most $2m(|A|+1)+1$ columns in B corresponding to variables x_{jK} and hence at most $m(|A|+1)+1$ variables x_{jK} have value different from 0 and 1. Let $\mathcal{F}$ be the set of jobs that receive fractional assignments. For each job in $\mathcal{F}$ we arbitrarily choose one of its non-zero variables and set it to 1 while we set all other variables to 0. As before let $D(k)$ be the total processing time of jobs from $\mathcal{F}$ that were assigned to interval k. Then $\sum_{k=0}^{|A|} D(k) = O(m^2\mu)\varepsilon P_{max}$, and so we can find in linear time an integer solution for the linear program of length arbitrarily close to the length of an optimum schedule for the jobs J.

3.2 Merging Trick

Consider the instance of the job shop scheduling problem defined by the small and tiny jobs placed in interval k by the linear program. Sevastianov's algorithm finds in $O(n^2\mu^2m^2)$ time a schedule of length at most $a_{k+1} - a_k + D(k) + O(m\mu^3)p_{max}(k)$, where $p_{max}(k)$ is the length of the largest operation in interval k. For a job J_j let $(m_{1j}, m_{2j}, \ldots, m_{\mu j})$ be a vector that describes the machines on which its operations must be performed. Let us partition the set of jobs J into m^μ groups $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_{m^\mu}$ such that all jobs in some group $\mathcal{J}_i$ have the same machine vector and jobs from different groups have different machine vectors. Consider the jobs in one of the groups $\mathcal{J}_i$. Let J_j and J_h be two jobs from $\mathcal{J}_i$ such that each one of them has execution time for its operations smaller than $\alpha\varepsilon P_{max}/2$. We "glue" together these two jobs to form a composed job in 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_j and J_h. We repeat this process until at most one job from $\mathcal{J}_i$ has processing time smaller than $\alpha\varepsilon P_{max}/2$. The same procedure is performed in all other groups $\mathcal{J}_j$. At the end of this process, each one of the composed jobs has at most μ operations. The total number of composed jobs is at most $m^\mu + \lceil\frac{2m}{\alpha\varepsilon}\rceil$, and all operations in interval k have processing times smaller than $\max\{p_{max}(k), \alpha\varepsilon P_{max}\}$. Note that this merging procedure runs in linear time, and that a feasible schedule for the original jobs can be easily obtained from a feasible schedule for the composed jobs.

We run Sevastianov's algorithm on this set of composed jobs to get a schedule of length $a_{k+1} - a_k + D(k) + O(m\mu^3)\max\{p_{max}(k), \alpha\varepsilon P_{max}\}$. The time needed to get this schedule is $O((m^\mu + \frac{2m}{\alpha\varepsilon})^2\mu^2 m^2)$. So Sevastianov's algorithm needs only constant time plus linear preprocessing time. Notice also that the analysis in Subsection 2.3 with minor changes holds also for this case.

Theorem 2 *The algorithm described above is a linear time approximation scheme for the job shop scheduling problem when m and μ are fixed.*

4 Preemptive Job Shop Scheduling Problem

In this section we describe a PTAS for the preemptive version of the job shop scheduling problem when m and μ are fixed. As in the non-preemptive case we divide the set of jobs J into big jobs B, small jobs S, and tiny jobs T. The sets are chosen as in the non-preemptive version. We define a restricted schedule for the big jobs by stating for each big operation a set of consecutive intervals of the first type where the operation can be scheduled. Since there is a constant number of big jobs, there is also a constant number of restricted schedules. Fix one of the restricted schedules and define intervals of the second type as before.

An operation O_{ij} of a big job is scheduled in consecutive intervals of the second type $[a_k, a_{k+1}), \ldots, [a_{k+t-1}, a_{k+t})$, where a_k is the starting time and a_{k+t} is the completion time of O_{ij}. Any fraction (possible equal to zero) of the operation might be scheduled in any one of these intervals. However, and this is crucial for the analysis, in each interval of the second type there is at most one operation from any big job. This condition is easily ensured by defining disjoint intervals for the different operations of a big job.

As for the non-preemptive case, for every small and tiny job J_j we let

$$K_j = \{K = (k_1, k_2, \ldots, k_{\mu_j}) \in Z^{\mu_j} \mid 0 \le k_1 \le k_2 \le \ldots \le k_{\mu_j} \le |A|\}$$

be the set of all feasible assignments of operations of job J_j to intervals of the second type. For each big job we define a similar set K_j, but the tuples in K_j only allow placement of the operations of job J_j as described above.

