RC23070 (W0401-094) January 20, 2004
Computer Science

IBM Research Report

Minimizing Makespan in No-Wait Job Shops

Nikhil Bansal
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Mohammad Mahdian
Laboratory for Computer Science
MIT
Cambridge, MA 02139

Maxim Sviridenko
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Minimizing makespan in no-wait job shops

Nikhil Bansal[*] Mohammad Mahdian[†] Maxim Sviridenko[‡]

Abstract

In this paper we study polynomial time approximation schemes (PTASes) for the no-wait job shop scheduling problem with the makespan objective function. It is known that the problem is MaxSNP-hard in the case when each job is allowed to have three operations or more. We show that if each job has at most two operations, the problem admits a PTAS if the number of machines is a constant (i.e., not part of the input). If the number of machines is not a constant, we show that the problem is hard to approximate within a factor better than $5/4$.

1 Introduction

Problem statement. A *job shop* is a multi-stage production process with the property that all jobs have to pass through several machines. There are n jobs numbered 1 through n, where each job j is a chain of m_j operations $O_{j,1}, \dots, O_{j,m_j}$. Every operation $O_{j,i}$ is preassigned to one of the m *machines* $M_1, \dots, M_m$ of the production process. The operation $O_{j,i}$ has to be processed for $p_{j,i}$ time units at its machine $m_{j,i}$; the value $p_{j,i}$ is called its *processing time* or its *length*. In a feasible schedule for the n jobs, at any moment in time every job is processed by at most one machine and every machine executes at most one job. Furthermore, for each job j, the operation $O_{j,i-1}$ is always processed before the operation $O_{j,i}$, and each operation is processed without interruption on the machine to which it was assigned. A *flow shop* is a special case of the job shop where each job has exactly one operation on each machine, and where all jobs pass through the machines in the same order $M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_m$. In an *open shop* the ordering of the operations in a job is not fixed and may be chosen by the scheduler. In a *mixed shop* there are jobs of the "open shop type" with any order between jobs and jobs of the "job shop type" with chain precedence constraints between operations. For further references and a survey of the area, see Lawler, Lenstra, Rinnooy Kan, and Shmoys [8] and Chen, Potts, and Woeginger [1].

[*]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Email: nikhil@cs.cmu.edu.

[†]Lab. for Computer Science, MIT, Cambridge, MA 02139, USA. Email: mahdian@theory.lcs.mit.edu.

[‡]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA. Email: sviri@us.ibm.com.

In this paper, we are interested in shop scheduling problems under the *no-wait constraint*. In such problems, there is no waiting time allowed between the execution of consecutive operations of the same job. Once a job has been started, it has to be processed without interruption, operation by operation, until it is completed. There are two main types of applications of shop scheduling problems with the no-wait constraint. The first is when the no-wait constraint is embedded into production technology. For example some characteristics of a material (such as temperature) require that one operation must processed immediately after another. It happens for instance in steel manufacturing when different operations like molding into ingots, unmolding, reheating, soaking, rolling must be done in a no-wait fashion. Other examples occur in chemical industry (anodizing of aluminum products), food processing (canning must be done immediately after cooking). The second type of applications of the no-wait scheduling problems is modelling the situation when there are no intermediate storage between or on machines to store the job between two consecutive operations. In this case the only way is to execute a job in a no-wait fashion. Such situations are quit common in packet routing. The job shop scheduling includes hot potato routing problems as special cases if routes in the network are fixed for all packets.

Our goal is to find a feasible schedule that minimizes the *makespan* (or *length*) $C_{\max}$ of the schedule, i.e., the maximum completion time among all jobs. The minimum makespan among all feasible schedules is denoted by $C^*_{\max}$. We say that an approximation algorithm has *performance guarantee* ρ for some real $\rho > 1$, if it always delivers a solution with makespan at most $\rho C^*_{\max}$. Such an approximation algorithm is then called a *ρ-approximation* algorithm. A family of polynomial time $(1+\epsilon)$-approximation algorithms over all $\epsilon > 0$ is called a *polynomial time approximation scheme* (PTAS).

Approximability of "ordinary" shop problems. The approximability of classical shop problems (without the no-wait constraint) is fairly well understood: If the number of machines and number of operations per job is a fixed value that is not part of the input, then the flow shop (Hall [4]), the open shop (Sevastianov and Woeginger [15]), and the job shop (Jansen, Solis-Oba and Sviridenko [7]) admit PTASes. On the other hand, if the number of machines is part of the input, then none of the three shop scheduling problems have a PTAS unless $\mathcal{P} = \mathcal{NP}$ (Williamson et al. [22]).

Complexity of no-wait shop problems with the makespan objective. Sahni and Cho [14] prove that the no-wait job shop and the no-wait open shop problems are strongly $\mathcal{NP}$-hard, even if there are only two machines and if each job consists of only two operations. Röck [12] proves that the three-machine no-wait flow shop is strongly $\mathcal{NP}$-hard, refining the previous complexity result by Papadimitriou and

Kanellakis [10] for four machines while the classic result by Gilmore and Gomory [2] yields an $O(n \log n)$ time algorithm for the two-machine no-wait flow shop. Sviridenko and Woeginger [18] prove that the three machine no-wait job shop with at most three operations per job is MAXSNP-hard and therefore is unlikely to admit a PTAS. Hall and Sriskandarajah [5] provide a thorough survey of complexity and algorithms for no-wait scheduling.

Approximability of no-wait shop problems. For all shop scheduling problems on m machines, sequencing the jobs in arbitrary order yields a (trivial) polynomial time m-approximation algorithm. Röck and Schmidt [13] improve on this for the no-wait flow shop and give an $\lceil m/2 \rceil$-approximation algorithm. For the flow shop scheduling problem a *permutation schedule* is a schedule in which each machine processes the jobs in the same order. In the *no-wait permutation flow shop* problem, only permutation schedules are feasible schedules. Sviridenko and Woeginger [18] obtain a PTAS for the no-wait permutation flow shop problem with a fixed number of machines. Sidney and Sriskandarajah [17] obtain a $3/2$-approximation algorithm for the 2-machine no-wait open shop problem. Glass, Gupta and Potts [3] consider the no-wait scheduling of n jobs in a two-machine flow shop where some jobs require processing on the first machine only. In contrast with the standard no-wait two-machine flow shop, this problem is NP-hard. Glass, Gupta and Potts describe a $4/3$-approximation algorithm for this problem. Papadimitriou and Kanellakis [10], and Sidney, Potts, and Sriskandarajah [16] study various generalizations and modifications of the no-wait flow shop problem on two machines. For these generalizations the authors manage to design approximation algorithms with performance guarantees strictly better than two. All of the above algorithms, except for results of Röck and Schmidt [13], are based on the famous algorithm of Gilmore and Gomory [2].

