Improved Approximation Algorithms for Shop Scheduling Problems*

David B. Shmoys† Cornell University

Clifford Stein‡ Dartmouth College

Joel Wein§ Polytechnic University

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; the objective 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. Our algorithms also extend to the more general case where a job is given not by a linear ordering of the machines on which it must be processed but by an arbitrary partial order. Comparable bounds can also be obtained when there are m' types of machines, a specified number of machines of each type, and each operation must be processed on one of the machines of a specified type, as well as for the problem of scheduling unrelated parallel machines subject to chain precedence constraints.

Key Words: scheduling, approximation algorithms
AMS(MOS) subject classifications: 68A10, 68Q25, 90B35, 68R99

*Research partially supported by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun, by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS89-20550, and by DARPA Contract N00014-89-J-1988. A preliminary version of this paper appeared in the proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, January 1991.

†Address: School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853. Electronic mail: shmoys@cs.cornell.edu

‡Additional support provided by an AT&T graduate fellowship. Address: Department of Mathematics and Computer Science, 6188 Bradley Hall, Dartmouth College, Hanover, NH 03755-3551. Electronic mail: cliff@bondcliff.dartmouth.edu

§Additional support provided by an ARO graduate fellowship. Address: Department of Computer Science, Polytechnic University, Five MetroTech Center, Brooklyn, NY 11201, Electronic mail: wein@mem.poly.edu

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; a job may have 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 [6, 7]. 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 μ, the maximum number of operations in a job. 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 [7]. Furthermore, several benchmark instances with 15 jobs, 15 machines and 225 operations are too hard to be solved by known methods and were posed as open problems by Applegate and Cook [1].

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 maximum completion time of a job 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 polynomial-time polylogarithmic approximation algorithm for the job shop scheduling problem.

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

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 $\mathcal{O} = \{O_{ij} | i = 1, \ldots, \mu_j, j = 1, \ldots, n\}$ of operations, where κ_{ij} indexes the machine which must process operation O_{ij}. Thus m is the number of machines, n is the number of jobs, μ_j is the number of operations of job J_j, and $\mu = \max_j \mu_j$. Operation O_{ij} is the ith operation of J_j; it requires processing time on a given machine $M_k \in \mathcal{M}$, where $k = \kappa_{ij}$, for an uninterrupted period of a given length p_{ij}. (In other words, this is a *non-preemptive* model; a model in which operations may be interrupted and resumed at a later time 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, and denote by $P_{\max}$. Furthermore, each machine must process all of its operations, and so $C^*_{\max}$ must be at least

$\max_{M_k} \sum_{\kappa_{ij}=k} p_{ij}$, which we shall call the *maximum machine load* of the instance, and denote by $\Pi_{\max}$.

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 [3] and Sevast'yanov [16] as well as by Fiala [4]. This approach typically produces schedules for which the length can be bounded by $\Pi_{\max} + q(m, \mu)p_{\max}$, 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 [17, 18] gave a polynomial-time algorithm that delivered a schedule of length at most $\Pi_{\max} + O(m\mu^3)p_{\max}$. The bounds obtained in this way do not give good worst-case relative error bounds. 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 [8] 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 $O(\Pi_{\max} + P_{\max})$. 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 $O(\Pi_{\max} + P_{\max} \log n)$, 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 $O(\log^2(m\mu)C^*_{\max})$.

This is the first polylogarithmic performance guarantee for a deterministic polynomial-time approximation algorithm for job shop scheduling; no such algorithm was known even for the special case of flow shop scheduling. Note that if each job must be processed on each machine at most once, the factor of μ 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 [14] and Raghavan [12] to approximate certain integer packing problems.

In contrast to this, the only "negative" result previously 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. Subsequent to our work, Williamson, Hall, Hoogeveen, Hurkens, Lenstra, and Shmoys [20] showed that the existence of a ρ-approximation algorithm for any shop scheduling problem with $\rho < 5/4$ would imply that

$\mathcal{P} = \mathcal{NP}$.

Our techniques can also be made to apply to three important generalizations of the job shop problem. The first is *dag shop scheduling*, where each job consists of a set of operations on different machines which must be processed in an order consistent with a given 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.) Note that we still require that no two operations of the same job can be processed simultaneously. 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, a further generalization of this problem is the problem of scheduling on unrelated parallel machines subject to chain precedence constraints: each job now consists of a single operation which may be scheduled on any machine, but its processing time depends on the machine on which it is scheduled; there is set of disjoint chain precedence constraints that further restricts the order in which the jobs may be scheduled.

