SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 921

August 1990

IMPROVED APPROXIMATION ALGORITHMS FOR SHOP SCHEDULING PROBLEMS

By

D.B. Shmoys, C. Stein[1]
and J. Wein[2]

[1]Laboratory for Computer Science, MIT, Cambridge, MA.

[2]Department of Mathematics, MIT, Cambridge, MA.

Research partially supported by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun, by the Cornell Computational Optimization Project, and by DARPA Contract N00014-89-J-1988. Joel Wein is also supported by an ARO graduate fellowship.

Improved Approximation Algorithms for Shop Scheduling Problems

(Extended Abstract)

David B. Shmoys
SORIE
Cornell University
Ithaca, NY

Clifford Stein
Laboratory for Computer Science
MIT
Cambridge, MA

Joel Wein
Department of Mathematics
MIT
Cambridge, MA

August 22, 1990

Abstract

In the *job shop scheduling* problem we are given m machines and n jobs; a job consists of a sequence of operations, each of which must be processed on a specified machine, and the aim is to complete all jobs as quickly as possible. This problem is strongly $\mathcal{NP}$-hard even for very restrictive special cases. We give the first randomized and deterministic polynomial-time algorithms that yield polylogarithmic approximations to the optimal length schedule. Specifically, if $C^*_{\max}$ is the length of the optimal schedule, and $m_{\max}$ is the maximum number of operations in a job, we give a randomized algorithm that produces a schedule of length $O(\frac{\log^2(m \cdot m_{\max})}{\log\log(m \cdot m_{\max})} C^*_{\max})$, and a deterministic algorithm that yields a schedule of length $O(\log^2(m \cdot m_{\max})\, C^*_{\max})$. Our algorithms also work in the more general case where a job is given not a linear ordering of the machines on which it must be processed but an arbitrary partial order. Comparable bounds can also be obtained when there are m' types of machines, and each operation must be processed on one of the machines of a specified type.

Research partially supported by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun, by the Cornell Computational Optimization Project, and by DARPA Contract N00014-89-J-1988. Joel Wein is also supported by an ARO graduate fellowship.

1 Introduction

In the *job shop scheduling* problem we are given m machines and n jobs. A job consists of a sequence of operations, each of which must be processed on a specified machine; there may be more than one operation on a given machine. The operations of a job must be processed in the order specified by the sequence, subject to the constraint that on each machine, at most one job is scheduled at any point in time. We wish to produce a schedule of jobs on machines that minimizes $C_{\max}$, the time when all jobs have completed processing. This problem is strongly $\mathcal{NP}$-hard; furthermore, except for the cases when there are two jobs or when there are two machines *and* each job has at most two operations, essentially all special cases of this problem are $\mathcal{NP}$-hard, and typically strongly $\mathcal{NP}$-hard [5] [6]. For example, it is $\mathcal{NP}$-hard even if there are 3 machines, 3 jobs and each operation is of unit length; note that in this case we can think of the input length as the maximum number of operations in a job, $m_{\max}$. In addition to these theoretical results, the job shop problem is also one of the most notoriously difficult $\mathcal{NP}$-hard optimization problems in terms of practical computation, with even very small instances being difficult to solve exactly. A classic single instance of this problem involving only 10 jobs, 10 machines and 100 operations which was published in 1963, remained unsolved for 23 years despite repeated attempts to find an optimal solution [6].

In this paper we will focus on obtaining approximation algorithms for the job shop problem, and will evaluate these algorithms in terms of their performance guarantee, or in other words, their worst-case relative error. Let $C^*_{\max}$ be the value of $C_{\max}$ in the optimal solution. If a polynomial-time algorithm always delivers a solution of maximum completion time at most $\rho C^*_{\max}$, then we shall call it a ρ-approximation algorithm. The main result of this paper is the first randomized polylogarithmic approximation algorithm for job shop scheduling.

Theorem 1.1 There exists a polynomial-time randomized algorithm for job shop scheduling, that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2(m \cdot m_{\max})}{\log\log(m \cdot m_{\max})} C^*_{\max})$.

We formally define the job shop problem as follows. We are given a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of machines, a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of jobs, and a set $\mathcal{O} = \{O_{ij} | i = 1, \ldots, m_j, j = 1, \ldots, n\}$ of operations. Thus m is the number of machines, n is the number of jobs, m_j is the number of operations of job J_j, and $m_{\max} = \max_j m_j$. O_{ij} is the ith operation of J_j; it requires processing time on a given machine $\mu_{ij} \in \mathcal{M}$ for an uninterrupted period of a given length p_{ij}. (In other words, this is a *non-preemptive* model; if operations may be interrupted and resumed at a later time, this is called a *preemptive* model.) Each machine can process at most one operation at a time, and each job may be processed by at most one machine at a time. If the completion time of operation O_{ij} is denoted by C_{ij}, then the objective is to produce a schedule that minimizes the maximum completion time, $C_{\max} = \max_{i,j} C_{ij}$; the optimal value is denoted by $C^*_{\max}$.

Note that there are two very easy lower bounds on the length of an optimum schedule. Since each job must be processed, $C^*_{\max}$ must be at least the maximum total length of any job, $\max_{J_j} \sum_i p_{ij}$, which we shall call the *maximum job length* of the instance, $\Pi_{\max}$. Furthermore, each machine must process all of its operations, and so $C^*_{\max}$ must be at least $\max_{M_k} \sum_{\mu_{ij}=k} p_{ij}$, which we will call the *maximum machine load* of the instance, $P_{\max}$.

Our work is based on two very different approaches to the job shop problem. One approach is a geometric approach to shop scheduling, while the other is a randomized approach that finds its genesis in problems of packet routing. We briefly review both approaches here.