Our results. In this paper we concentrate on the no-wait job shop scheduling with two operations per job, since the problem with three operations per job is MAXSNP-hard even for the problem restricted to the instances having just three machines [18]. We prove that there exists a PTAS for this problem if the number of machines is a fixed number; i.e., it is not a part of the input. We also show that it is NP-hard to find a schedule of length at most four for this problem if the number of machines is a part of the input by a reduction from the NP-complete problem [6] of edge coloring of cubic graphs using three colors. This result implies that there is no approximation algorithm for the problem with performance guarantee better then $5/4$ unless $\mathcal{P} = \mathcal{NP}$. Our PTAS can be easily generalized to the mixed shop scheduling problem with at most two operations per job, and therefore our result is an improvement over results from [3] and [17].

2 The Algorithm

In this section we present a polynomial time approximation scheme for the no-wait job-shop scheduling with a constant number of machines and at most two operations per job. Let $\mathcal{J}$ be a collection of jobs. We can assume, without loss of generality, that each job has exactly two operations, since if there are jobs with only one operation, we can add a second operation of length zero to such a job, which has to be run on a new dummy machine (The same dummy machine is used for all such jobs, so that the number of machines remains constant). We also assume that precision parameter $\varepsilon > 0$ is chosen in such a way that $1/\varepsilon$ is an integer. For a job $j \in \mathcal{J}$, let $O_{j,1}$ and $O_{j,2}$ denote the two operations of this job, and $p_{j,1}$ and $p_{j,2}$ be the length of these two operations. The length of a job is the sum of the lengths of its operations. The pair $(p_{j,1}, p_{j,2})$ indicates the *type* of job j. Let $L_{max} = \max_{s=1,\dots,m}\{\sum_{O_{ij}|m_{ij}=M_s} p_{ij}\}$ denote the maximum machine load. This is a trivial lower bound on the optimal makespan.

Our algorithm consists of three steps. In the first step, we round the given instance to obtain a *well-behaved* instance of the problem. The results of this step are summarized in the following lemma.

Lemma 1 *Let $\mathcal{J}$ be a given instance of the no-wait job shop scheduling problem with two operations per job and m machines where m is a fixed number. Then there exists another instance $\mathcal{J}'$, such that*

- **i.** *A $(1 + O(\epsilon))$-approximately optimal schedule for $\mathcal{J}'$ can be transformed in polynomial time into a $(1 + O(\epsilon))$-approximately optimal solution for $\mathcal{J}$.*
- **ii.** *The processing time of every operation of jobs in $\mathcal{J}'$ is a positive integer, and the maximum load on a machine in $\mathcal{J}'$ (i.e., $L_{max}(\mathcal{J}')$) is at most n/ϵ.*
- **iii.** *The jobs in $\mathcal{J}'$ are partitioned into k* blocks $B_1, B_2, \dots, B_k$ *such that every job in B_i has length at least l_i and at most u_i, where l_i and u_i satisfy $u_{i+1} = \epsilon\, l_i = \epsilon^{1/\epsilon}\, u_i$ for every i.*
- **iv.** *For every i, there are at most a constant number of job types in B_i.*

This lemma will be proved in Section 2.1. In the second step of the proof (presented in Section 2.2), for a rounded instance $\mathcal{J}$ and a target makespan C, we will define an auxiliary graph $G(C)$ with one *root* and a set of *sinks*, and prove the following lemmas.

Lemma 2 *For every rounded instance $\mathcal{J}$ and a target makespan C, if there is a schedule with makespan C for $\mathcal{J}$, then there is a path in $G((1 + O(\epsilon))C)$ from the root to one of the sinks.*

Lemma 3 *If there is a path in $G(C)$ from the root to one of the sinks, then we can construct in polynomial time a feasible schedule for $\mathcal{J}$ that has makespan at most $(1 + O(\epsilon))C$.*

Note that constants hidden in $O(.)$ notation depend on m which is a fixed integer. Finally, in Section 2.3 we will prove the following lemma.

Lemma 4 *For any rounded instance $\mathcal{J}$ and any C, $1 \leq C \leq mn/\epsilon$, the graph $G(C)$ can be computed in polynomial time.*

Using the above lemmas, we can prove the following theorem.

Theorem 1 *There is a PTAS for no-wait job scheduling where each job has two operations and the number of machines is a constant.*

Proof. We start by rounding the instance using Lemma 1. Then, we do a binary search to find the smallest value of C between 1 and $mL_{max} = mn/\epsilon$ such that there is path from the root to a sink in $G(C)$ (By Lemma 4 every step of this search can be done in polynomial time). Call this value C^*. We know by Lemma 2 that the optimum makespan is at least $C^*/(1 + O(\epsilon))$. Furthermore, Lemma 3 gives us a way to construct a schedule with makespan at most $C^*(1+O(\epsilon))$ for the rounded instance. Finally, we use Lemma 1 to convert this schedule into a schedule of the original instance. □

2.1 Rounding

The following lemma is the main tool we are going to use in our rounding steps. The main reason why it is hard (or impossible) to build a PTAS for the no-wait job shop with more then two operations per job is that this lemma cannot be generalized to handle more than two operations per job.

Lemma 5 *Let $\mathcal{J}'$ be the instance obtained from $\mathcal{J}$ by increasing the size of an operation $O_{j,i}, i \in \{1, 2\}$ of a job $j \in \mathcal{J}$ by Δ. Then the optimal makespan for $\mathcal{J}'$ is at most the optimal makespan of $\mathcal{J}$ plus Δ.*

Proof. We first prove the following claim.