We also give some extensions of these results, including a $\mathcal{RNC}$ approximation algorithm for each scheduling model mentioned above, and a $(2+\epsilon)$-approximation algorithm for the job shop scheduling problem 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 the job shop scheduling problem takes $O((\mu m n)^2)$ time. Furthermore, the deterministic versions rely on linear programming algorithms. 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 [8] 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 $P_{\max}$, 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 $[0, \Pi_{\max}]$. The resulting schedule, with high probability, has $O(\frac{\log(n\mu)}{\log\log(n\mu)})$ 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 μ; in the next section we will show how to remove this assumption. For simplicity, we shall 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 μ, the strategy of delaying each job an initial integral amount chosen randomly and uniformly from $[0, \Pi_{\max}]$ and then continuously processing its operations in sequence will yield an (invalid) schedule that is of length at most $\Pi_{\max} + P_{\max}$ and, with high probability, has no more than $O(\frac{\log(n\mu)}{\log\log(n\mu)})$ jobs scheduled on any machine during any unit of time.

Proof: Fix a time t and a machine M_i; let $p = \text{Prob}[$at least τ units of processing are scheduled on machine M_i at time $t]$. There are at most $\binom{\Pi_{\max}}{\tau}$ ways to choose τ units of processing from those required on M_i. If we focus on a particular one of these τ units and a specific time t, then the probability that it is scheduled at time t is at most $1/\Pi_{\max}$, since we selected a delay uniformly at random from among $\Pi_{\max}$ possibilities. If all τ units are from different jobs then the probability that they are all scheduled at time t is at most $(\frac{1}{\Pi_{\max}})^\tau$ since the delays are chosen independently. Otherwise, the probability that all τ are scheduled at time t is 0, since two units from the same job can never be assigned to the same time. Therefore

$$\begin{aligned} p &\leq \binom{\Pi_{\max}}{\tau}\left(\frac{1}{\Pi_{\max}}\right)^\tau \\ &\leq \left(\frac{e\Pi_{\max}}{\tau}\right)^\tau \left(\frac{1}{\Pi_{\max}}\right)^\tau \leq \left(\frac{e}{\tau}\right)^\tau. \end{aligned}$$

If $\tau = k\frac{\log(n\mu)}{\log\log(n\mu)}$ then $p < (n\mu)^{-(k-1)}$. We can bound the probability that *any* machine at *any* time is assigned more than $k\frac{\log(n\mu)}{\log\log(n\mu)}$ jobs by $m(P_{\max}+\Pi_{\max})p < n(P_{\max}+\Pi_{\max})(n\mu)^{-(k-1)}$. Since we have assumed that $p_{\max}$ is bounded by a polynomial in n and μ, $P_{\max} + \Pi_{\max}$ is as well; if we choose k sufficiently large, then with high probability, no more than $k\frac{\log(n\mu)}{\log\log(n\mu)}$ jobs are scheduled on 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 cL 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 each machine at any time and produces a schedule of length $O(cL\log p_{\max})$.

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

Proof: To begin, we round up each processing time p_{ij} to the next power of 2 and denote the corresponding rounded time by p'_{ij}; that is, $p'_{ij} = 2^{\lceil \log_2 p_{ij} \rceil}$. Let $p'_{\max} = \max_{ij} p'_{ij}$. From $\mathcal{S}_0$,

