# Minimizing Average Completion Time in the Presence of Release Dates


Cynthia Phillips[*] Clifford Stein[†] Joel Wein [‡]


September 13, 1999


**Abstract**

A natural and basic problem in scheduling theory is to provide good average quality of service to a stream of jobs that arrive over time. In this paper we consider the problem of scheduling $n$ jobs that are released over time in order to minimize the average completion time of the set of jobs. In contrast to the problem of minimizing average completion time when all jobs are available at time 0, all the problems that we consider are $\mathcal{NP}$-hard, and essentially nothing was known about constructing good approximations in polynomial time.

We give the first constant-factor approximation algorithms for several variants of the single and parallel machine model. Many of the algorithms are based on interesting algorithmic and structural relationships between preemptive and nonpreemptive schedules and linear programming relaxations of both. Many of the algorithms generalize to the minimization of average *weighted* completion time as well.



**Keywords:** Scheduling, Preemptive Scheduling, Average Weighted Completion Time, Approximation Algorithms, Relaxations, Linear Programming


## 1 Introduction

Two important characteristics of many real-world scheduling problems are that (1) the tasks to be scheduled arrive over time and (2) the goal is to optimize some function of average performance or satisfaction. In this paper we study several scheduling models that include both of these characteristics; in particular we study the minimization of the average (weighted) completion time of a set of jobs with release dates. In many of the models that we study, polynomial-time algorithms to


[*] `caphill@cs.sandia.gov`. Sandia National Labs, Albuquerque, NM.


This work was performed under U.S. Department of Energy contract number DE-AC04-76AL85000.



[†] `cliff@cs.dartmouth.edu`. Department of Computer Science, Sudikoff Laboratory, Dartmouth College, Hanover, NH.


Research partly supported by NSF Award CCR-9308701, a Walter Burke Research Initiation Award and a Dartmouth College Research Initiation Award.



[‡] `wein@mem.poly.edu`. Department of Computer Science, Polytechnic University, Brooklyn, NY, 11201.


Research partially supported by NSF Research Initiation Award CCR-9211494 and a grant from the New York State Science and Technology Foundation, through its Center for Advanced Technology in Telecommunications.

minimize average completion time when all the jobs are available at time 0 were known. The introduction of release dates makes these problems $\mathcal{NP}$-hard, and little was known about approximation algorithms.

Our major contribution is to give the first constant-factor approximation algorithms for the minimization of average completion time in these models. Our performance bounds come from the combination of two different types of results. First, we prove structural theorems about the relative quality of preemptive versus nonpreemptive schedules, and give an algorithm to convert from the former to the latter with only a small degradation in schedule quality. Second, we give the first constant-approximation algorithms for preemptively scheduling parallel machines with release dates. When the machines are identical, we give a combinatorial algorithm which yields a schedule of average completion time at most a factor of two worse than optimal. For unrelated machines, we give a new integer programming formulation. We then solve its linear programming relaxation, and give methods to round the fractional solution to valid preemptive schedules.

**Models:** We are given $n$ jobs $J_1, \ldots, J_n$ where job $J_j$ has associated processing time $p_j$ and a release date $r_j$, before which it cannot be processed on any machine. We will also be given $m$ machines $M_1, \ldots, M_m$. We will focus both on the one machine ($m = 1$) environment and two fundamental variants of parallel machine scheduling. In the *identical parallel machine* environment, job $J_j$ runs in time $p_j$ on every machine [16]. In the *unrelated parallel machine scheduling* environment, we are given speeds $s_{ij}$ which characterize how fast job $J_j$ runs on machine $M_i$, and $p_{ij}$, the processing time of job $J_j$ on machine $M_i$, is defined to be $p_{ij} = p_j/s_{ij}$, and thus depends on both the machine and the job [16].

We will give algorithms for both preemptive and nonpreemptive scheduling models. In *nonpreemptive* scheduling, once a job begins running on a machine, it must run uninterruptedly to completion, while in *preemptive* scheduling a job that is running can be preempted and continued later on any machine. At any point in time a job may be running on at most one machine, and a machine may be running at most one job.