Claim 1 *Let S be a valid no-wait schedule for a set of jobs $\mathcal{J}$, t be a given time step, and $t_{j,i}$ ($j \in \mathcal{J}, i \in \{1, 2\}$) denote the time when the operation $O_{j,i}$ finishes execution in S. Let $A = \{j : j \in \mathcal{J}, t_{j,1} \geq t\}$. Then the schedule S' obtained by scheduling the operation i of job j such that it finishes at time $t'_{j,i} = t_{j,i}$ for $j \notin A$ and $t'_{j,i} = t_{j,i} + \Delta$ for $j \in A$ is a valid no-wait schedule for $\mathcal{J}$.*

Proof. Since both operations of a job are shifted by the same amount (0 or Δ), S' is clearly a no-wait schedule. We only need to verify that no two jobs in S' are scheduled on the same machine at the same time. One of these two operations, say O_{ji}, finishes before the other one starts in S, because S is a valid schedule. Thus, by the definition of A, if $j \in A$, we must also have $j' \in A$. Therefore, either $t'_{ji} - t_{ji} = t'_{j'i'} - t_{j'i'}$, or $t'_{ji} - t_{ji} = 0$ and $t'_{j'i'} - t_{j'i'} = \Delta$. It is easy to see that in both cases O_{ji} and $O_{j'i'}$ do not conflict in schedule S'. □

Consider an optimal schedule S. Let t denote the time when the operation $O_{j,i}$ (whose length we would like to increase) finishes in S. We apply the above claim on the schedule S with this t. If $i = 1$, by our choice of t, the operation $O_{j,i}$ will be scheduled Δ time units later in S' than in S. However, any operation that was scheduled before $O_{j,i}$ on the same machine, will be scheduled at the same time in S' as in S. Therefore, in the schedule S', this machine is free for Δ time steps before it executes the operation $O_{j,i}$. Thus, in S' we can increase the length of $O_{j,i}$ by Δ, without increasing the makespan. Similarly, if $i = 2$, then $O_{j,i}$ is scheduled at the same time in S' as in S, but any operation after that on the same machine will be scheduled Δ units later. Thus, this machine is free for Δ time steps after $O_{j,i}$. This allows us to increase the length of this operation by Δ without increasing the makespan beyond that of S'. Finally, we note that the makespan of S' is Δ plus the makespan of S. □

Proof of Lemma 1. Armed with the rounding tool, we describe a series of transformations which affect the optimum only by a factor of $1 + O(\epsilon)$. Without loss of generality we assume that $1/\epsilon$ is an integer. Notice that in the following, L_{max} refers to the maximum machine load with respect to the current instance (i.e., after rounding in the previous steps).

1. Dealing with small operations: First, we round up the length of operations such that the length of each operation is at least $\epsilon L_{max}/n$. This affects the total processing time by at most $2\epsilon L_{max}$. Then, for every job j, we round up the length of the smaller operation of j, such that the length of this operation is at least ϵ times the length of j. This increases the total processing time by at most $\epsilon m L_{max}$.

2. Rounding processing times: We round the sizes of the operations in two steps. We first round up each size to a power of $1 + \epsilon$, and then round up the size to an integral multiple of $\epsilon L_{max}/n$. This affects the makespan by at most a factor of $1 + O(\epsilon)$ since the change in the total processing time during the rounding is at most $(m + 2)\epsilon L_{max}$. We now rescale the processing times by dividing all of them by $\epsilon L_{max}/n$. After doing so, all processing times will be positive integers, and the maximum machine load in the resulting instance is n/ϵ. Note that in any

schedule for this instance without “unnecessary” idle times each operation starts at an integral time moment.

3. Partitioning jobs into blocks: Let $\mathcal{J}_i$ be the subset of jobs whose length is in the interval $(\epsilon^{i+1}L_{max}, \epsilon^i L_{max}]$. This partitions the set of jobs into groups $\mathcal{J}_0, \mathcal{J}_1, \ldots$. Next we define the set of blocks. Block B_1, consists of groups 0 through $b-1$ where $b < 1/\epsilon$ will be a constant specified later. Block B_i, for $i \geq 2$, consists of groups $b + (i-2)/\epsilon + 1$ through $b + (i-1)/\epsilon - 1$. By definition, the length of a job in B_i is at least l_i and at most u_i, where $l_i = \epsilon^{b+(i-1)/\epsilon} L_{max}$ and $u_i = \epsilon^{b+(i-2)/\epsilon+1} L_{max}$. Notice that we have $u_{i+1} = \epsilon\, l_i = \epsilon^{1/\epsilon}\, u_i$ for every i, as desired. The only problem with this “partition” is that it leaves out some of the jobs: the jobs that are in groups $\mathcal{J}_b, \mathcal{J}_{b+1/\epsilon}, \mathcal{J}_{b+2/\epsilon}, \ldots$ are not included in any of the blocks. However, we will argue that by a suitable choice of b, the total length of these jobs is small, and therefore we can delete them from the instance.

Let X_i be the total length of the jobs in $\mathcal{J}_i$, and $Y_b = \sum_{i \geq 0} X_{b+i/\epsilon}$ be the total length of the jobs that are left out from the blocks defined above. Since the total length of all jobs in the instance is $\sum_{b=1}^{1/\epsilon} Y_b$, there must be a choice of b for which Y_b is at most ϵ times the total length of all jobs, i.e., $Y_b \leq \epsilon\, m L_{max}$.

By deleting jobs belonging to the set $\cup_{i \geq 0} \mathcal{J}_{b+i/\epsilon}$ the optimal makespan changes by at most $O(\epsilon) L_{max}$, since we can schedule these jobs separately at the end. Moreover, the set of remaining jobs is exactly $\cup_{i \geq 1} B_i$. Thus, we can assume that the instance consists only of the blocks $\cup_{i \geq 1} B_i$.

The second and the third steps in the above rounding procedure guarantee that conditions (ii) and (iii) of Lemma 1 hold. In order to verify the condition (iv), notice that by the first step in the above procedure, the length of the smaller operation of jobs in B_i is at least $\epsilon\, l_i$, and the total length of any such job is between l_i and $\epsilon^{1/\epsilon - 1}\, l_i$. Furthermore, the second step of the above procedure guarantees that there are at most $O(\log(a_{max}/a_{min}))$ different possible values for the length of an operation in the interval $[a_{min}, a_{max}]$. This ensures that the number of job types in each block is $O(1)$. □

Throughout the rest of this section, we denote the number of jobs in B_i by n_i. Also, let V_i denote the total processing time of the jobs in B_i. Clearly, $n_i \cdot l_i \leq V_i \leq n_i \cdot u_i$.