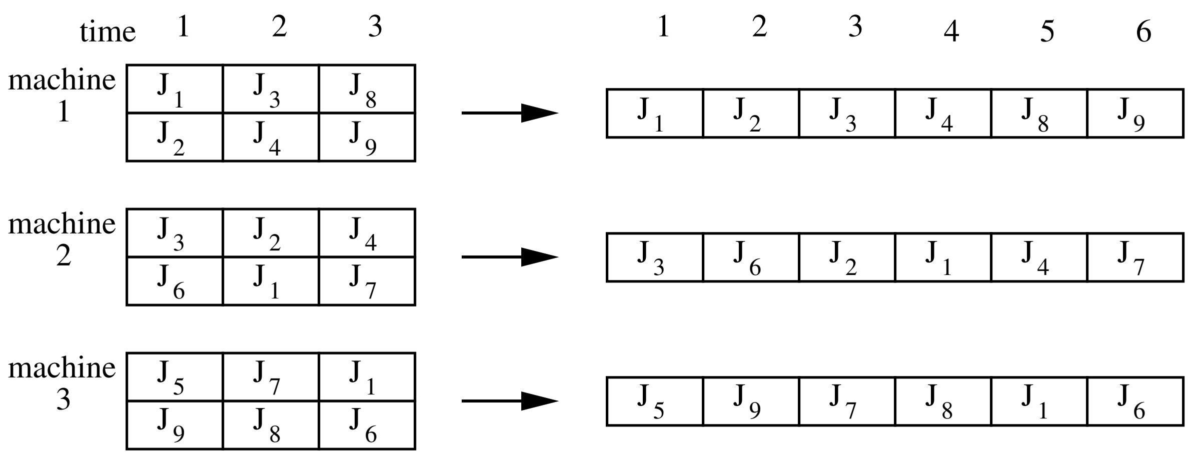

Figure 1: Flattening a schedule in the case with unit length operations.

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 that begins during this 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 $p \log p$; the operations scheduled in a fragment of length T are all of length T, and start at the beginning of the fragment. This algorithm takes advantage of the fact that if an operation of length p is scheduled to begin in a block of size p, then 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, any 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 length 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 the schedule produced from one block. Within a block, if an operation is rescheduled to the final fragment then 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

$$O\left((\Pi_{\max} + P_{\max})\frac{\log(n\mu)}{\log\log(n\mu)}\log p_{\max}\right),$$

under the assumption that $p_{\max}$ was bounded above by a polynomial in n and μ. Since

$$\Pi_{\max} + P_{\max} = O(\max\{\Pi_{\max}, P_{\max}\}),$$

this schedule is within a factor of $O(\frac{\log(n\mu)}{\log\log(n\mu)}\log p_{\max})$ of optimality. In this section, we will first remove the assumption that $p_{\max}$ is bounded above by a polynomial in n and μ 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 $O(\frac{\log^2(n\mu)}{\log\log(n\mu)})$–approximation algorithm. Then we will prove a similar result that reduces this case to one in which n is bounded by a polynomial in m and μ. Combining these two results, we conclude that we can reduce the job shop scheduling problem to its special case where n and $p_{\max}$ are polynomially bounded in m and μ, while changing the performance guarantee by only 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 μ. Let $\omega = |\mathcal{O}|$ be the total number of required operations. Note that $\omega \leq n\mu$. Round down each p_{ij} to the nearest multiple of $p_{\max}/\omega$, and denote this value by p'_{ij}. This ensures that the value p'_{ij} take at most ω distinct values, which 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 processing times p'_{ij}. (Note that assigning $p'_{ij} = 0$ does not mean that this operation does not exist; instead, it should viewed as an operation that takes an arbitrarily small amount of time.) Let L denote the rescaled length of $\mathcal{S}'$.

We claim that $\mathcal{S}'$ for this rounded instance can be interpreted as a schedule for the original operations of length at most $L + p_{\max}$. If we increase the processing time of O_{ij} from p'_{ij} to its original time p_{ij}, we add an amount that is at most $p_{\max}/\omega$. 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 any path; 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 $\mathcal{I}'$ for which $p_{\max} = O(n\mu)$; 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 $p_{\max} = O(n\mu)$ with the property that a schedule for the modified instance of length $kC^*_{\max}$ can be converted in polynomial time to a schedule for the original 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 that is polynomial in m and μ, 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 $\Pi_{\max}/(2m\mu^3)$; otherwise we call the job *small*. For the instance consisting of just the short jobs, let $\Pi'_{\max}$ and $p'_{\max}$ denote the maximum machine load and operation length, respectively. Using the algorithm of [18] described in the introduction, we can, in time polynomial in the input size, produce a schedule of length $\Pi'_{\max} + 2m\mu^3 p'_{\max}$ for this instance. Since $p'_{\max}$ is at most $\Pi_{\max}/(2m\mu^3)$ and $\Pi'_{\max} \leq \Pi_{\max}$, we get a schedule that is of length no more than $2\Pi_{\max}$. 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 $2m^2\mu^3$ long jobs, since otherwise there would be more than $m\Pi_{\max}$ units of processing to be divided amongst m machines, which contradicts the definition of $\Pi_{\max}$. 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 $O(m^2\mu^3)$ jobs with the property that a schedule for the modified instance of length $kC^*_{\max}$ can be converted in polynomial time to a schedule for the original 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 $O(\frac{\log^2(m\mu)}{\log\log(m\mu)} C^*_{\max})$.