For each job J_j we define variables $x_{jK}, K \in K_j$. The new linear program is as follows,

$$\sum_{K \in K_j} x_{jK} = 1, \quad J_j \in J, \tag{7}$$

$$\sum_{J_j \in J} \sum_{K \in K_j} \sum_{k_i = k, \pi_{ij} = t} p_{ij}\, x_{jK} \le \Delta_k, \quad t = 1, \ldots, m, k = 0, \ldots, |A| \tag{8}$$

$$x_{jK} \ge 0, \quad K \in K_j, J_j \in J. \tag{9}$$

Note that in any solution of this linear program the schedule for the long jobs is always feasible since there is at most one operation of a given job in any interval of the second type. Let C_{min} be the smallest value $C = a_{|A|+1}$ such that linear program (7)-(9) has a feasible solution for some outline. Using an argument similar to that of the proof of Lemma 1 we can prove that C_{min} is a lower bound on the makespan of an optimum preemptive schedule for the given set of jobs J.

Using binary search we can find a value C' satisfying $C_{min} \leq C' \leq C_{min} + \varepsilon P_{max}$ by approximately solving the linear program a constant number of times. Since linear programs (3)-(5) and (7)-(9) have the same structure we can use our rounding procedure to find in linear time a solution for the new linear program in which at most $2m(|A|+1)+1$ jobs receive fractional assignments (see Section 3.1).

After rounding the solution of the linear program we find a feasible schedule for every interval as follows. Consider an interval $[a_k, a_{k+1})$. Remove from the interval the operations belonging to big jobs. These operations will be reintroduced to the schedule later. Then use Sevastianov's algorithm as described in Section 3.3 to find a feasible schedule for the small and tiny jobs assigned to that interval. Finally place back the operations from the big jobs, scheduling them in the empty gaps left by the small and tiny jobs. Note that it might be necessary to split an operation of a big job in order to make it fit in the empty gaps. At the end we have a feasible schedule because there is at most one operation of each big job in the interval.

In this schedule the number of preemptions is at most $n\mu$ (since after introducing the operations from the big jobs, we might have this many preemptions for them). So there are in total $O(n)$ preemptions and only big operations are preempted.

Theorem 3 *The above algorithm is a linear time approximation scheme for the preemptive version of the job shop scheduling problem when m and μ are fixed. The solution that the algorithm finds has $O(n)$ preemptions.*

5 Extensions

Multi-stage job shop problem. In the s-stage job shop problem each machine of the classical job shop problem is replaced by a set of m_i parallel identical machines, $1 \leq m_i \leq m$. Our polynomial time approximation scheme works also in this case if the number of machines m_i on each stage i is fixed. Let the machines on stage i be numbered $s_1, s_2, \ldots, s_{m_i}$. In the linear program we use variables $x_{jK(r_1,\ldots,r_{\mu_j})}$, where r_i indicates the machine where the i-th operation O_{ij} of job J_j is scheduled. The same techniques used for the job shop scheduling problem can be used to design a polynomial time approximation scheme for this more general problem.

Dag shop problem. Another generalization of the job shop problem is the dag shop problem [17] (also called G-problem by Sevastianov [12, 13]). Here each job consists of a set of operations $\{O_{1j}, \ldots, O_{\mu j}\}$, and each job $J_j \in J$ has associated an acyclic directed graph $R_j = (O_j, E_j)$. In this graph an arc $(O_{i'j}, O_{ij})$ indicates that operation O_{ij} has to be executed after operation $O_{i'j}$. The problem is to find a schedule of minimum length that respects these ordering constraints.

The acyclic graph R_j can be translated directly into a set of tuples $K_j = \{(k_1, \ldots, k_\mu) | 0 \leq k_j \leq |A|$ for all $1 \leq j \leq \mu$ and $k_{i'} \leq k_i$ for every edge $(O_{i'j}, O_{ij}) \in E_j\}$ for each job $J_j \in T \cup S$. Again, the size of each set of tuples is constant, $|K_j| \leq (|A|+1)^\mu$, so we can use our algorithm with some small changes. Let us consider a single interval $[a_k, a_{k+1})$. Let $O(k)$ be the set of operations assigned to this interval. For each job J_j corresponding to the operations in $O(k)$

we use, instead of the acyclic graph $R_j(k)$ induced by the operations $O_{ij} \in O(k)$, a linear order that extends $R_j(k)$ and apply Sevastianov's algorithm [12] to a smaller instance of the job shop problem in each interval k. The rest of the algorithm is as before.

Two stage assembly problem. A further extension of our techniques allows the introduction of undirected edges in graph R_j. An undirected edge connecting two operations means that the operations are independent, i.e., they can be processed simultaneously. In the two-stage assembly scheduling problem [11] there are m machines in the first stage and one machine in the second stage. Every job has $m+1$ operations. The first m operations are connected by undirected edges and they must be processed on the first stage, and there is a directed edge (O_{ij}, O_{m+1j}) for each job J_j and each operation $O_{ij}, i = 1, \ldots, m$. The objective is to schedule the jobs on the machines so that the makespan is minimized.

We split the set of jobs into big, small, and tiny jobs. Then we fix the starting times for the big operations, allowing the possibility of processing in parallel operations that are connected by an undirected edge in R_j. For the small and tiny operations we define tuples K_j which also allow operations connected by an undirected edge to be processed in parallel. The rest of the algorithm is similar as that for the dag shop scheduling problem.