The best approximation algorithms to date for job shop scheduling have primarily appeared in the Soviet literature and are based on a beautiful connection to geometric arguments. This

approach was independently discovered by Belov and Stolin [2] and Sevast'yanov [11] as well as by Fiala [3]. This approach typically produces schedules for which the length can be bounded by $P_{\max} + q(m, m_{\max})p_{\max}$, where $q(\cdot,\cdot)$ is a polynomial, and $p_{\max} = \max_{ij} p_{ij}$ is the maximum operation length. For the job shop problem, Sevast'yanov [12] gave a polynomial-time algorithm that delivered a schedule of length at most $P_{\max} + O(m_{\max}^3 m^2)p_{\max}$. The bounds obtained in this way do not give good worst-case relative error bounds, and even for the special case of the *flow shop problem*, where each job has a single operation on each machine and for each job the operations must be done in the same order, the best known algorithms delivered solutions of length $\Omega(mC^*_{\max})$.

In a different vein, Leighton, Maggs and Rao [7] have proposed the following model for the routing of packets in a network: find paths for the packets and then schedule the transmission of the packets along these paths so that no two packets traverse the same edge simultaneously. The primary objective is to minimize the time by which all packets have been delivered to their destination.

It is easy to see that the problem considered by Leighton, Maggs and Rao is simply the job shop scheduling problem with each processing time $p_{ij} = 1$. They also added the restriction that each path does not traverse any edge more than once, or in scheduling terminology, each job has at most one operation on each machine. This restriction of the job shop problem remains (strongly) $\mathcal{NP}$-hard. The main result of Leighton, Maggs and Rao was to show that for their special case of the job shop problem, there always exists a schedule of length $O(P_{\max} + \Pi_{\max})$. Unfortunately, this is not an algorithmic result, as it relies on a nonconstructive probabilistic argument based on the Lovász Local Lemma. They also obtained a randomized algorithm that delivers a schedule of length $O(P_{\max} + \Pi_{\max} \log n)$, with high probability. In this paper, we will show how their techniques can be generalized to handle the general job shop problem, as well as several related scheduling problems.

We also give a deterministic version of the job shop scheduling algorithm.

Theorem 1.2 There exists a deterministic polynomial-time algorithm for job shop scheduling which finds a schedule of length $O(\log^2(m \cdot m_{\max})C^*_{\max})$.

This is the first polylogarithmic performance guarantee for a deterministic polynomial-time approximation algorithm for either job shop scheduling or for the special case of flow shop scheduling. Note that if each job must be processed on each machine at most once, the $m_{\max}$ factor can be deleted for this, and all other performance guarantees in this paper. As a corollary, we also obtain a deterministic version of the randomized algorithm of Leighton, Maggs and Rao. Our results rely on results of Raghavan and Thompson [10] and Raghavan [8] to approximate certain integer packing problems.

In contrast to this, the only "negative" result known for any shop scheduling problem is that the existence of a fully polynomial approximation scheme would imply that $\mathcal{P} = \mathcal{NP}$, due to the fact that these problems are strongly $\mathcal{NP}$-hard.

Our techniques can also be made to apply to two important generalizations of the job shop problem. The first is *dag scheduling*, where each job consists of a set of operations on different machines which must be processed in an order consistent with a particular partial order. (For job shop scheduling, this partial order is always a chain, while for flow shop the partial order is the same chain for all jobs.) One can further generalize the problem to the situation where, rather than having m different machines, there are m' types of machines, and for each type, there are a specified number of identical machines; each operation, rather than being assigned to one machine, may be processed on any machine of the appropriate type. These problems have

significant practical importance, since in real-world shops we would expect that a job need not follow a strict total order and that the shop would have more than one copy of many of their machines.

Finally, we give some extensions to these results, including $\mathcal{RNC}$ approximation algorithms for all the scheduling models mentioned above, and a $(2+\epsilon)$-approximation algorithm for job shop scheduling with a fixed number of machines.

While all of the algorithms that we give are polynomial-time, they are all rather inefficient. Most rely on the algorithms of Sevast'yanov; for example, his algorithm for job shop scheduling takes $O(m^2 m_{\max}^2 n^2)$ time. Furthermore, the deterministic versions rely on linear programming. As a result, we will not refer explicitly to running times throughout the remainder of this paper.

2 The Basic Algorithm

In this section we extend the technique due to Leighton, Maggs and Rao [7] of assigning random delays to jobs to the general case of non-preemptive job shop scheduling.

A valid schedule assigns at most one job to a particular machine at any time, and schedules each job on at most one machine at any time. Our approach will be to first create a schedule that obeys only the second constraint, and then build from this a schedule that satisfies both constraints and is not much longer. The outline of the strategy follows:

1. Define the *oblivious* schedule, where each job starts running at time 0 and runs continuously until all of its operations have been completed. This schedule is of length $\Pi_{\max}$, but there may be times when more than one job is assigned to a particular machine.

2. Perturb this schedule by delaying the start of the first operation of each job by a random integral amount chosen uniformly in $[0, P_{\max}]$. The resulting schedule, with high probability, has no more than $O(\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})})$ jobs assigned to any machine at any time.

3. "Spread" this schedule so that at each point in time all operations currently being processed have the same size, and then "flatten" this into a schedule that has at most one job per machine at any time.

This strategy is very similar to the one used by Leighton, Maggs and Rao for the special case of unit-length operations. Whereas Step 2 differs in only a few technical details, the essential difficulty in obtaining the generalization is in Step 3. For the analysis of Step 2, we assume that $p_{\max}$ is bounded above by a polynomial in n and $m_{\max}$; in the next section we will show how to remove this assumption. As is usually the case, we assume that $n \geq m$; analogous bounds can be obtained when this is not true.

Lemma 2.1 Given a job shop instance in which $p_{\max}$ is bounded above by a polynomial in n and $m_{\max}$, the strategy of delaying each job an initial integral amount chosen randomly and uniformly from $[0, P_{\max}]$ and then processing the jobs in sequence will yield a schedule that is of length at most $P_{\max} + \Pi_{\max}$ and, with high probability, has no more than $O(\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})})$ jobs scheduled on any machine during any unit of time.