Proof: In Section 2 we showed how to produce a schedule of length

$$O\left((\Pi_{\max} + P_{\max})\frac{\log(n\mu)}{\log\log(n\mu)}\log p_{\max}\right)$$

under the assumption that $p_{\max}$ was bounded above by a polynomial in n and μ. 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 μ, while adding only a constant to the factor of approximation. Since now $\log p_{\max} = O(\log(m\mu))$ and $\log n = O(\log(m\mu))$ our algorithm produces a schedule of length $O(\frac{\log^2(m\mu)}{\log\log(m\mu)}C^*_{\max})$. ■

Note that when μ 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 $O(\frac{\log^2(n\mu)}{\log\log(n\mu)}C^*_{\max})$.

4 Applying the techniques to related problems

4.1 A Fixed Number of Machines

In this subsection we will show that the technique of partitioning the set of jobs by size can be applied to give a much better performance guarantee in the special case in which the number of machines and the maximum number of operations per machines are assumed to be constants. It is interesting to note that Sevast'yanov's algorithm for the job shop problem can be viewed as a $(1 + m\mu^3)$-approximation algorithm, so that when m and μ are constant, this is a $O(1)$-approximation algorithm. We will give a ρ-approximation algorithm for ρ arbitrarily close to 2.

To apply the idea of partitioning the jobs in this setting, call a job J_j *big* if there is an operation O_{ij} with $p_{ij} > \epsilon\Pi_{\max}/(m\mu^3)$, where ϵ is an arbitrary positive constant. Note that there are at most $m^2\mu^3/\epsilon$ big jobs, and since m, μ 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 4.1 For any fixed value of $\epsilon > 0$, there is a polynomial-time algorithm for the special case of the job shop scheduling problem where both m and μ are fixed that produces a schedule of length $\leq (2+\epsilon)C^*_{\max}$.

4.2 Dag shop scheduling with identical copies of machines

The fact that the quality of our approximations is based solely on the lower bounds $\Pi_{\max}$ and $P_{\max}$ makes it quite easy to extend our techniques to the more general problem of *dag shop scheduling*, in which each job is given by a specified partial order of operations. We define $\Pi_{\max}$ and $P_{\max}$ exactly the same way, and $\max\{\Pi_{\max}, P_{\max}\}$ remains a lower bound for the length of any schedule. We can reduce this dag shop scheduling problem to a job shop problem by selecting for each job an arbitrary total order that is consistent with its partial order. The maximum job length and maximum machine load for job shop instance constructed are equal to the analogous values for the original dag shop instance. Therefore, a schedule of length $\rho\cdot(\Pi_{\max}+P_{\max})$ for this job shop instance is a schedule for the original dag shop 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 has to run on only one of these identical copies. $P_{\max}$ remains a lower bound on the length of any schedule for this problem. $\Pi_{\max}$, 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 $\Pi(S_i)$ be the sum of the lengths of the operations which run on S_i. Our strategy is to reduce 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. For each set of machines S_i, $i = 1,\ldots,m'$, the average load on that set of machines is clearly a lower bound on the maximum machine load of machines within that set; thus

$$\Pi_{\text{avg}} = \max_{S_i} \frac{\Pi(S_i)}{|S_i|}$$

is 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 $2\Pi_{\text{avg}}+p_{\max}$. A schedule for the resulting job shop problem of length $\rho\cdot(\Pi_{\max}+P_{\max})$ yields a solution for the more general problem of length $O(\rho\cdot(\Pi_{\text{avg}}+P_{\max}))$. Sevast'yanov [18] used a somewhat more complicated reduction to handle a slightly more general setting.

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

Theorem 4.2 There exists a polynomial-time randomized algorithm for dag shop scheduling with identical copies of machines that, with high probability, yields a schedule that is of length at most

$O(\frac{\log^2(m\mu)}{\log\log(m\mu)}C^*_{\max})$.

Corollary 4.3 There exists an $\mathcal{RNC}$ algorithm for dag shop scheduling with identical copies of machines that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2(n\mu)}{\log\log(n\mu)}C^*_{\max})$.