2.2 The auxiliary graph

Let S be a partial schedule of jobs. Consider a time instant t. At this time instant, some of the machines are free and some are not. We can denote this by a number g between 0 and $2^m - 1$, whose binary representation has a zero in the i'th position if and only if the i'th machine is free at time t. A *gap* of type g is a continuous interval of time, during which the pattern of free machines is given by g. See Figure 1 for an example.

The *gap profile* of a partial schedule S is a 2^m tuple indexed by 2^m different gap types such that its g'th entry is the total length of all time intervals (from the start of the schedule until its completion) that are of type g. In other words, the gap profile of a schedule keeps track of the cumulative length of intervals of each gap type. For example, the gap profile of the schedule in Figure 1 is a 32-tuple with values 2, 2, and 1 in the positions 00110, 01010, and 01001, zero everywhere else. Conversely, given a gap profile G, we can build a configuration called the *canonical configuration* for G as follows: The configuration consists of 2^m time intervals, where the g'th interval is of length G_g and the set of machines that are free during this time interval is precisely the set of bits of g that are zero. The ordering of the intervals does not matter. We denote the canonical configuration corresponding to the gap profile G by $Z(G)$. For example, the canonical configuration corresponding to the gap profile in Figure 1 is shown in Figure 2.

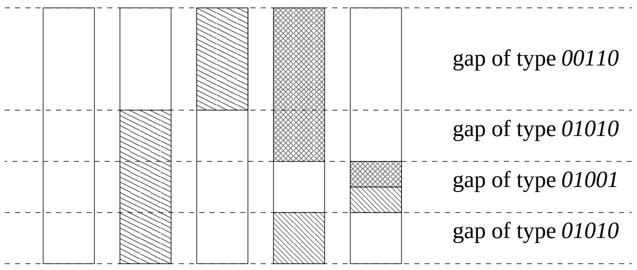

Figure 1: An example for gap types

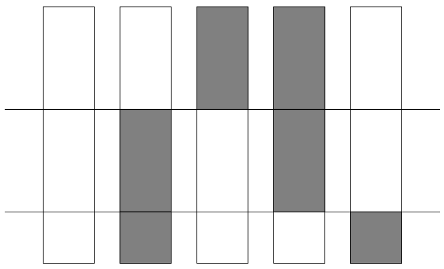

Figure 2: Canonical configuration of the schedule in Figure 1

For a set $\mathcal{J}$ of jobs and gap profiles G and G', we say $G \xrightarrow{\mathcal{J}} G'$ if there is a way to schedule the jobs in $\mathcal{J}$ in the free spaces of the configuration $Z(G)$ such that the gap profile of the resulting configuration is G'. The idea of our algorithm is to compute a sequence $G^{(0)}, G^{(1)}, \ldots, G^{(k+1)}$ of gap profiles such that $G^{(0)}$ is the profile corresponding to the empty schedule with makespan C, and for every i, $G^{(i-1)} \xrightarrow{B_i} G^{(i)}$. Notice that this does not immediately give us a schedule for $\cup_i B_i$, since, for example, after scheduling the jobs of B_1 in $Z(G^{(0)})$, we get a schedule with gap profile $G^{(1)}$, which might not be the same as $Z(G^{(1)})$ (i.e., intervals of the same gap type may not be consecutive in this schedule). However, we will argue in the rest of this section that this problem only costs us an extra factor of $1 + O(\epsilon)$.

We are now ready to define the auxiliary graph. Fix a number C, $1 \leq C \leq mL_{max}$. This number is the makespan that we are aiming for. Let $\mathcal{G}$ be the set of all possible gap profiles $G = (G_0, \ldots, G_{2^m-1})$ such that $\sum_{i=0}^{2^m-1} G_i \leq C$, and each G_i is an integer. Clearly, the number of such gap profiles (i.e. $|\mathcal{G}|$) is at most $O(n^{2^m})$. Let k denote the number of blocks in the rounded scheduling instance. The auxiliary graph $G(C)$ consist of $k+1$ layers $L_0, \ldots, L_k$. The layer L_0 consists of a single node corresponding to the gap structure that has C in the 0'th entry (i.e., the entry corresponding to the intervals where all machines are free), and zero everywhere else. We call this node the *root*. For $i > 0$, layer L_i consists of $|\mathcal{G}|$ nodes each corresponding to a gap profile and denoted by (i, G) where $G \in \mathcal{G}$. There is a directed edge from a node $(i-1, G)$ in level $i-1$ to a node (i, G') in level i if $G \xrightarrow{B_i} G'$.

We now prove Lemmas 2 and 3. These lemmas show how the optimal makespan and the graph $G(C)$ are related.

Proof of Lemma 2. Consider the optimal schedule S. Let $i = 0$. Consider the time instants when an operation in B_i starts or finishes execution. There are at most $3n_i$ such time instants. Consider all the jobs in $\cup_{j>i} B_j$ that are running at one of these time instants. There are at most $3n_i m$ such jobs. Delete these jobs from the schedule and schedule them separately at the end such that their processing times do not overlap with each other or with other jobs. Repeat for $i = 1, \ldots, k$. Call the resulting schedule S'.

Since the length of each job in $\cup_{j>i} B_j$ is at most u_{i+1}, the above procedure adds at most $\sum_{i=1}^{k} 3n_i m u_{i+1}$ to the makespan. Recall that the total processing time of jobs in B_i is $V_i \geq n_i l_i = n_i u_{i+1}/\epsilon$. Therefore, the makespan C' of the schedule S' is at most $\sum_{i=1}^{k} 3\epsilon m V_i \leq 3\epsilon m^2 L_{max}$ more than the optimal makespan.