Proof: Fix a time t and a machine M_i; consider $p = \text{Prob}[\text{at least } \tau \text{ units of processing are scheduled on machine } i \text{ at time } t]$. There are at most $\binom{P_{\max}}{\tau}$ ways to choose τ units of processing from all those required on M_i. The probability that a particular one of those τ units is on

machine M_i at time t is at most $\frac{1}{P_{\max}}$ since there are $P_{\max}$ different places it might be at time t and at most one is on M_i. If all τ units are from different jobs then the probability that they are all on M_i at time t is at most $(\frac{1}{P_{\max}})^\tau$ since the delays are chosen independently. Otherwise, the probability that all τ are there is 0, since it is impossible. Therefore $p \leq \binom{P_{\max}}{\tau}(\frac{1}{P_{\max}})^\tau \leq (\frac{eP_{\max}}{\tau})^\tau(\frac{1}{P_{\max}})^\tau \leq (\frac{e}{\tau})^\tau$.

If $\tau = k\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})}$ then $p < (n \cdot m_{\max})^{-(k-1)}$. To bound the probability that *any* machine at *any* time has more than $k\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})}$ jobs using it, multiply p by $\Pi_{\max} + P_{\max}$ for the number of time units in the schedule, and by m for the number of machines. Since we have assumed that $p_{\max}$ is bounded by a polynomial in n and $m_{\max}$, $\Pi_{\max} + P_{\max}$ is as well; choosing k large enough yields that, with high probability, no more than $k\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})}$ jobs are scheduled for any machine during any unit of time. ■

In the special case of unit-length operations treated by Leighton, Maggs and Rao, a schedule $\mathcal{S}$ of length L that has at most c jobs scheduled on any machine at any unit of time can trivially be "flattened" into a valid schedule of length $c \cdot L$ by replacing one unit of $\mathcal{S}$'s time with c units of time in which we run each of the jobs that was scheduled for that time unit. (See Figure 1.) For *preemptive* job shop scheduling, where the processing of an operation may be interrupted, each unit of an operation can be treated as a unit-length operation and a schedule that has multiple operations scheduled simultaneously on a machine can easily be flattened into a valid schedule. This is not possible for *non-preemptive* job shop scheduling, and in fact it seems to be more difficult to flatten the schedule in this case. We give an algorithm that takes a schedule of length L with at most c operations scheduled on one machine at any time and produces a schedule of length $O(Lc \log p_{\max})$.

Lemma 2.2 Given a schedule $\mathcal{S}_0$ of length L that has at most c jobs scheduled on one machine during any unit of time, there exists a polynomial-time algorithm that produces a valid schedule of length $O(Lc \log p_{\max})$.

Proof: To begin, we round up each processing time p_{ij} to the next power of 2 and denote the rounded times by p'_{ij}; that is, $p'_{ij} = 2^{\lceil \log_2 p_{ij} \rceil}$. Let $p'_{\max} = \max_{ij} p'_{ij}$. From $\mathcal{S}_0$, it is easy to obtain a schedule $\mathcal{S}$ that uses the modified p'_{ij} and is at most twice as long as $\mathcal{S}_0$; furthermore, an optimal schedule for the new problem is no more than twice as long as an optimal schedule for the original problem.

A *block* is an interval of a schedule with the property that each operation which begins during that interval is of length no more than that of the entire interval. (Note that this does not mean that the operation finishes within the interval.) We can divide $\mathcal{S}$ into $\lceil \frac{L}{p'_{\max}} \rceil$ consecutive blocks of size $p'_{\max}$. We will give a recursive algorithm that reschedules – "spreads" – each block of size p (where p is a power of 2) into a sequence of schedule *fragments* of total length $p \log p$, so that the resulting schedule has the property that at any time all operations scheduled are of the same length. This algorithm takes advantage of the fact that if an operation of length p is scheduled to begin in a block of size p, that job is not scheduled on any other machine until after this block. Therefore, that operation can be scheduled to start after all of the smaller operations in the block finish.

To reschedule a block B of size $p'_{\max}$, we first construct the final fragment (which is of length $p'_{\max}$), and then construct the preceding fragments by recursive calls of the algorithm. For each operation of length $p'_{\max}$ that begins in B, reschedule that operation to start at the beginning of the final fragment, and delete it from B. Now each operation that still starts in B is of length

at most $p'_{\max}/2$, so B can be subdivided into two blocks, B_1 and B_2, each of size $p'_{\max}/2$, and we can recurse on each. See Figure 2.

The recurrence equation that describes the total length of the fragments produced from a block of size T is $f(T) = 2f(\frac{T}{2}) + T; f(1) = 1$. Thus $f(T) = \Theta(T \log T)$, and each block B in $\mathcal{S}$ of size $p'_{\max}$ is spread into a schedule of length $p'_{\max} \log p'_{\max}$. By spreading the schedule $\mathcal{S}$, we produce a new schedule $\mathcal{S}'$ that satisfies the following conditions:

1. At any time in $\mathcal{S}'$, all operations scheduled are of the same length; furthermore, two operations either start at the same time or do not overlap.
2. If $\mathcal{S}$ has at most c jobs scheduled on one machine at any time, then this must hold for $\mathcal{S}'$ as well.
3. $\mathcal{S}'$ schedules a job on at most one machine at any time.
4. $\mathcal{S}'$ does not schedule the ith operation of job J_j until the first $i-1$ are completed.

Condition 1 is satisfied by each pair of operations on the same machine by the definition of spreading, and by each pair of operations on different machines because the division of time into fragments is the same on all machines. To prove condition 2, note that operations of length T that are scheduled at the same time on the same machine in the expanded schedule started in the same block of size T on that machine. Since they all must have been scheduled during the last unit of time of that block, there can be at most c of them.

To prove condition 3, note that if a job is scheduled by $\mathcal{S}'$ on two machines simultaneously that means that it must have been scheduled by $\mathcal{S}$ to start two operations of length T in the same block of time T on two different machines. This means it was scheduled by $\mathcal{S}$ on two machines during the last unit of time of that block, which violates the properties of $\mathcal{S}$.