We will use the notation $C_j^S$ to denote the completion time of job $J_j$ in a schedule $S$, and will often drop the superscript when it is clear to which schedule we refer. Our basic optimization criterion is the *average completion time* of the set of jobs, $\frac{1}{n}\sum_j C_j$. At times we will associate with $J_j$ a nonnegative weight $w_j$ and seek to minimize the average *weighted* completion time, $\frac{1}{n}\sum_j w_j C_j$. These optimality criteria are fundamental ones in scheduling theory and accordingly have received much attention, e.g. [2, 5, 11, 12, 14, 17, 18, 19, 22]. For simplicity, we will typically drop the factor of $1/n$ and refer, equivalently, to the optimization of *total* (weighted) completion time.

Since there are a number of scheduling models considered in this paper, it will be convenient to refer to them in the notation of Graham, Lawler, Lenstra, & Rinnooy Kan (1979) [7]. Each problem is denoted by $\alpha|\beta|\gamma$, where in this paper (i) $\alpha$ is either 1, $P$, or $R$, denoting that the scheduling environment contains either one machine, $m$ identical parallel machines, or $m$ unrelated parallel machines respectively; (ii) $\beta$ contains $r_j$, *pmtn*, both, or neither, indicating whether there are (non-trivial) release date constraints, whether preemption is allowed, both, or neither; and (iii) $\gamma$ is either $\sum C_j$, indicating that we are minimizing total completion time, or $\sum w_j C_j$, indicating that we are minimizing total weighted completion time. For example, $1|r_j, pmtn|\sum w_j C_j$ denotes the problem of minimizing the total weighted completion time on one machine subject to release dates, with preemption allowed. We shall assume, without loss of generality, that the data for each instance are integral.

We distinguish between *off-line* and *on-line* algorithms. An off-line algorithm is given, in advance, knowledge of all the input data ($r_j$ and $p_j$) associated with the jobs, and uses it to construct the schedule. In contrast, an on-line scheduling algorithm constructs the schedule over time, and at time $t$ knows only about jobs with release dates $r_j \leq t$. The algorithm must decide what job (if any) to be scheduling at time $t$ *with no knowledge of what jobs will arrive in the future.* In this paper, unless we explicitly state that an algorithm is on-line, it can be assumed to be off-line.

We define a $\rho$-approximation algorithm to be one which, in polynomial time, produces a schedule whose value is guaranteed to be at most $\rho$ times the minimum possible value.

**Results:** We first describe an algorithm which converts preemptive schedules to nonpreemptive schedules while only increasing the total (weighted) completion time by a small constant factor. Given a preemptive schedule for one machine, our algorithm produces a nonpreemptive schedule while increasing the total (weighted) completion time by at most a factor of 2, and given a preemptive schedule for $m$ identical parallel machines, our algorithm produces a nonpreemptive schedule while increasing the total (weighted) completion time by at most a factor of 3. This technique applies even when the preemptive schedule is fractional (fractional parts of different jobs are assigned simultaneously to one machine). Note that these results give some insight into the power of preemption: the best preemptive schedule in these settings has an average weighted completion time that is no more than a small constant factor smaller than that of the best nonpreemptive schedule.

We apply this conversion algorithm to the problem $1|r_j|\sum C_j$. This problem is $\mathcal{NP}$-hard [17]; we give a simple 2-approximation algorithm for it, which works by transforming the optimal preemptive schedule for the problem, which can be found in polynomial time [1]. The best previous approximation algorithm for this problem was an $O(\log^2 n)$-approximation algorithm [20]. Our algorithm can be modified slightly to be on-line; we show as well that it is close to an optimal on-line scheduler, by showing that any on-line scheduling algorithm for the problem must be at least a $\frac{3}{2}$-approximation algorithm.

We then turn to parallel identical machines. McNaughton [18] showed that for $P|pmtn|\sum C_j$ no preemptions are needed in order to minimize the average completion time; therefore the polynomial-time algorithm for the nonpreemptive version of this problem [2, 11] solves it directly. When release dates are introduced, however, even the two machine problem becomes $\mathcal{NP}$-hard [5]. When one attempts, in addition, to minimize the average weighted completion time even the one machine version of the problem becomes $\mathcal{NP}$-hard [14]. To the best of our knowledge, nothing was known about approximation algorithms for any version of these problems.