We now show how to obtain a path in the graph $G(C')$. Let $S'_{\leq i}$ denote the partial schedule that consists only of jobs in $\cup_{j\leq i} B_j$ scheduled at the same time as in S'. Let $G^{(i)}$ be the gap profile of $S'_{\leq i}$. We show that $G^{(i)} \xrightarrow{B_{i+1}} G^{(i+1)}$. This would give us a path from the root to a vertex in L_k in $G(C')$. Since in S' no job in B_{i+1} is executed at a time when a job in $\cup_{j\leq i} B_j$ begins or finishes execution, we can do the following: Consider all intervals of gap type g in $S'_{\leq i}$. Look at the schedule of the jobs in B_{i+1} in S' in these intervals. Now if we concatenate these schedules, this gives a way of placing the jobs in B_{i+1} in the free spaces of the interval of type g in the configuration $Z(G^{(i)})$. Do this for each g. This gives us a schedule of all jobs of B_{i+1} in the free spaces of the configuration $Z(G^{(i)})$. Furthermore, by construction, the gap profile of this schedule is the same as the gap profile of $S'_{\leq i+1}$, which is $G^{(i+1)}$. Therefore, $G^{(i)} \xrightarrow{B_{i+1}} G^{(i+1)}$. This gives us a path in $G(C')$. □

Proof of Lemma 3. Let $(0, G^{(0)}), (1, G^{(1)}), \ldots, (k, G^{(k)})$ be a path from the root to a node in the last level of $G(C)$. The edge e_i from $(i-1, G^{(i-1)})$ to $(i, G^{(i)})$ gives a way of scheduling B_i in the configuration $Z(G^{(i-1)})$ to yield the gap profile $G^{(i)}$.

We first construct a *preemptive* schedule. The construction is inductive: in the i'th step, we construct a preemptive schedule S_i of jobs in $\cup_{j\leq i} B_j$ whose gap profile is $G^{(i)}$. The base of the induction ($i = 0$) follows from the definition of the root of $G(C)$. Suppose we have constructed a partial preemptive schedule S_i with gap profile $G^{(i)}$ for $\cup_{j\leq i} B_j$. We now show how to schedule B_{i+1} in the empty spots of S_i and obtain S_{i+1} such that the gap profile of S_{i+1} is $G^{(i+1)}$. The edge e_i shows us how to schedule B_{i+1} in the empty spots of $Z(G^{(i)})$ and obtain a schedule of profile $G^{(i+1)}$. Call this schedule R. By definition, the empty spots of S_i have the same structure as those of $Z(G^{(i)})$, except in $Z(G^{(i)})$ all intervals of the same type are concatenated. Therefore, we can cut each gap of $Z(G^{(i)})$ into several gaps and order them so that we obtain the same configuration as S_i. Using these cuts on the schedule R gives us a schedule of B_{i+1} in the empty spots of S_i with gap profile $G^{(i+1)}$, as desired. The only catch is that by cutting a gap of $Z(G^{(i)})$ into several gaps and reordering them, we might preempt some of the jobs of B_{i+1}. That is why we call this schedule *preemptive*.

After finding a preemptive schedule S_k of all jobs, the next step is to convert this schedule into a non-preemptive feasible schedule. This is done in the straightforward way: if any job is preempted, we remove it and place it at the end of the schedule.

We only need to show that this does not add too much to the makespan. We show this by proving that the total length of the jobs that are preempted in the schedule S_k is at most $O(\epsilon L_{max})$. In the i'th step of building S_k (i.e., when we are constructing S_{i+1}), the number of jobs in B_{i+1} that we might preempt is upper bounded by the number of gaps in S_i times m, which is at most $3m|\cup_{j=1}^{i} B_j| = 3m\sum_{j=1}^{i} n_j$. Since the length of each job in B_{i+1} is at most u_{i+1}, the total length of jobs that are preempted in this step is at most $3m\sum_{j=1}^{i} n_j u_{i+1} =$

$3m\sum_{j=1}^{i} n_j l_j \epsilon^{1+(i-j)/\epsilon}$. Using the fact that the total processing time of jobs in B_j is $V_j \geq n_j l_j$, we can bound above the total length of jobs that are preempted in S_k by

$$\sum_{i=1}^{k} 3m \sum_{j=1}^{i} V_j \epsilon^{1+(i-j)/\epsilon} = 3m\epsilon \sum_{j=1}^{k} (\sum_{i=j}^{k} \epsilon^{(i-j)/\epsilon}) V_j \leq 3m\epsilon \sum_{j=1}^{k} \frac{V_j}{1-\epsilon^{1/\epsilon}} \leq 4m^2 \epsilon L_{max},$$

where the last inequality holds for all $\epsilon \in [0, 1/2]$. Therefore, the total length of jobs that are preempted is at most $O(\epsilon)C$. Hence, after scheduling these jobs at the end the total makespan is at most $(1 + O(\epsilon))C$. □

2.3 Computing the auxiliary graph

In this section we will use dynamic programming to prove that the graph $G(C)$ defined in the previous section can be computed in polynomial time.

Proof of Lemma 4. The number of vertices of the graph $G(C)$ is a polynomial in the number of jobs. Therefore, we only need to show that it is possible to decide whether there is an edge between two vertices in polynomial time. In other words, given two gap profiles G and G' and a block of jobs B_i, we need to decide whether jobs in B_i can be scheduled in $Z(G)$ to yield the gap structure G'. By Lemma 1 we know that the number of different job types in B_i is a constant r independent of the size of the input. Therefore, there are at most n^r distinct sets of jobs in B_i. Call the collection of all such sets $\mathcal{S}$. Also, recall that $\mathcal{G}$ denotes the set of all possible gap profiles $(G_0, G_2, \dots, G_{2^m-1})$ with $\sum G_i \leq C$, and that $|\mathcal{G}|$ is a polynomial in n. In order to decide whether we can schedule B_i in the free spaces of $Z(G)$ to get the gap profile G', we define a table A and fill it using dynamic programming.

For a set $S \in \mathcal{S}$ of jobs in B_i, a gap profile $H \in \mathcal{G}$, and numbers $t \in \{1, \dots, C\}$, $a_1, a_2, \dots, a_m \in \{0, \dots, r\}$ and $C_1, \dots, C_m \in \{0, \dots, C\}$, we define the entry $A(t, S, H, a_1, \dots, a_m, C_1, \dots, C_m)$ of the table. This entry keeps an answer on the following question: Is there a feasible schedule such that jobs from the set S are scheduled completely in the free spaces of $Z(G)$ in time interval $(0, t]$ (recall that ordering of intervals corresponding to configurations in G is not important but we must fix one for $Z(G)$), machine i is running a job of type a_i in time interval $(t-1, t]$ (this job does not belong to S) and completion time of this job is $C_i \geq t$ (if machine i was not running anything at time t, we let $a_i = 0$ and $C_i = t$) and the gap profile of the schedule in time interval $(0, t]$ is H (i.e., for every g, there are exactly H_g time units before t that are of type g)?