Finally we verify condition 4 by first noting that if two operations of a job are in different blocks of size $p'_{\max}$ in $\mathcal{S}$ then they are certainly rescheduled in the correct order. Therefore it suffices to focus on what happens within the schedule produced from one block. Within a block an operation is only rescheduled to the final fragment when it is the last operation for that job in that block. Therefore $\mathcal{S}'$ does not schedule the ith operation of job J_j until the first $i-1$ are completed.

The schedule $\mathcal{S}'$ can easily be flattened to a schedule that obeys the constraint of one job per machine at any time, since c operations of length T that start at the same time can just be executed one after the other in total time cT. Note that since what we are doing is effectively synchronizing the entire schedule block by block, it is important when flattening the schedule to make each machine wait enough time for all machines to process all operations of that fragment length, even if some machines have no operations of that length in that fragment.

The schedule $\mathcal{S}'$ was of length $L \log p'_{\max}$; therefore the flattened schedule is of length $Lc \log p'_{\max}$. ■

3 Reducing the Problem

In the previous section we showed how to produce, with high probability, a schedule of length $O((P_{\max}+\Pi_{\max})\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})} \log p_{\max})$, under the assumption that $p_{\max}$ was bounded above by a polynomial in n and m. Since $P_{\max}+\Pi_{\max} = O(\max(P_{\max}, \Pi_{\max}))$ this schedule is within a factor of $O(\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})} \log p_{\max})$ of optimality. In this section, we will remove the assumption that

$p_{\max}$ is bounded above by a polynomial in n and $m_{\max}$ by showing that we can reduce the general problem to that special case while only sacrificing a constant factor in the approximation. This yields an $O(\frac{\log^2(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})})$–approximation algorithm. Furthermore, n need not be polynomially bounded in m and $m_{\max}$; we will prove a stronger result and show that we can reduce the job shop problem to the case where n is polynomially bounded in m and $m_{\max}$, while changing the performance guarantee by a constant.

3.1 Reducing $p_{\max}$

First we will show that we can reduce the problem to one where $p_{\max}$ is bounded by a polynomial in n and $m_{\max}$. Let $\omega = |\mathcal{O}|$ be the total number of required operations. Note that $\omega = O(n \cdot m_{\max})$. Round down each p_{ij} to the nearest multiple of $p_{\max}/\omega$, denoted by p'_{ij}. Now there are at most ω distinct values of p'_{ij} and they are all multiples of $p_{\max}/\omega$. Therefore we can treat the p'_{ij} as integers in $\{0, \ldots, \omega\}$; a schedule for this problem can be trivially rescaled to a schedule $\mathcal{S}'$ for the actual p'_{ij}. Let L denote the length of $\mathcal{S}'$. We claim that $\mathcal{S}'$ for this reduced problem can be interpreted as a schedule for the original operations that will be of length at most $L + p_{\max}$. When we adjust the p'_{ij} up to the original p_{ij}, we add an amount that is at most $p_{\max}/\omega$ to each p'_{ij}. Since the length of a schedule is determined by a critical path through the operations and there are ω operations, we add a total amount of at most $p_{\max}$ to the length of the schedule; thus the new schedule is of length at most $L + p_{\max} \leq L + C^*_{\max}$. Therefore we have rounded a general instance $\mathcal{I}$ of the job shop problem to an instance $\mathcal{I}'$ that can be treated as having $p_{\max} = O(n \cdot m_{\max})$; further, a schedule for $\mathcal{I}'$ yields a schedule for $\mathcal{I}$ that is no more than $C^*_{\max}$ longer. Thus we have shown:

Lemma 3.1 There exists a polynomial-time algorithm which transforms any instance of the job shop scheduling problem into one with $p_{\max} = O(n \cdot m_{\max})$ with the property that a schedule for the modified instance of length $kC^*_{\max}$ can be converted in polynomial time to an instance of length $(k+1)C^*_{\max}$.

3.2 Reducing the Number of Jobs

To reduce an arbitrary instance of job shop scheduling to one with a number of jobs polynomial in m and $m_{\max}$ we divide the jobs into big and small jobs. We say that job J_j is *big* if it has an operation of length more than $\frac{P_{\max}}{2m^2 m^3_{\max}}$; otherwise we call the job *small*. For the instance consisting of just the short jobs, let $P'_{\max}$ and $p'_{\max}$ denote the maximum machine load and operation length, respectively. Using the algorithm of [12] described in the introduction, we can, in time polynomial in the input size, produce a schedule of length $P'_{\max} + 2m^2 m^3_{\max} p'_{\max}$ for this instance. Since $p'_{\max}$ is at most $\frac{P_{\max}}{2m^2 m^3_{\max}}$ and $P'_{\max} \leq P_{\max}$, we get a schedule that is of length no more than $2P_{\max}$. Thus, an algorithm that produces a schedule for the long jobs that is within a factor of k of optimal will yield a $(k+2)$-approximation algorithm. Note that there can be at most $2m^3 m^3_{\max}$ long jobs, since otherwise there would be more than $mP_{\max}$ units of processing to be divided amongst m machines, which contradicts the definition of $P_{\max}$. Thus we have shown:

Lemma 3.2 There exists a polynomial-time algorithm which transforms any instance of the job shop scheduling problem into one with $O(m^3 \cdot m^3_{\max})$ jobs with the property that a schedule for the modified instance of length $kC^*_{\max}$ can be converted in polynomial time to an instance of length $(k+2)C^*_{\max}$.

From the results of the previous two sections we can conclude that:

Theorem 3.3 There exists a polynomial-time randomized algorithm for job shop scheduling, that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2(m \cdot m_{\max})}{\log\log(m \cdot m_{\max})} C^*_{\max})$.