We give a combinatorial 2-approximation algorithm for $P|r_j, pmtn|\sum C_j$. This algorithm is the natural $m$-machine generalization of the shortest processing time rule. Combining this algorithm with our conversion algorithm yields a 6-approximation algorithm for the nonpreemptive problem $P|r_j|\sum C_j$.

For the more general objective of minimizing average *weighted* completion time, we may again use our conversion techniques, which now construct nonpreemptive schedules of total weighted completion time at most twice or three times that of the preemptive schedule. The simple techniques, however, for finding optimal or near-optimal preemptive schedules that minimize $\sum C_j$ do not generalize easily to the $\sum w_j C_j$ objective. Further, none of the above-mentioned algorithms apply when the machines are unrelated. To solve these problems, we introduce an integer program that is closely related to optimal preemptive schedules to minimize average weighted completion time on unrelated machines. We solve the corresponding linear programming relaxation and then show how to round this to a feasible, provably good, preemptive schedule. This formulation yields

| Environment | Nonpreempt. $\sum C_j$ | Nonpreempt. $\sum w_j C_j$ | Preempt. $\sum C_j$ | Preempt. $\sum w_j C_j$ |
|---|---|---|---|---|
| One machine | 2 | $16+\epsilon$ | 1[1] | $8+\epsilon$ |
| Identical | 6 | $24+\epsilon$ | 2 | $8+\epsilon$ |
| unrelated | $O(\log^2 n)$[20] | $O(\log^2 n)$[20] | $8+\epsilon$ | $8+\epsilon$ |

Figure 1: Summary of results. Unreferenced results are new results found in this paper, and $\epsilon$ is an arbitrarily small constant.

different approximation guarantees for different problems; a summary of our results is presented in Figure 1.

We note also that one consequence of this last result is the first constant-approximation algorithm for $R|pmtn|\sum C_j$. It is not known if this problem is $\mathcal{NP}$-hard; Lawler, Lenstra, Rinnooy-Kan and Shmoys [16] state, "Very little is known about this problem ... it remains one of the more vexing questions in the area of preemptive scheduling."

**Previous Work:** The only prior work we know of that gives approximation algorithms for these problems is [20] which gives $O(\log^2 n)$-approximation algorithms for the nonpreemptive versions of the problems, as a special case of a very general theorem. There is some similarity of spirit between the algorithms there and those we give in Section 5 in that both solve a type of generalized matching problem. The particular generalization of matching, the rounding techniques, and the quality of approximation achieved, however, are quite different.

Subsequent to our work, Hall, Schulz, Shmoys and Wein [8], and Chakrabarti, Phillips, Schulz, Shmoys, Stein and Wein [3] have given new approximation algorithms with performance guarantees that improve upon several of those proved in this paper. Many of their results can be viewed as building upon the basic ideas in this paper.

Also subsequent to our work, Stougie [23] and Hoogeveen and Vestjens [10] gave different on-line 2-approximation algorithms for $1|r_j|\sum C_j$; in addition they improved the lower bound on on-line scheduling from $\frac{3}{2}$ to 2.

Whereas our algorithms in Sections 2 - 4 run in $O(n\log n)$ time, the algorithms in Section 5 run in polynomial time, but the polynomial is quite large. We also note that we are *not* studying the average flow time of a set of jobs, where the flow time is $C_j - r_j$. Although average flow time is equivalent to average completion time at optimality, Kellerer, Tautenhahn and Woeginger have shown that the approximability of these two criteria can be quite different [13].

# 2 Converting Preemptive Schedules to Nonpreemptive Schedules

In this section we give an algorithm that converts preemptive schedules into nonpreemptive schedules while increasing the total (weighted) completion time by at most a constant factor. We will analyze this algorithm in both the one machine and identical parallel machine models. We will also define and discuss *fractional preemptive schedules* and extend our conversion algorithm to these schedules. The application of these techniques to give approximation algorithms is presented in the next section.