Assume we computed all elements of the table A for $t \leq \tau - 1$ we now show how to compute elements of the table for $t = \tau$. In order to compute the entry $A(\tau, S, H, a_1, \dots, a_m, C_1, \dots, C_m)$, we need to guess the jobs that machines run at time step $\tau - 1$. On some machines, this job is uniquely specified from the information that we have about the jobs (their types and completion times) that machines are running at time τ and the assumption that the schedule must be no-wait and non-preemptive. On other machines, we need to decide about the type of the job that we want to run at time $\tau - 1$. But since both the number of job types and the number of machines are constants, the number of possibilities is a constant. Moreover, the only way the job can run at time $\tau - 1$ and not run at time τ is when this job completes at time $\tau - 1$. For each such possibility, let ΔS be a set of jobs running at time $\tau - 1$ and $a'_1, \ldots, a'_m, C'_1, \ldots, C'_m$ be their types and completion times. We update update H by subtracting one from the entry corresponding to the gap type in the time interval $(\tau - 1, \tau]$, and look at the table A to check whether it is possible to finish the schedule corresponding to these updated parameters in the time interval $(0, \tau - 1]$, i.e. we look at the entry $A(\tau, S \setminus \Delta S, H, a'_1, \ldots, a'_m, C'_1, \ldots, C'_m)$. If this entry is one for at least one choice of the set $\Delta S \subseteq S$, i.e. jobs that are running at time $\tau - 1$, then we record 1 in the entry $A(\tau, S, H, a_1, \ldots, a_m, C_1, \ldots, C_m)$ of the table; otherwise, we write 0 in this entry.

It is easy to verify that the above procedure fills the table in polynomial time. Now it is enough to check the entries $A(C, B_i, G', a_1, \ldots, a_m, C, \ldots, C)$ of the table for all possible choices of $a_1, \ldots, a_m$, i.e. types of jobs which have completion time equal to the makespan of the schedule. If there is one in at least one such entry then there is an edge from $(i - 1, G)$ to (i, G') in the graph $G(C)$ and conversely if all those entries are 0 then there is no edge from $(i - 1, G)$ to (i, G') in the graph $G(C)$. If we would like actually to compute the schedule corresponding to some entry $A(C, B_i, G', a_1, \ldots, a_m, C, \ldots, C)$, we should use the standard backtracking procedure which works in C steps and on step $i = 0, \ldots, C - 1$ it finds jobs finishing at time $C - i$. $\square$

Remark 1 As we already noticed our algorithm can be simply generalized to handle jobs with one operation only by introducing a dummy machine, therefore our result provides a PTAS for the problem studied in [3]. Another generalization of our algorithm can easily handle the so-called mixed shop scheduling problem with two operations per job, i.e. the problem where in addition to the jobs with precedence constraint between operations we have jobs with two operations which can be processed in any order. This problem is a mix between job shop and open shop scheduling problems. The only extension we need is to generalize notion of job type for this problem. Now job type includes operations lengths and an order in which those operations must be processed. All other steps of our algorithm are implemented without any changes. Therefore our algorithm provides a PTAS for the two machine open shop problem studied in [17].

3 Hardness result

In this section we will prove the following theorem.

Theorem 2 *The problem of deciding whether an instance of the no-wait job shop scheduling with 2 operations per job has a schedule of makespan at most 4 is NP-hard, even when restricted to instances that have operations of length one and two only.*

Proof. We give a reduction from the NP-complete problem [6] of edge-coloring 3-regular graphs with 3 colors. Let G be a given 3-regular graph. We construct an instance of the scheduling problem that has a schedule of makespan 4 if and only if G is 3-edge-colorable. This instance consists of two jobs J_{uv} and J_{vu} for each edge $e = (u, v)$ of G, and gadgets A_e that guarantee that the two jobs J_{uv} and J_{vu} are scheduled at the same time. The job J_{uv} has two operations, each of unit length. The first operation of this job runs on a machine M_u corresponding to the vertex u. Clearly, in any valid schedule with makespan 4, this operation must be schedule in one of the time steps 1, 2, or 3. We would like the time step at which this operation is scheduled to correspond to the color of the edge uv in the 3-edge-coloring of G. Since the first operation of the three jobs corresponding to the three edges incident on u must all be scheduled on the same machine M_u, they must be scheduled at different time steps. This corresponds to the condition that in an edge-coloring of G the edges incident on u must get different colors. Therefore, the only thing that we need in order to establish a correspondence between schedules in our instance and 3-edge-colorings in G is to somehow guarantee that the first operations of the two jobs J_{uv} and J_{vu} corresponding to the same edge e are scheduled at the same time step. This will be done by having the second operation of these jobs on machines in the gadget A_e that is described below. This gadget will guarantee that the second operations of J_{uv} and J_{vu} (and therefore their first operations) are scheduled at the same time step.

We now describe how to build the gadget A_e. We start by constructing a simple gadget T_1 such that by adding T_1 to a machine M, we can guarantee that a job of T_1 will be scheduled at time step 1 on M, and no other job of T_1 will be scheduled on M (in other words, T_1 keeps the machine M busy at time step 1). This gadget consists of two jobs J_1 and J_2, and two machines M_1 and M_2. J_1 has two operations, each of length two, that must be scheduled on machines M_1 and M_2, respectively. J_2 has two operations of length one, the first one of which is on machine M and the second one is on machine M_2. It is easy to verify that this gadget has the required property. Similarly, we can construct a gadget T_4 that keeps a machine M busy at time step 4. These gadgets are shown in Figure 3.