Proof: In Section 2 we showed how to produce a schedule of length $O((P_{\max}+\Pi_{\max})\frac{\log(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})} \log p_{\max})$ under the assumption that $p_{\max}$ was bounded above by a polynomial in n and $m_{\max}$. From Lemmas 3.1 and 3.2 we know that we can reduce the problem to one where n and $p_{\max}$ are polynomial in m and $m_{\max}$, while adding only a constant to the factor of approximation. Since now $\log p_{\max} = O(\log(m \cdot m_{\max}))$ and $\log n = O(\log(m \cdot m_{\max}))$ our algorithm produces a schedule of length $O(\frac{\log^2(m \cdot m_{\max})}{\log\log(m \cdot m_{\max})} C^*_{\max})$. ■

Note that when $m_{\max}$ is bounded by a polynomial in m the bound only depends on m. In particular, this implies the following corollary:

Corollary 3.4 There exists a polynomial-time randomized algorithm for flow shop scheduling, that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2 m}{\log\log m} C^*_{\max})$.

Except for the use of Sevast'yanov's algorithm, all of these techniques can be carried out in $\mathcal{RNC}$. We assign one processor to each operation. The rounding in the proof of Lemma 2.2 can be done in $\mathcal{NC}$. We set the random delays and inform each processor about the delay of its job. By summing the values of p_{ij} for all of its job's operations, each processor can calculate where its operation is scheduled with the delays and then where it is scheduled in the recursively spread out schedule. These sums can be calculated via parallel prefix operations. With simple $\mathcal{NC}$ techniques we can assign to each operation a rank among all those operations that are scheduled to start at the same time on its machine, and thus flatten the spread out schedule to a valid schedule.

Corollary 3.5 There exists a $\mathcal{RNC}$ algorithm for job shop scheduling, that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2 nm_{\max}}{\log\log nm_{\max}} C^*_{\max})$.

3.3 A Fixed Number of Machines

It is interesting to note that Sevast'yanov's algorithm for the job shop problem can be viewed as a $(1 + m^2 m^3_{\max})$-approximation algorithm, so that when m and $m_{\max}$ are constant, this is a $O(1)$-approximation algorithm; that is, it delivers a solution within a constant factor of the optimum. The technique of partitioning the set of jobs by size can be applied to give much a better performance guarantee in this case. Now call a job J_j *big* if there is an operation O_{ij} with $p_{ij} > \epsilon P_{\max}/(m^2 \cdot m^3_{\max})$, where ϵ is an arbitrary positive constant. Note that there are at most $\frac{m^3 \cdot m^3_{\max}}{\epsilon}$ big jobs, and since m, $m_{\max}$ and ϵ are fixed, this is a constant.

Now use Sevast'yanov's algorithm to schedule all of the small jobs. The resulting schedule will be of length at most $(1+\epsilon)C^*_{\max}$. There are only a constant (albeit a huge constant) number of ways to schedule the big jobs. Therefore the best one can be selected in polynomial time and executed after the schedule of the short jobs. The additional length of this part is no more than $C^*_{\max}$.

Thus we have shown:

Theorem 3.6 For the job shop scheduling problem where both m and $m_{\max}$ are fixed, there is a polynomial-time algorithm that produces a schedule of length $\leq (2+\epsilon)C^*_{\max}$.

3.4 Applications to More General Scheduling Problems

The fact that the quality of our approximations is based solely on the lower bounds $P_{\max}$ and $\Pi_{\max}$ makes it quite easy to extend our techniques to the more general problem of *dag scheduling*. We define $P_{\max}$ and $\Pi_{\max}$ exactly the same way, and $\max(P_{\max}, \Pi_{\max})$ remains a lower bound for the length of any schedule. We can convert this dag scheduling problem to a job shop problem by selecting for each job an arbitrary total order that is consistent with its partial order. $P_{\max}$ and $\Pi_{\max}$ have the same values for both problems. Therefore, a schedule of length $\rho \cdot (P_{\max} + \Pi_{\max})$ for this job shop instance is a schedule for the original dag scheduling instance of length $O(\rho C^*_{\max})$.

A further generalization to which our techniques apply is where, rather than m different machines, we have m' types of machines, and for each type we have a specified number of identical machines of that type. Instead of requiring an operation to run on a particular machine, an operation now only has to run on one of these identical copies. $\Pi_{\max}$ remains a lower bound on the length of any schedule for this problem. $P_{\max}$, which was a lower bound for the job shop problem must be replaced, since we do not have a specific assignment of operations to machines, and the sum of the processing times of all operations assigned to a type is *not* a lower bound. Let S_i, $i = 1, \ldots m'$, denote the sets of identical machines, and let $P(S_i)$ be the sum of the lengths of the operations which run on S_i. Our strategy is to convert this to a job shop problem by assigning operations to specific machines in such a way that the maximum machine load is within a constant factor of the fundamental lower bounds for this problem. To obtain a lower bound on the maximum machine load, note that the best we could do would be to evenly distribute the operations across machines in a set, thus

$$P_{\text{avg}} = \max_{S_i} \frac{P(S_i)}{|S_i|}$$

is certainly a lower bound on the maximum machine load. Furthermore, we can not split operations, so $p_{\max}$ is also a lower bound. We will now describe how to assign operations to machines so that the maximum machine load of the resulting job shop scheduling problem is at most $2P_{\text{avg}} + p_{\max}$. A schedule for the resulting job shop problem of length $\rho \cdot (P_{\max} + \Pi_{\max})$ yields a solution for the more general problem of length $O(\rho \cdot (P_{\max} + \Pi_{\max}))$. This reduction to the job shop problem is much simpler than the one used by Sevast'yanov in his algorithm for dag scheduling with identical copies of machines.