## 2.1 The Algorithm

Assume that we are given an instance of either $1|r_j, pmtn|\sum w_j C_j$ or $P|r_j, pmtn|\sum w_j C_j$, and a preemptive schedule $P$ in which each job $J_j$ has completion time $C_j^P$. We introduce the following simple algorithm Convert, which orders the jobs by their preemptive completion times, and then nonpreemptively schedules them in this order.

| Algorithm Convert |
|---|
| **Input:** Preemptive schedule $P$ |
| **Output:** Nonpreemptive schedule $N$ |
| 1. Form a list $L$ of the jobs, ordered by their preemptive completion times $C_j^P$. |
| 2. List schedule nonpreemptively using $L$, with the constraint that no job $J_j$ starts before its release date $r_j$. |

To slightly simplify our analysis, we specify that our list scheduling algorithm will not start the $j$th job in $L$ until the $(j-1)$st is completed (in the one machine environment) or started (in the parallel machine environment). It is possible that the constructed schedule will contain "holes", into which jobs might be shifted forward for an improved schedule, but our analysis does not require this algorithmic enhancement.

## 2.2 One Machine

We now analyze algorithm Convert for the case of one machine.

**Theorem 2.1** *Given a preemptive schedule $P$ for $1|r_j, pmtn|\sum w_j C_j$, algorithm* Convert *produces, in $O(n)$ time, a nonpreemptive schedule $N$ in which, for each job $J_j$, $C_j^N \leq 2C_j^P$.*

**Proof:** Algorithm Convert can be visualized as follows. Consider the last scheduled piece of any job $J_j$, which has length $k_j$, and is scheduled from time $C_j^P - k_j$ to $C_j^P$ in the preemptive schedule. Insert $p_j - k_j$ extra units of time in the schedule at time $C_j^P$, and schedule $J_j$ nonpreemptively in the resulting available block of length $p_j$. This requires that we remove from the schedule all pieces of $J_j$ that were processed before $C_j^P$. We then push all jobs forward in time as much as possible without changing the scheduled order of the jobs or violating a release date constraint. The result is exactly the schedule computed by Algorithm Convert. See Figure 2 for an example.

In this process, job $J_j$ can only be moved back by processing times associated with jobs that finish earlier in $P$ and hence:

$$C_j^N \leq C_j^P + \sum_{k: C_k^P \leq C_j^P} p_k.$$

However, since all the processing captured in the sum in the equation above occurred before time $C_j^P$ in $P$, this sum is at most $C_j^P$ and hence $C_j^N \leq 2C_j^P$. ■

For a particular schedule $S$, let $C^S = \sum_j w_j C_j^S$. Applying Theorem 2.1 to each job in turn, we get the following corollary:

**Corollary 2.2** *Given a schedule $P$ for $1|r_j, pmtn|\sum w_j C_j$, algorithm* Convert *produces a schedule $N$ for the corresponding instance of $1|r_j|\sum w_j C_j$ with $C^N \leq 2C^P$.*

Our analysis of the performance guarantee of algorithm Convert on one machine is tight, as is indicated by the following instance. At time 0 a job of processing time $B$ is released, at time $B-2$ a job of processing time 1 is released, and at time $B+1$, $x$ jobs of processing time 1 are released. The

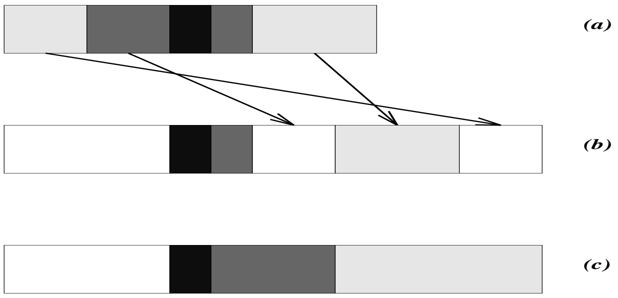


Figure 2: Illustration of the one machine algorithm. Schedule (a) is the optimal preemptive schedule. Schedule (b) has all but the last piece of each job removed from schedule (a), and empty space inserted in the schedule to make room for the pieces to be collected at the completion point of the job in (a). Schedule (c) is the resulting nonpreemptive schedule.