4.3 Unrelated parallel machines with chain precedence constraints

A further generalization of the job shop problem is the problem of scheduling jobs on unrelated parallel machines subject to chain precedence constraints, which is denoted $R|chain|C_{\max}$ in the notation of [7]. In this problem, we are given a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, each of which is to be processed by exactly one of a set of machines $\mathcal{M} = \{M_1 \ldots, M_m\}$; if J_j is processed by M_i, it takes p_{ij} time units. We are also given a partial order $\prec$ that specifies job precedence constraints: if $J_j \prec J_k$ then J_k may not start processing until J_j has completed processing. The precedence constraints are restricted to be a disjoint union of chains; that is, we are given a set $\mathcal{C} = \{C_1, \ldots, C_k\}$, where each chain $C_l \in \mathcal{C}$ consists of some μ_l jobs, given with a linear ordering on these jobs; each job is in exactly one chain (where unconstrained jobs are viewed as chains of length one). The objective is to construct a schedule consistent with the precedence constraints so as to minimize the maximum job completion $C_{\max}$.

We will view the problem of finding a feasible solution for this problem as having two phases: finding an assignment of jobs to machines, and then constructing a schedule consistent with that assignment. It is easy to see that the second phase is a job shop scheduling problem: each operation of the job shop problem corresponds to a job of the parallel machines problem, and each job of the job shop problem corresponds to a chain of the parallel machines problem. We will construct an assignment such that the resulting job shop instance has $\Pi_{\max} \leq 4L$ and $P_{\max} \leq 2L$, where L is a lower bound for the original instance of $R|chain|C_{\max}$. By applying our approximation algorithms for the job shop problem to this instance, we obtain approximation algorithms for $R|chain|C_{\max}$ with an identical performance guarantee (up to a constant factor).

To find a suitable assignment, we give an algorithm that takes a given threshold value d, and either proves that any feasible schedule for the instance of $R|chain|C_{\max}$ has $C_{\max} > d$, or else constructs an assignment such that the resulting job shop instance has $\Pi_{\max} \leq 4d$ and $P_{\max} \leq 2d$. By performing a bisection search with initial lower bound 0 and upper bound $\sum_j \max_i p_{ij}$, we obtain a polynomial-time algorithm that finds a value L such that no schedule of length $L-1$ exists (and hence L is a valid lower bound), and we have an assignment such that the resulting job shop instance has $\Pi_{\max} \leq 4L$ and $P_{\max} \leq 2L$.

The algorithm that tests a given threshold is based on results of Lenstra, Shmoys, & Tardos [9] and Lin & Vitter [10]. It works by first constructing an integer program which must be feasible if there is a schedule of length d, and then checks if its linear relaxation is feasible: if the linear program is infeasible, then no schedule of length d exists, and if the linear program is feasible, then the fractional solution can be rounded to yield an integer assignment with the claimed properties.

Observe that if there is a schedule with $C_{\max} \leq d$, then the following linear program has a feasible solution:

$$\begin{aligned} \sum_{j=1}^{n} p_{ij}x_{ij} &\leq d, & i = 1, \ldots, m, \\ \sum_{i=1}^{m}\sum_{j \in C_l} p_{ij}x_{ij} &\leq d, & l = 1, \ldots, k, \end{aligned}$$

$$\begin{aligned} \sum_{i=1}^{m} x_{ij} &= 1, \qquad i = 1,\ldots,m, \\ x_{ij} &\geq 0, \qquad i = 1,\ldots,m,\ j = 1,\ldots,n. \end{aligned}$$

Let x_{ij} denote such a feasible solution; we can view this solution as a fractional assignment where an x_{ij} fraction of job J_j is assigned to M_i. Let p_j denote the total time spent processing J_j in this fractional assignment; that is,

$$p_j = \sum_{i=1}^{m} p_{ij} x_{ij}, \quad j = 1,\ldots,n.$$

In this fractional assignment, it is possible for J_j to have a small fraction assigned to a machine M_i for which its p_{ij} value is substantially more than p_j. To make sure that this does not occur, we first apply the filtering technique of Lin & Vitter [10], and round x_{ij} to $\bar{x}_{ij}$ by setting

$$\bar{x}_{ij} = \begin{cases} x_{ij}, & \text{if } p_{ij} \leq 2p_j, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Of course, for each job J_j, only a fraction $t_j = \sum_{i=1}^{m} \bar{x}_{ij}$ of it is now assigned. We renormalize this fractional assignment by forming $\tilde{x}_{ij} = \bar{x}_{ij}/t_j$. Observe that $t_j \geq 1/2$, $j = 1,\ldots,n$, since less than half of each job J_j can be assigned to machines M_i for which $p_{ij} > 2p_j$. Therefore, $\tilde{x}$ satisfies