For each operation O_{ij} to be processed by a machine in S_k, if $p_{ij} \geq P(S_k)/|S_k|$, assign O_{ij} to one machine in S_k. There are certainly enough machines in S_k to do this and this contributes at most $p_{\max}$ to the maximum machine load. Those operations not yet assigned are each of length at most $P(S_k)/|S_k|$ and have total length $\leq P(S_k)$. Therefore, these can be assigned easily to the remaining machines so that less than $2P(S_k)/S_k$ processing units are assigned to each machine. Combining these two bounds, we get an upper bound on the maximum machine load of $2P_{\text{avg}} + p_{\max}$ which is within a constant factor of the lower bound of $\max\{P_{\text{avg}}, p_{\max}\}$.

Theorem 3.7 There exists a polynomial-time randomized algorithm for *dag scheduling with identical copies of machines* that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2(m \cdot m_{\max})}{\log\log(m \cdot m_{\max})} C^*_{\max})$.

Corollary 3.8 There exists an $\mathcal{RNC}$ algorithm for *dag scheduling with identical copies of machines* that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2(n \cdot m_{\max})}{\log\log(n \cdot m_{\max})} C^*_{\max})$.

4 A Deterministic Approximation Algorithm

In this section, we "derandomize" the results of the previous sections, i.e., we give a deterministic polynomial-time algorithm which finds a schedule of length $O(\log^2(m \cdot m_{\max})C^*_{\max})$. Of all the components of the algorithm of Theorem 3.3, the only step which is not already deterministic is the step which chooses a random initial delay for each job and then proves that, with high probability, no machine is assigned too many jobs at any one time. In particular, the reduction to the special case in which n and $p_{\max}$ are bounded by a polynomial in m and $m_{\max}$ is entirely deterministic, and so we can focus on that case alone. We will give an algorithm which deterministically assigns delays to each job so as to produce a schedule in which each machine has $O(\log(m \cdot m_{\max}))$ jobs running at any one time. We then apply Lemma 2.2 to produce a schedule of length $O(\log^2(m \cdot m_{\max})C^*_{\max})$. Note that the $O(\log(m \cdot m_{\max}))$ jobs per machine is not as good as the probabilistic bound of $O(\frac{\log(m \cdot m_{\max})}{\log\log(m \cdot m_{\max})})$; we do not know how to achieve this deterministically. However, by a proof nearly identical to that of Lemma 2.1, we can show that in order to achieve this weaker bound on the number of jobs per machine, we now only need to choose delays in the range $[0, P_{\max}/\log(m \cdot m_{\max})]$. In fact, the reduced range of delays yields a schedule of length $O(\Pi_{\max}\log^2(m \cdot m_{\max}) + P_{\max}\log(m \cdot m_{\max}))$ which is within an $O(\log(m \cdot m_{\max}))$ factor of optimal if $\Pi_{\max} = O(\frac{P_{\max}}{\log(m \cdot m_{\max})})$.

Our approach to solving this problem is to frame it as a *vector selection* problem and then apply techniques developed by Raghavan and Thompson [9, 10] and Raghavan [8] which find constant factor approximations to certain "packing" integer programs. The approach is to formulate the problem as a $\{0,1\}$–integer program, solve the linear programming relaxation, and then randomly round the solution to an integer solution.

For certain types of problems this yields provably good approximations with high probability [9, 10]. Furthermore, for many of the problems for which there are approximations with high probability, the algorithm can be derandomized. Raghavan [8] has shown how to do this for many of these problems by essentially setting the random bits one at a time.

We now state the problem formally:

Problem 4.1 Deterministically assign delays in the range $[0, P_{\max}/\log(m \cdot m_{\max})]$ so as to produce a schedule with no more than $O(\log(m \cdot m_{\max}))$ jobs on any machine at any time

Lemma 4.2 Problem 4.1 can be solved in deterministic polynomial time.

Proof: Since we introduce delays in the range $[0, P_{\max}/\log(m \cdot m_{\max})]$, the resulting schedule has length $\ell = \Pi_{\max} + \frac{P_{\max}}{\log(m \cdot m_{\max})}$. We can represent the processing of a job j with a given initial delay d by an $(\ell \cdot m)$–length $\{0,1\}$–vector where each position corresponds to a machine at a particular time. The position corresponding to machine M_i and time t is 1 if M_i is processing job J_j at time t, and 0 otherwise. For each job J_j and each possible delay d, there is a vector $V_{j,d}$ which corresponds to assigning delay d to J_j.

Let λ_j be the set of vectors $\{V_{j,1}, \ldots, V_{j,d_{\max}}\}$, where $d_{\max} = P_{\max}/\log(m \cdot m_{\max})$, and let $V_{j,k}(i)$ be the i^{th} component of $V_{j,k}$. Given the set $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ of sets of vectors, our problem can be stated as the problem of choosing one vector from each λ_j (denoted V^*_j), such that $||\sum_{j=1}^{n} V^*_j||_\infty = O(\log(m \cdot m_{\max}))$, i.e., at any time on any machine, the number of jobs using that machine is $O(\log(m \cdot m_{\max}))$.

As in [8], we can reformulate this as a $\{0,1\}$–integer program. Let $x_{j,k}$ be the indicator variable used to indicate whether $V_{j,k}$ is selected from λ_j. Consider the integer program (IP) that assigns $\{0,1\}$ values to the variables $x_{j,k}$ to minimize W subject to the constraints:

$$\sum_{k=1}^{d_{\max}} x_{j,k} = 1, \quad j = 1, \ldots, n,$$

$$\sum_{j=1}^{n} \sum_{k=1}^{d_{\max}} x_{j,k} V_{j,k}(i) \leq W, \quad i = 1, \ldots, \ell \cdot m.$$

Let W_{OPT} be the optimum value of W, which is the maximum number of jobs that ever use a machine at any given time. We already know, by Lemma 2.1, that $W_{\mathrm{OPT}} = O(\log(m \cdot m_{\max}))$, so if we could solve this integer program optimally we would be done. However, the problem is $\mathcal{NP}$-hard. Instead, we rely on the following theorem which is immediate from the results in [8] and [10].

Theorem 4.3 [8, 10] A feasible solution to (IP) with $W = O(W_{\mathrm{OPT}} + \log(m \cdot m_{\max}))$ can be found in polynomial time.