optimal preemptive schedule for this instance has total completion time $B(x+2)-1+\frac{(x+1)(x+2)}{2}$. The optimal nonpreemptive schedule, which has total completion time 1 greater, can be obtained by completing the job of processing time $B$ first and then completing all of the jobs with unit processing time. Algorithm Convert, however, yields total completion time $3B+2Bx-2+\frac{x(x-1)}{2}$. Taking $x=\Theta(\sqrt{B})$ and letting $B$ go to infinity causes the ratio of these two quantities to go to 2.

Corollary 2.2 can be interpreted as a statement about the power of preemption: for a one machine problem, allowing preemption will improve the optimal average weighted completion time by at most a factor of 2. We now demonstrate an instance for which it improves the average weighted completion time by a factor of approximately $\frac{4}{3}$.

**Theorem 2.3** *Given $n$ jobs with release dates on one machine, let $C^P$ be the minimum possible average completion time of those jobs when preemption is allowed, and let $C^N$ be the minimum average completion time with no preemption allowed. Then there exist instances for which $C^N/C^P = \frac{4}{3}-\epsilon$, for any $\epsilon > 0$.*

**Proof:** Consider an instance with two jobs, $J_0$ with $p_0 = x$ and $r_0 = 0$, and $J_1$ with $p_1 = 1$ and $r_1 = \frac{x}{2}$. In the optimal preemptive schedule the total completion time is $(\frac{x}{2}+1)+(x+1) = \frac{3}{2}x+2$. In the optimal nonpreemptive schedule the total completion time is $2x+1$, yielding a ratio arbitrarily close to $\frac{4}{3}$. ■

Recently, T.C. Lai has given a slightly stronger lower bound of $\frac{18}{13}$ [15].

### 2.3 Parallel Identical Machines

In this section we analyze the application of algorithm Convert to parallel identical machines. Note that the analysis of the one machine case does not immediately apply, since a particular job $J_j$ may run on one machine in the preemptive schedule $P$, but may run on a different machine in the nonpreemptive schedule $N$. We can, however, obtain the following bound.

**Theorem 2.4** *Given a preemptive schedule $P$ for $P|r_j, pmtn|\sum w_j C_j$, algorithm* Convert *produces, in $O(n)$ time, a nonpreemptive schedule $N$ in which, for each job $J_j$, $C_j^N \le 3C_j^P$.*

**Proof:** Index the jobs by the order in which they complete in preemptive schedule $P$. Consider job $J_j$ which in $P$ was released at time $r_j$ and completed at $C_j^P$. $J_j$ is not added to schedule $N$ until $J_1, J_2, \ldots, J_{j-1}$ are added, so it will not start until all of $J_1, \ldots, J_{j-1}$ have started. Let $r'_j$ be the latest release date of any job in $J_1, \ldots, J_j$. We know that $r'_j \leq C_j^P$ since $r'_j$ is the release date of a job that finished no later than $C_j^P$. By time $r'_j$, jobs $J_1, \ldots, J_{j-1}$ have all been released. Thus, even if in $N$ no processing happens before time $r'_j$, by standard averaging arguments at least one machine will have no more processing to do on any one of $J_1, \ldots, J_{j-1}$ by time $(\sum_{i=1}^{j-1} p_i)/m$. Thus $J_j$ starts in the nonpreemptive schedule by time

$$r'_j + (\sum_{i=1}^{j-1} p_i)/m \leq r'_j + C_j^P.$$

This last inequality is true because jobs $J_1, \ldots, J_j$ all completed by time $C_j^P$ in $P$. Therefore, $J_j$ completes by time $r'_j + C_j^P + p_j \leq 3C_j^P$. ■

**Corollary 2.5** *Given a schedule for $P|r_j, pmtn|\sum w_jC_j$, algorithm* Convert *produces a schedule for the corresponding instance of $P|r_j|\sum w_jC_j$ with $C^N \leq 3C^P$.*

Corollary 2.5 can also be interpreted as a statement about the power of preemption; namely, in the parallel machine environment, preemption improves the average weighted completion time by at most a factor of 3.