$$\begin{aligned} \sum_{j=1}^{n} p_{ij}\tilde{x}_{ij} &\leq 2d, \qquad i = 1,\ldots,m, \\ \sum_{i=1}^{m} \tilde{x}_{ij} &= 1, \qquad i = 1,\ldots,m, \\ \tilde{x}_{ij} &= 0, \qquad \text{if } p_{ij} > 2p_j,\ i = 1,\ldots,m,\ j = 1,\ldots,n, \\ \tilde{x}_{ij} &\geq 0, \qquad i = 1,\ldots,m,\ j = 1,\ldots,n. \end{aligned}$$

A result of Lenstra, Shmoys, & Tardos[9] states that, in polynomial-time, any extreme point of such a linear program can be rounded to an integer solution x^* such that

$$\begin{aligned} \sum_{j=1}^{n} p_{ij}x^*_{ij} &\leq 2d + 2p_j, \qquad i = 1,\ldots,m, \\ \sum_{i=1}^{m} x^*_{ij} &= 1, \qquad i = 1,\ldots,m, \\ x^*_{ij} &= 0, \qquad \text{if } p_{ij} > 2p_j,\ i = 1,\ldots,m,\ j = 1,\ldots,n, \\ x^*_{ij} &\geq 0, \qquad i = 1,\ldots,m,\ j = 1,\ldots,n. \end{aligned}$$

Furthermore, since $\sum_{j \in C_l} p_j \leq d$, $l = 1,\ldots,k$, we have that

$$\sum_{j \in C_l} \sum_{i=1}^{m} p_{ij} x^*_{ij} \leq \sum_{j \in C_l} 2p_j \leq 2d.$$

Consequently, if for each job J_j we assign the corresponding operation to the machine M_i for which $x^*_{ij} = 1$, then we obtain a job shop instance for which $P_{\max} \leq 2d$, and $\Pi_{\max} \leq 2d + 2p_j \leq 4d$, as claimed.

Theorem 4.4 There exists a polynomial-time randomized algorithm for $R|chain|C^*_{\max}$ that, with high probability, yields a schedule that is of length at most $O(\frac{\log^2 m}{\log\log m} C^*_{\max})$.

5 A Deterministic Approximation Algorithm

In this section, we "derandomize" the results of the previous sections: we give a deterministic polynomial-time algorithm that finds a schedule of length $O(\log^2(m\mu)C^*_{\max})$. Of all the components of the algorithm of Theorem 3.3, the only step which is not already deterministic is the step that uniformly chooses a random initial delay for each job with the resulting property that, with high probability, no machine is assigned too many jobs at any time. In particular, the reduction to the special case in which n and $p_{\max}$ are bounded by a polynomial in m and μ 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 $O(\log(m\mu))$ jobs running at any one time. We then apply Lemma 2.2 to produce a schedule of length $O(\log^2(m\mu)C^*_{\max})$. Note that the $O(\log(m\mu))$ jobs per machine is not as good as the probabilistic bound of $O(\frac{\log(m\mu)}{\log\log(m\mu)})$. Recently, Schmidt, Siegel, and Srinivasan [15] have given a different derandomizing strategy for this problem that yields delays that match the performance of the randomized algorithm.

Our approach to the problem of selecting good delays is to frame it as a *vector selection* problem and then apply techniques developed by Raghavan and Thompson [13, 14] and Raghavan [12] 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 [13, 14]. Furthermore, for many of the problems for which there are approximations with high probability, the algorithm can be derandomized. Raghavan [12] has shown how to do this by essentially setting the random bits one at a time.

We now state the problem formally:

Problem 5.1 Deterministically assign a delay to each job in the range $[0, \Pi_{\max}/\log(m\mu)]$ so as to produce a schedule with $O(\log(m\mu))$ jobs on any machine at any time.

Lemma 5.2 Problem 5.1 can be solved in deterministic polynomial time.

Proof: Since we introduce delays in the range $[0, \Pi_{\max}/\log(m\mu)]$, the resulting schedule has length $\ell = P_{\max} + \Pi_{\max}/\log(m\mu)$. We can represent the processing of a job J_j with a given initial delay d by $\{0,1\}$–vector of length $\ell \cdot m$, where each component corresponds to a particular 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 $d_{\max} = \Pi_{\max}/\log(m\mu)$, and let $V_{j,k}(i)$ be the ith 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 V^*_j from each λ_j such that

$$\left\| \sum_{j=1}^{n} V^*_j \right\|_\infty = O(\log(m\mu));$$

that is, at any time on any machine, the number of jobs using that machine is $O(\log(m\mu))$.