We then apply Lemma 2.2 and obtain the following result:

Theorem 4.4 There exists a deterministic polynomial-time algorithm which finds a schedule of length $O(\log^2(m \cdot m_{\max})\, C^*_{\max})$.

5 Conclusions and Open Problems

We have given the first polynomial-time polylog-approximation algorithms for minimizing the maximum completion time for the problems of job shop scheduling, flow shop scheduling, dag scheduling and a generalization of dag scheduling in which there are groups of identical machines.

One particularly simple special case of dag scheduling can be obtained if the partial order for each job is empty; in other words, each job consists of a number of operations which may be performed in any order. This is called the *open shop* problem, and it is traditional in the scheduling literature to focus on the case when each job is processed on each machine at most once (since operations on the same machine can be coalesced).

A consequence of our results is the following observation about the structure of shop scheduling problems. Assume we have a set of jobs which need to run on a set of machines. We know that any schedule for the associated open shop problem must be of length $\Omega(P_{\max} + \Pi_{\max})$. Furthermore, we know that no matter what type of partial ordering we impose on the operations of each job we can produce a schedule of length $O((P_{\max} + \Pi_{\max})\frac{\log^2 m}{\log\log m})$. Hence for any instance, *the gap between the best open shop schedule and the best dag schedule is at most* $O(\frac{\log^2 m}{\log\log m})$.

On the other hand, there does exist a schedule of length $O(\Pi_{\max} + P_{\max})$ for the open shop problem. Consider the simple greedy algorithm that, whenever a machine is idle, assigns to it any job that has not yet been processed on that machine and is not currently being processed on another machine. Anná Racsmány has observed that the greedy algorithm delivers a schedule of length at most $P_{\max} + (m-1)p_{\max}$ [1]. We can adapt her proof to show that, in fact, the greedy algorithm delivers a schedule that is of length less than $P_{\max} + \Pi_{\max} \leq 2C^*_{\max}$. Consider the machine M_k that finishes last in the greedy schedule; this machine is active sometimes, idle sometimes, and finishes by completing some job J_j. Since the schedule is greedy, whenever M_k is idle, J_j is being processed by some other machine, and so the idle time is at most $\sum_{M_i \neq M_k} p_{ij} < \Pi_{\max}$. Thus, machine M_k is processing for at most $P_{\max}$ units of time and is idle for less than

$\Pi_{\max}$ units of time; hence $C_{\max} < P_{\max} + \Pi_{\max}$. Fiala [4] has also shown that if $P_{\max} \geq (16m \log m + 21m)p_{\max}$, then $C^*_{\max}$ is just $P_{\max}$, and there is a polynomial-time algorithm to find an optimal schedule.

We have seen that in two interesting special cases, job shop scheduling with unit-length operations and open shop scheduling, there is a schedule of length $O(P_{\max} + \Pi_{\max})$, and so the major open question left unresolved by this paper is:

- Does there exist an $O(P_{\max} + \Pi_{\max})$ schedule for the general job or flow shop scheduling problem? If so, when can it be found in polynomial time?

Beyond this, there are a number of interesting questions raised by this work, including

- Do there exist parallel algorithms that achieve the approximations of our sequential algorithms? For the general job shop problem this seems hard, since we rely heavily on the algorithm of Sevast'yanov. For open shop scheduling, however, a simple sequential algorithm achieves a factor of 2, whereas the best $\mathcal{NC}$ algorithm that we have achieves only an $O(\log n)$-approximation. As a consequence of the results above, all one would need to do is to produce any greedy schedule.
- Are there simple variants of the greedy algorithm for open shop scheduling that achieve better performance guarantees? For instance, how good is the algorithm that always selects the job with the maximum total (remaining) processing time? Is there a polynomial approximation scheme?
- Our algorithms, while polynomial-time algorithms, are inefficient. Are there significantly more efficient algorithm which have the same performance guarantees?

Acknowledgments

We are grateful to David Williamson for working with us on the early parts of this research, and to Imre Bárány, Tom Leighton, Bruce Maggs, and Yishay Mansour for many helpful discussions. We thank Jim Orlin for the observation about the gap between open shop and dag scheduling.

References

[1] Imre Bárány and Tibor Fiala. Többgépes ütemezési problémák közel optimális megoldása. *Szigma–Mat.–Közgazdasági Folyóirat*, 15:177–191, 1982.

[2] I.S. Belov and Ya. N. Stolin. An algorithm in a single path operations scheduling problem. In *Mathematical Economics and Functional Analysis [In Russian]*, pages 248–257. Nauka, Moscow, 1974.

[3] Tibor Fiala. Közelítő algorithmus a három gép problémára. *Alkalmazott Matematikai Lapok*, 3:389–398, 1977.

[4] Tibor Fiala. An algorithm for the open-shop problem. *Mathematics of Operations Research*, 8(1):100–109, 1983.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[6] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinooy Kan, and David B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical Report BS-R8909, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, 1989. To appear in Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory.

[7] Tom Leighton, Bruce Maggs, and Satish Rao. Universal packet routing algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[8] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[9] P. Raghavan and C. D. Thompson. Provably good routing in graphs: regular arrays. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 79–87, 1985.

[10] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365 – 374, 1987.

[11] S. V. Sevast'yanov. On an asymptotic approach to some problems in scheduling theory. In *Abstracts of papers at 3rd All-Union Conf. of Problems of Theoretical Cybernetics [in Russian]*, pages 67–69. Inst. Mat. Sibirsk. Otdel. Akad. Nauk SSSR, Novosibirsk, 1974.

[12] S.V. Sevast'yanov. Efficient construction of schedules close to optimal for the cases of arbitrary and alternative routes of parts. *Soviet Math. Dokl.*, 29(3):447–450, 1984.