Theorem 2.4 is essentially tight, as the following example shows. This example has $m+1$ big jobs with $p_j = m$ and $r_j = 0$ and $m$ small jobs with $p_j = 1$ and $r_j = m-2$. The optimal preemptive schedule runs the big jobs in a "round-robin" manner from time 0 through $m-2$. Then it interrupts them all and runs the small jobs to completion, and finally returns to the big jobs, finishing the last job $J_{j'}$ at time $m+2$. The list $L$ has all the small jobs, followed by all the big jobs, with $J_{j'}$ being last on the list. Thus in the nonpreemptive schedule $N$ no processing is done during time 0 through $m-2$, then all the small jobs run from time $m-2$ to $m-1$, then $m$ big jobs run from time $m-1$ through $2m-1$, and finally job $J_{j'}$ runs from time $2m-1$ through time $3m-1$. Thus for job $J_{j'}$, $C_{j'}^N = [(3m-1)/(m+2)]C_{j'}^P$, which, for large $m$ approaches a ratio of 3.

Corollary 2.5 is not tight, and in fact has recently been improved by Chakrabarti, Phillips, Schulz, Shmoys, Stein and Wein to $\frac{7}{3}$ [3].

## 2.4 Fractional Schedules

Define a *fractional* preemptive schedule as one in which a job $J_j$ can be scheduled simultaneously with several other jobs on one machine, where each job receives some fraction of the machine's resources and the sum of all the fractions assigned to a machine at any time is at most one. In other words, let $\gamma_j^i(t)$ be the fractional amount of machine $M_i$ assigned to $J_j$ at time $t$, with time taken as continuous, and let $T$ be the completion time of a schedule. A fractional schedule satisfies $\sum_j \gamma_j^i(t) \leq 1$ for all machines $i$ and times $t$; $\sum_i \int_{t=0}^{T} s_{ij}\gamma_j^i(t) = p_j$ for all jobs $j$, and for any pair of $t, j$ only one of the $\gamma_j^i(t)$ is non-zero. The following corollary follows by proofs identical to those of Theorems 2.1 and 2.4.

**Corollary 2.6** *Suppose we apply algorithm* Convert *to an input consisting of a fractional preemptive schedule $P$ for a [one machine/parallel identical machine] problem. Then Theorems 2.1 and 2.4 still hold.*

## 3 Minimizing $\sum C_j$ on One Machine

In this section we consider the $\mathcal{NP}$-hard problem of nonpreemptively scheduling $n$ jobs with release dates on one machine so as to minimize the average completion time. We first give a two-approximation algorithm, and then show that it can be modified slightly to be an on-line scheduling algorithm. We then prove a lower bound on the performance of *any* on-line scheduling algorithm for this problem.

It is well known how to solve, in polynomial time, $1|r_j, pmtn|\sum C_j$ using the shortest processing time rule: always be processing the job with the shortest remaining processing time [1]. Using this schedule as input to CONVERT, and applying Corollary 2.2, we get the following result:

**Theorem 3.1** *There is a polynomial-time* 2*-approximation algorithm for* $1|r_j|\sum C_j$.

We cannot immediately obtain analogous results for $1|r_j|\sum w_j C_j$, since the preemptive version of this problem is $NP$-hard. We will see in Section 5 how to obtain an $(8+\epsilon)$-approximation for $1|r_j, pmtn|\sum w_j C_j$, and thus a $(16+\epsilon)$-approximation algorithm for $1|r_j|\sum w_j C_j$.