As in [12], 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 time. We already know, by Lemma 2.1, that $W_{\mathrm{OPT}} = O(\log(m\mu))$, and so an optimal solution to this integer program would solve Problem 5.1. However, the problem is $\mathcal{NP}$-hard. Instead, we rely on the following theorem which is immediate from the results in [12] and [14].

Theorem 5.3 [12, 14] A feasible solution to (IP) with $W = O(W_{\mathrm{OPT}} + \log(m\mu))$ can be found in polynomial time.

■

We then apply Lemma 2.2 to obtain the following result:

Theorem 5.4 There exists a deterministic polynomial-time algorithm that finds a schedule of length $O(\log^2(m\mu)\, C^*_{\max})$.

6 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 shop scheduling, and a several further generalizations.

One particularly simple special case of dag shop 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 $\Omega(\Pi_{\max} + P_{\max})$. 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 $O((\Pi_{\max} + P_{\max})\frac{\log^2 m}{\log\log m})$. Hence for any instance of the open shop problem, we can impose an arbitrary partial order on the operations of each job and increase the length of the optimal schedule by a factor of no more than $O(\frac{\log^2 m}{\log\log m})$.

On the other hand, there does exist a schedule of length $O(P_{\max} + \Pi_{\max})$ 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 [2] has observed that the greedy algorithm delivers a schedule of length at most $\Pi_{\max} + (m-1)p_{\max}$. We can adapt her proof to show that, in fact, the greedy algorithm delivers a schedule that is of length less than $\Pi_{\max} + P_{\max} \leq 2C^*_{\max}$. 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 $\sum_{M_i \neq M_k} p_{ij} < P_{\max}$. Thus, machine M_k is processing for at most $\Pi_{\max}$ units of time and is idle for less than $P_{\max}$ units of time; hence $C_{\max} < \Pi_{\max} + P_{\max}$. Fiala [5] has also shown that if $\Pi_{\max} \geq (16m \log m + 21m)p_{\max}$, then $C^*_{\max}$ is just $\Pi_{\max}$, 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 $O(\Pi_{\max} + P_{\max})$, and so the major open question left unresolved by this paper is:

- Does there exist an $O(\Pi_{\max} + P_{\max})$ 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? Williamson et al. [20] have shown that the existence of a ρ-approximation algorithm with $\rho < 5/4$ would imply that $\mathcal{P} = \mathcal{NP}$. It remains an interesting open problem to close this gap.

- Our algorithms, while polynomial-time algorithms, are inefficient. Are there significantly more efficient algorithms which have the same performance guarantees? Stein [19] has given an algorithm that directly finds a good approximate solution to the integer program (IP) by using the framework of Plotkin, Shmoys, and Tardos [11]. This yields an implementation of our algorithm that runs in $O(n^2m^2\mu^2 + n^3\mu^2 \log(m\mu)(\mu + \log(m\mu)))$ time. Although this represents a dramatic improvement over the previously known bound, it remains an interesting question to give substantially more efficient algorithms.

Acknowledgments

We are grateful to David Williamson for working with us in the early stages 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 shop scheduling.

References

[1] D. Applegate and B. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 3:149–156, 1991.

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

[3] 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.

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

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

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[7] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol 4., Logistics of Production and Inventory*, pages 445–522. North-Holland, 1993.

[8] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 256–269, 1988.

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

[10] J.H. Lin and J.S. Vitter. ϵ-approximation with minimum packing constraint violation. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 771–782, 1992.

[11] S. Plotkin, D. B. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. To appear in Math of Operations Research, 1995.

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

[13] 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.

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

[15] J.P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 331–340, 1993.

[16] 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.

[17] 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.

[18] S.V. Sevast'yanov. Bounding algorithm for the routing problem with arbitrary paths and alternative servers. *Kibernetika*, 22(6):74–79, 1986. Translation in Cybernetics 22, pages 773-780.

[19] C. Stein. *Approximation algorithms for multicommodity flow and shop scheduling problems.* PhD thesis, MIT, Cambridge, MA, August 1992. Also appears as MIT/LCS/TR-550.