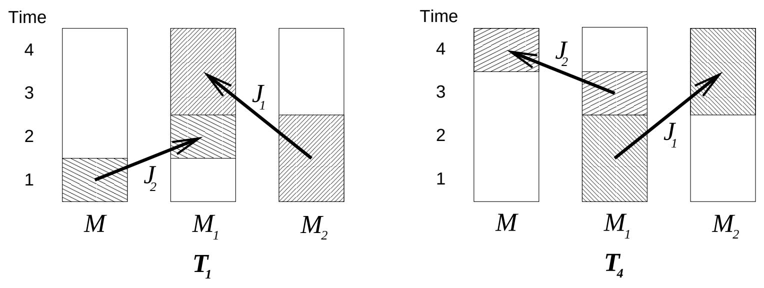

Figure 3: The gadgets T_1 and T_4

The second step in the construction of A_e is to construct gadgets H_{ij} (for $i, j \in \{2, 3, 4\}$) such that by adding H_{ij} to two machines M and M', we can guarantee that both M and M' are kept busy by two jobs J and J' of H_{ij} at the *same* time step, and this time step can be either i or j. More precisely, there are two jobs J and J' in H_{ij} which have one unit-length operation on machines M and M', respectively. These are the only operations in H_{ij} on M or M'. For $s \in \{i, j\}$, there is a valid schedule of H_{ij} of makespan 4 that schedules the operations of J and J' on machines M and M' both at time step s. These two combinations are the only possible time steps at which the operations of J and J' on M and M' can be scheduled in a valid schedule with makespan 4. Here we construct the gadgets H_{24} and H_{34}, which will be used in A_e.

The gadget H_{24} consists of two machines M_3 and M_3', jobs J, J', J_3, and J_3', and two copies of the gadget T_4. The two copies of T_4 keep the machines M_3 and M_3' busy at time step 4. Job J_3 has two unit-length operations, the first one on M_3 and the second one on M_3'. Similarly, J_3' has two unit-length operations on M_3' and M_3, respectively. Job J (J', respectively) has two unit-length operations, the first one on M_3 (M_3', respectively), and the second one on M (M', respectively). This gadget is shown in Figure 4. It is easy to verify that in any valid schedule of H_{24} of makespan 4, the first operations of the jobs J_3 and J_3' both start at the same time step, 1 or 2. Therefore, the only time steps that remains free on M_3 and M_3' for J and J' are the same time steps, and they are either both 1, or both 3. Thus, the second operations of J and J' are both scheduled either at time 2, or at time 4 on M and M'.

We can easily generalize the gadget H_{24} to a gadget $H_{24}^{(4)}$ that guarantees that either the four machines $M^{(1)}$, $M^{(2)}$, $M^{(3)}$, and $M^{(4)}$ are all kept busy at time step 2, or are all busy at time step 4. This gadget is shown in Figure 5.

We now describe the gadget H_{34}. This gadget consists of four machines $M^{(1)}$, $M^{(2)}$, $M^{(3)}$, and $M^{(4)}$, four jobs J, J', J_4, and J_5, four copies of the gadget T_1, and one copy of the gadget $H_{24}^{(4)}$. The four copies of T_1 keep the machines $M^{(1)}$, $M^{(2)}$,

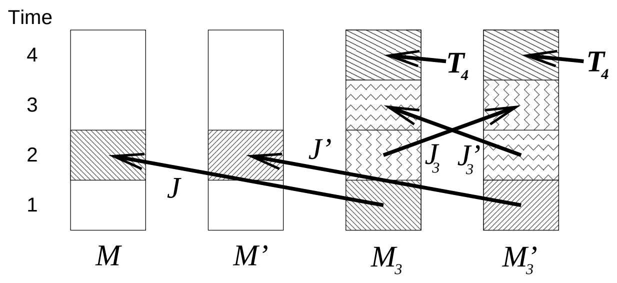

Figure 4: The gadget H_{24}

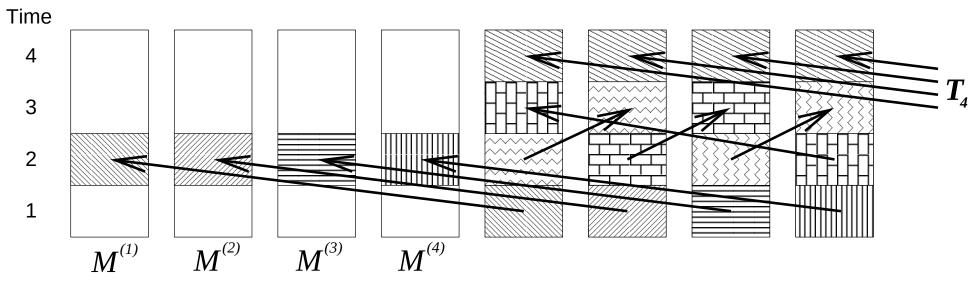

Figure 5: The gadget $H_{24}^{(4)}$

$M^{(3)}$, and $M^{(4)}$ busy at time step 1. The gadget $H_{24}^{(4)}$ keeps these machines busy either all at time step 2, or all at time step 4. The job J_4 (J_5, respectively) has two unit-length operations, the first one on $M^{(3)}$ ($M^{(4)}$, respectively), and the second one on $M^{(1)}$ ($M^{(2)}$, respectively). The operations of the job J (J', respectively) are of unit length, and are on machines $M^{(1)}$ ($M^{(2)}$, respectively), and M (M', respectively). See Figure 6. It is easy to verify that by this construction, either both machines $M^{(1)}$ and $M^{(2)}$ are kept busy by $H_{24}^{(4)}$ at time 2, and they are running the second operations of J_4 and J_5 at time 4, or they are both kept busy by $H_{24}^{(4)}$ at time 4, and are running J_4 and J_5 at time 3. Therefore, the first operations of J and J' are either both scheduled at time 2, or they are both scheduled at time 3. This proves that H_{34} has the required property.

We are now ready to describe the gadget A_e. This gadget consists of two machines M_{uv} and M_{vu} that will be assigned the second operations of J_{uv} and J_{vu}, a copy of H_{24} that keeps M_{uv} and M_{vu} busy either both at time 2, or both at time 4, and a copy of H_{34} that keeps M_{uv} and M_{vu} busy either both at time 3, or both at time 4. It is not difficult to see that this gadget has the required property that J_{uv} and J_{vu} must be processed in the same time unit on M_{uv} and M_{vu} which is either 2 or 3 or 4 in any valid schedule of makespan 4.