The algorithm of Theorem 3.1 can be easily modified to be an *on-line* 2-approximation algorithm. The preemptive shortest processing time algorithm is on-line, since it needs no information about jobs to be released in the future. The nonpreemptive on-line scheduler simulates the preemptive shortest processing time algorithm. If job $J_j$ finishes at time $C_j^p$ in the simulated preemptive schedule, then it simply runs starting at time $C_j^P$ in the on-line, nonpreemptive schedule, provided no other job is already running at this time. If another job (that finishes earlier in the preemptive schedule) is running at time $C_j^P$, then job $J_j$ is queued and run as soon as all jobs preceding it in the queue are run. The proof of Theorem 2.1 then yields the following corollary:

**Corollary 3.2** *There is an* on-line *nonpreemptive* 2*-approximation algorithm for* $1|r_j|\sum C_j$.

**Proof:** The proof is identical to that of Theorem 2.1 except that $p_j$ extra units are inserted into the schedule at time $C_j^P$ instead of $p_j - k_j$. It is still true that

$$C_j^N \leq C_j^P + \sum_{k: C_k^P \leq C_j^P} p_k.$$

■

We now give a lower bound on the performance of any on-line algorithm for this problem.

**Theorem 3.3** *No on-line nonpreemptive $\rho$-approximation algorithm exists for* $1|r_j|\sum C_j$ *with* $\rho < \frac{3}{2}$.

**Proof:** Assume that an on-line algorithm $A$ has a performance guarantee of $1+c$ for some $c < 1$. Assume that this algorithm is given a job $J_0$ with $p_j = x$ and $r_j = 0$. Then we claim that algorithm $A$ must start $J_0$ by time $cx$. If it did not, and this were the only job ever released, then the schedule would complete at some time greater than $(1+c)x$, while the optimal off-line schedule would have run the job immediately, for a completion time of $x$. Thus the approximation ratio is greater than $(1+c)x/x = (1+c)$, contradicting the fact that the approximation ratio of $A$ is at most $(1+c)$. Thus we can assume that $A$ must start $J_0$ by time $cx$.

Now consider the case where, in addition to $J_0$, there are $x$ jobs with $p_j = 1$ and $r_j = cx + 1$. In this case, the best thing an on-line algorithm that has to start $J_0$ by time $cx$ can do is to run $J_0$

starting at time 0, followed by (in the best case) the other jobs, yielding a total completion time of

$$x + \sum_{i=1}^{x}(x+i) = \frac{3}{2}x^2 + \frac{3}{2}x$$

while the optimal schedule processes all the jobs with $p_j = 1$ first and then the large job, for a total completion time of

$$\left(\sum_{i=1}^{x}(cx+1+i)\right) + (cx+1+2x) = (c+\frac{1}{2})x^2 + (c+\frac{7}{2})x + 1.$$

As $x$ gets large, this ratio tends towards $\frac{3}{2c+1}$, which is greater than $1+c$ (yielding a contradiction) whenever $c < \frac{1}{2}$. ■

Hoogeveen and Vestjens [10] and Stougie [23] have recently improved this lower bound for on-line algorithms to 2, which implies our algorithm is, in the worst case, an optimal on-line algorithm.

# 4 Minimizing $\sum C_j$ on Identical Parallel Machines

We now turn to the problem of scheduling jobs with release dates on parallel machines, so as to minimize the average completion time. The nonpreemptive versions of these problems are $\mathcal{NP}$-hard due to the $\mathcal{NP}$-hardness of the one-machine problem. When preemption is allowed, as noted, the one machine problem is solvable in polynomial-time, using the shortest processing time rule, but the scheduling of even two identical machines is $\mathcal{NP}$-hard [5].

In this section we first give a 2-approximation algorithm for $P|r_j, pmtn|\sum C_j$. The algorithm is the multiple-machine generalization of the one-machine algorithm: always be running the $m$ jobs with the shortest remaining amount of work. We call this the *SPT* algorithm and the resulting schedule an *SPT* schedule. We will then combine this algorithm with algorithm CONVERT to obtain a 6-approximation algorithm for $P|r_j|\sum C_j$.