[20] D. P. Williamson, L. Hall, J. A. Hoogeven, C. A. J. Hurkens, J. K. Lenstra, and D. B. Shmoys. Short shop schedules. Unpublished Manuscript, 1993.

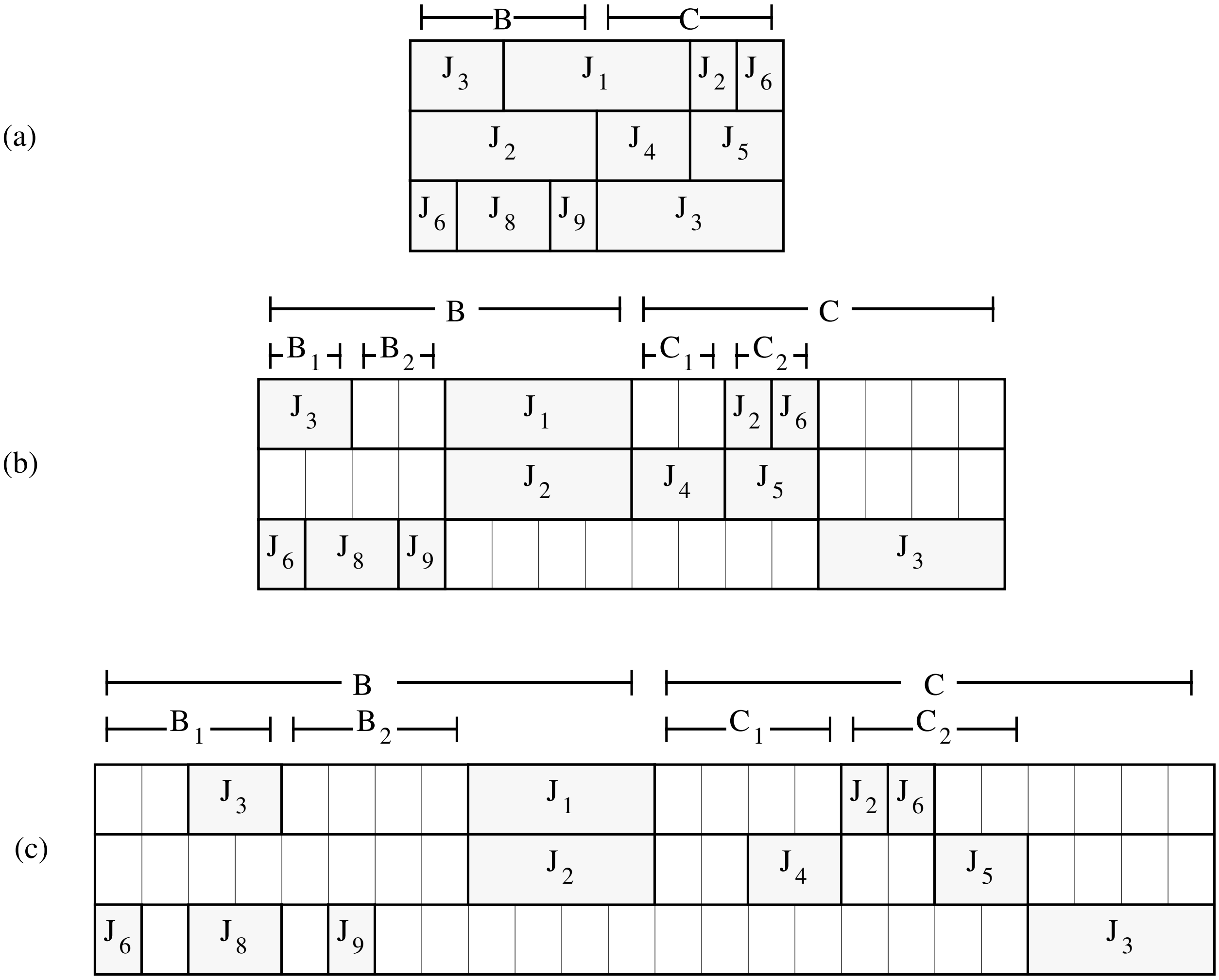

Figure 2: (a) The initial greedy schedule of length 8. $p'_{\max} = 4$. (b) The first level of spreading. All jobs of length 4 have been put in the final fragments. We must now recurse on B_1 and B_2 with $p'_{\max} = 2$. (c) The final schedule of length $8 \log_2 8 = 24$.