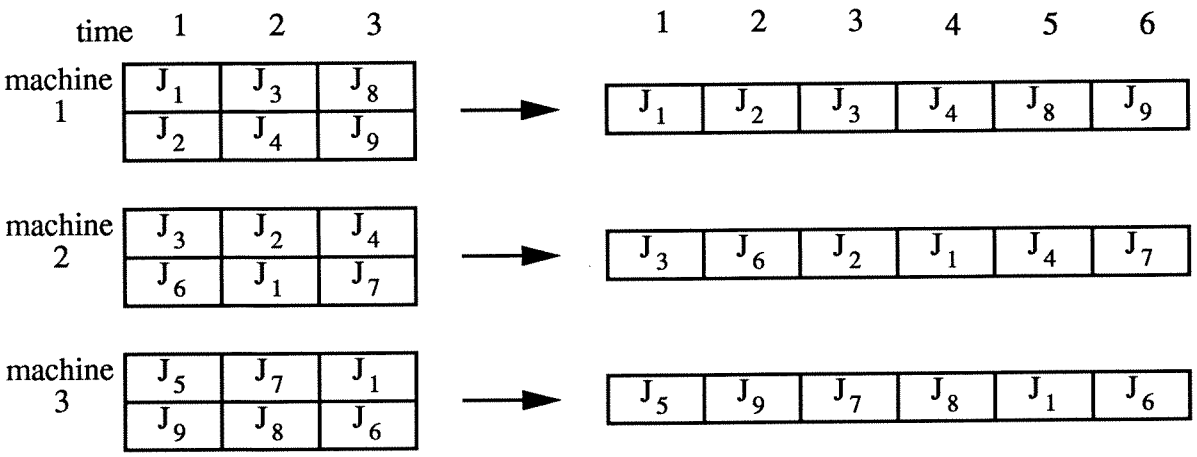

Figure 1: Flattening a schedule in the case with unit length operations.

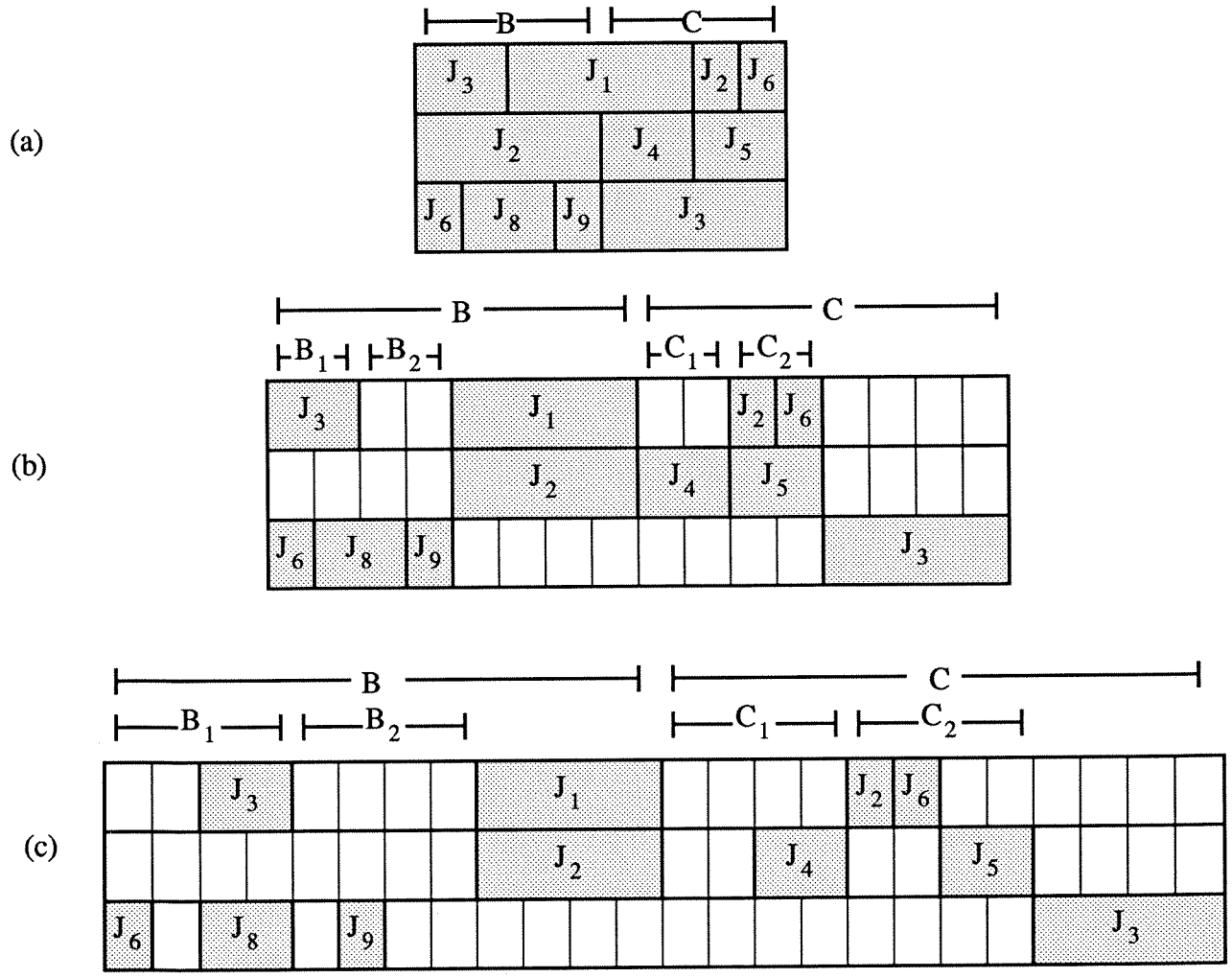

Figure 2: (a) The initial greedy schedule of length 8. $p'_{\max} = 4$. (b) The first level of spreading. All jobs of length 4 have been put in the final fragments. We must now recurse on B_1 and B_2 with $p'_{\max} = 2$. (c) The final schedule of length $8 \log 8 = 24$.