By combining the jobs J_{uv} and J_{vu} and the gadgets A_e for all edges e of the graph G, we obtain an instance of the scheduling problem that has a schedule of

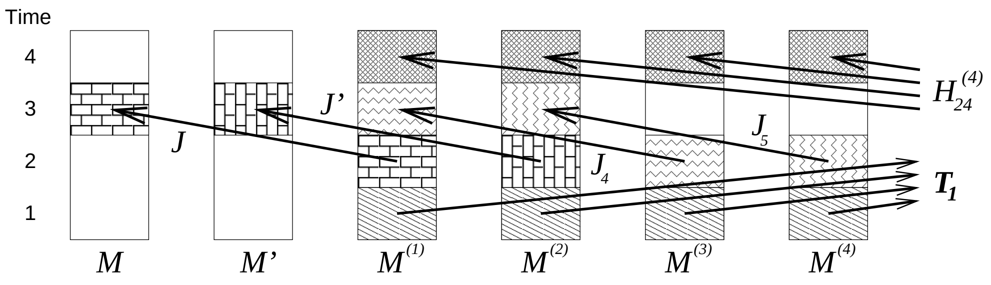

Figure 6: The gadget H_{34}

makespan 4 if and only if G is 3-edge-colorable. Furthermore, we know that every 3-regular graph is 4-edge-colorable. From a 4-edge-coloring of G, we can easily obtain a schedule of our instance by scheduling the jobs J_{uv} and J_{vu} at the time steps corresponding to the color of the corresponding edge in G, and scheduling the gadgets in the way that is described in the construction of the gadgets. Therefore, our instance always has a schedule of makespan at most 5. □

Corollary 3 *It is NP-hard to approximate the no-wait job shop scheduling problem within a factor less than* $5/4$*, even when the input is restricted to have 2 operations per job, and all operations of length 1 or 2.*

Remark 2 The only place we needed to use operations of length 2 in the above proof is in the construction of the gadgets T_1 and T_4. Unfortunately, we are unable to construct such gadgets only using unit-length operations. The no-wait job shop scheduling problem when the input is restricted to have 2 operations per job, and all operations are of length 1 can be studied as a special case of a graph coloring problem where every directed edge is colored with two colors a and b such that $k \leq b - a \leq l$. This problem is called (k, l)-coloring. It was first introduced in [20] and studied in the series of papers [9, 11, 19, 20, 21]. The no-wait job shop scheduling problem with unit length operations and two operations per job is exactly $(1, 1)$-coloring problem. By modifying our construction we can prove that recognizing if a directed multigraph can be $(1, 1)$-colored using 7 colors is an NP-complete problem. The question of finding the smallest number of colors for which this problem is NP-complete remains open (for three colors it can be reduced to the 2-SAT problem, see [22] for details).

References

[1] B. Chen, C. Potts, and G. Woeginger [1998]. A review of machine scheduling: complexity, algorithms and approximability. Handbook of combinatorial opti-

mization, Vol. 3, 21–169, Kluwer Acad. Publ., Boston, MA.

[2] P. GILMORE AND R. GOMORY [1964]. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research 12*, 655–679.

[3] C. GLASS, J.N.D. GUPTA, AND C.N. POTTS [1999]. Two-machine no-wait flow shop scheduling with missing operations. *Mathematics of Operations Research 24*, 911–924.

[4] L.A. HALL [1998]. Approximability of flow shop scheduling. *Mathematical Programming 82*, 175–190.

[5] N. HALL AND C. SRISKANDARAJAH [1996]. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research 44*, 510–525.

[6] I. HOLYER [1981]. The NP-completeness of edge-coloring. *SIAM Journal of Computing 10*, 718–720.

[7] K. JANSEN, R. SOLIS-OBA, AND M. SVIRIDENKO [2003]. Makespan minimization in job shops: a linear time approximation scheme. *SIAM Journal of Discrete Mathematics 16* , 288-300.

[8] E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS [1993]. Sequencing and scheduling: algorithms and complexity. In S. Graves, A.H.G. Rinnooy Kan and P. Zipkin (eds.) *Handbooks in Operations Research and Management Science, Volume 4, Logistics of Production and Inventory*, (North Holland, Amsterdam), 445–522.

[9] L. MELNIKOV AND V. VIZING [1999] The edge chromatic number of a directed/mixed multigraph. *J. Graph Theory 31* , 267–273.

[10] C.H. PAPADIMITRIOU AND P. KANELLAKIS [1980]. Flow-shop scheduling with limited temporary storage. *Journal of the ACM 27*, 533–549.

[11] A. PYATKIN [2002]. (k, l)-coloring of incidentors of cubic multigraphs. (Russian) *Diskretn. Anal. Issled. Oper. Ser. 1 v.9* , 49–53.

[12] H. RÖCK [1984]. The three-machine no-wait flow shop is NP-complete. *Journal of the ACM 31*, 336–345.

[13] H. RÖCK AND G. SCHMIDT [1983]. Machine aggregation heuristics in shop-scheduling. *Methods of Operations Research 45*, 303–314.

[14] S. SAHNI AND Y. CHO [1979]. Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research 4*, 448–457.

[15] S.V. SEVASTIANOV AND G.J. WOEGINGER [1998]. Makespan minimization in open shops: a polynomial time approximation scheme. *Mathematical Programming 82*, 191–198.

[16] J.B. SIDNEY, C.N. POTTS, AND C. SRISKANDARAJAH [2000]. A heuristic for scheduling two-machine no-wait flow shops with anticipatory setups. *Operations Research Letters 26*, 165–173.

[17] J.B. SIDNEY AND C. SRISKANDARAJAH [1999]. A heuristic for the two-machine no-wait open shop scheduling problem. *Naval Research Logistics 46*, 129–145.

[18] M. SVIRIDENKO AND G. WOEGINGER [2000]. Approximability and inapproximability results for no-wait shop scheduling problems. In Proceedings of FOCS00, 116-125.

[19] V. VIZING [2000]. The coloring of the incidentors and vertices of a directed multigraph. (Russian) *Diskretn. Anal. Issled. Oper. Ser. 1 7* , 6–16.

[20] V. VIZING, L. MELNIKOV AND A. PYATKIN [2000]. On the (k, l)-coloring of incidentors. (Russian) *Diskretn. Anal. Issled. Oper. Ser. 1 7* , 29–37.

[21] V. VIZING AND B. TOFT [2001]. The coloring of the incidentors and vertices of an undirected multigraph. (Russian) *Diskretn. Anal. Issled. Oper. Ser. 1 8,* 3–14.

[22] D.P. WILLIAMSON, L.A. HALL, J.A. HOOGEVEEN, C.A.J. HURKENS, J.K. LENSTRA, S.V. SEVASTIANOV, AND D.B. SHMOYS [1997]. Short shop schedules. *Operations Research 45*, 288–294.