Job shop problem with release and delivery times. Our techniques can also handle the case in which each job J_j has a *release time* r_j (when it becomes available for processing) and *delivery time* q_j. If in a schedule job J_j completes processing at time C_j, then its *delivery completion time* is equal to $C_j + q_j$. The goal is to minimize the maximum delivery completion time of any job. Let r_{max} and q_{max} be the maximum release and delivery times, respectively. Then, $\max\{r_{max}, P_{max}, q_{max}\} \leq C^*_{max} \leq r_{max} + mP_{max} + q_{max}$. The idea is to round each release and delivery time up to the nearest multiple of $\varepsilon \cdot \max\{r_{max}, P_{max}, q_{max}\}$ for some value $\varepsilon > 0$. This increases the length of an optimum schedule by at most $2\varepsilon C^*_{max}$. Next, we apply a $(1+\varepsilon)$-approximation scheme (described below) that can handle $O(1/\varepsilon)$ different release times and delivery times. This gives an algorithm that finds a solution of length at most $(1+\varepsilon)(1+2\varepsilon) \leq 1+5\varepsilon$ times larger than the optimum.

We can easily modify the linear program to allow a constant number, $O(1/\varepsilon)$, of release dates and delivery times. Now the number of intervals of the second type is larger since we add each release time r_j and each point $C - q_j$ to the set A, but the total number is still constant: $O(m\mu/\alpha + 1/\varepsilon)$. We can solve the linear program as before in linear time. The rest of the approximation scheme is similar to that for the job shop scheduling problem.

6 Conclusions

We have proposed a linear time approximation scheme for the job shop scheduling problem when m and μ are fixed. Our method can be extended to design approximation schemes for the preemptive job shop scheduling problem, multi-stage job shop, dag shop, assembly scheduling, and job shop problem with release and delivery times.

Acknowledgments

We are grateful to Alexander Kononov, Lorant Porkolab, and Sergey Sevastianov for many helpful discussions and comments. The authors would like to thank David Shmoys for sending us his manuscript [16] about Sevastianov's algorithm.

References

[1] L. A. Goldberg, M. Paterson, A Srinivasan, and E. Sweedyk, Better approximation guarantees for job-shop scheduling, In Proceedings of the 8th Symposium on Discrete Algorithms (1997), pp. 599–608.

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

[3] L. A. Hall, Approximability of flow shop scheduling, Mathematical Programming **82** (1998), pp. 175–190.

[4] K. Jansen, R. Solis-Oba and M.I. Sviridenko, Makespan minimization in job shops: a polynomial time approximation scheme, Proceedings of the 31th Annual ACM Symposium on Theory of Computing, (1999), pp. 394–399.

[5] K. Jansen, R. Solis-Oba and M.I. Sviridenko, A linear time approximation scheme for the job shop scheduling problem, Proceedings of the Second Workshop on Approximation Algorithms (APPROX'99), to appear.

[6] E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys, Sequencing and scheduling: Algorithms and complexity, In S. C. Graves, A. H. G. Rinooy Kan and P. H. Zipkin, editors, Handbook in Operation Research and Management Science, **v. 4**, Logistics of Production and Inventory, North-Holland (1993), pp. 445–522.

[7] T. Leighton, B. Maggs and S. Rao, Packet routing and job-shop scheduling in O(Congestion+Dilation) Steps, Combinatorica **14** (1994), pp. 167–186.

[8] T. Leighton, B. Maggs and A. Richa, Fast algorithms for finding O(congestion+dilation) packet routing schedules, to appear in Combinatorica.

[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), pp. 257-301.

[10] C. N. Potts, Analysis of a linear programming heuristic for scheduling unrelated parallel machines, Discrete Applied Mathematics **10** (1985), pp. 155–164.

[11] C. N. Potts, S. V. Sevastianov, V. A. Strusevich, L. N. Van Wassenhove and C. M. Zwaneveld, The two-stage assembly scheduling problem: complexity and approximation, Operations Research **43** (1995), pp. 346–355.

[12] S. V. Sevastianov, Bounding algorithm for the routing problem with arbitrary paths and alternative servers, Cybernetics **22** (1986), pp. 773–780.

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

[14] S. V. Sevastianov and G. J. Woeginger, Makespan minimization in preemptive two machine job shops, Computing **60** (1998), pp. 73–79.

[15] S. V. Sevastianov and G. J. Woeginger, Makespan minimization in open shops: A polynomial time approximation scheme, Mathematical Programming **82** (1998), pp. 191–198.

[16] D. B. Shmoys, unpublished manuscript.

[17] D. B. Shmoys, C. Stein and J. Wein, Improved approximation algorithms for shop scheduling problems, SIAM Journal of Computing **23** (1994), pp. 617–632.

[18] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov and D. B. Shmoys, Short shop schedules, Operation Research **45** (1997), pp. 288–294.