## 4.1 Preemptive Schedules

Our approach for showing that SPT is a 2-approximation algorithm for $P|r_j, pmtn|\sum C_j$ will be to show that by any time $2t$, SPT has finished at least as many jobs as *any* schedule could have finished by time $t$. Applying this fact to all time units allows us to conclude that if the $x$th job finishes at time $t_x$ in the optimal schedule, then the $x$th job to finish in the SPT schedule (not necessarily the same job) will finish by time $2t_x$. Thus the sum of completion times of the SPT schedule is at most 2 times the sum of completion times of the optimal schedule. More formally, given a schedule $S$ and a set of jobs $J$, let $f_S(J,t)$ be the number of jobs completed by time $t$ when the set of jobs $J$ is scheduled according to schedule $S$. Letting OPT denote the optimal schedule, we will show that $f_{\mathrm{SPT}}(J,2t) \geq f_{\mathrm{OPT}}(J,t)$ for $0 \leq t \leq T$, where $T$ is the length of the optimal schedule.

Our arguments will require two technical lemmas about ordered multisets. The proofs follow easily from case analysis and thus are left to the reader.

Let $Z_+^\infty$ be the set consisting of all non-negative integers and $\infty$. Given a multiset $X$, in which each element is a member of $Z_+^\infty$, we denote the $i^{\text{th}}$ smallest element of $X$ by $X[i]$, and we denote the number of elements in $X$ by $|X|$. Given two sets $X$ and $Y$ of non-negative integers, we say that $X \preceq Y$ if $|X| = |Y|$ and $X[i] \leq Y[i], i = 1, \ldots, |X|$.

**Lemma 4.1** *Let $X$ and $Y$ be multisets in which each element is in $Z_+^\infty$, with $X \preceq Y$. Let $x$ and $y$ be elements of $Z_+^\infty$ with $x \leq y$. Let $X' = X \cup \{x\}$ and $Y' = Y \cup \{y\}$. Then $X' \preceq Y'$.*

We now define an operation $\mathrm{dec}_m(X)$ which we apply to a multiset $X$ to decrement the $m$ smallest non-zero elements of $X$. Let $\ell_x$ be the smallest index such that $X[\ell_x] > 0$. We define $\mathrm{dec}_m(X)$ as follows:

$\underline{\mathrm{dec}_m(X)}$: Let $h_x = \min\{\ell_x + m, |X|\}$. Let $X[i] \leftarrow X[i] - 1$, for $i = \ell_x, \ldots, h_x$.

**Lemma 4.2** *Let $X$ and $Y$ be multisets in which each element is in $Z_+^\infty$, with $X \preceq Y$. Then $\mathrm{dec}_m(X) \preceq \mathrm{dec}_m(Y)$.*

In order to compare two schedules which have different prefixes, we will require the following lemma which shows that running SPT on a subset of a job set can only decrease the number of jobs completed by a specified time. We will use the convention that job set $J'$ has jobs $J'_j$ with processing time $p'_j$ and release dates $r'_j$, and use the analogous convention for $J''$.

**Lemma 4.3** *Let $J''$ and $J$ be two sets of jobs with $J'' \subseteq J$. Let $T$ be the completion time of SPT run on job set $J$. Then $f_{\mathrm{SPT}}(J,t) \geq f_{\mathrm{SPT}}(J'',t)$, $t = 1, \ldots, T$.*

**Proof:** Assume the jobs are indexed so that $J_j = J''_j$, $1 \leq j \leq |J''|$. We form job set $J'$ by augmenting $J''$ with jobs $J'_j$, $j = |J''|+1, \ldots, |J|$. These extra jobs have $r'_j = r_j$ and $p'_j = \infty$. So $|J'| = |J|$, but some of the jobs in $J'$ have infinite processing time. Clearly $f_{\mathrm{SPT}}(J',t) = f_{\mathrm{SPT}}(J'',t)$, for $0 \leq t \leq T$, since the jobs in $J' \backslash J''$ never finish, and they never run if a job with finite processing time could run instead. Thus it will suffice to show that for $0 \leq t \leq T$

$$f_{\mathrm{SPT}}(J,t) \geq f_{\mathrm{SPT}}(J',t). \tag{1}$$

Let $\mathrm{res}_t^J(j)$ be the total amount of processing time remaining on job $J_j$ at time $t$ in an SPT schedule for $J$. Note that if job $J_j$ is finished at time $t$, then $\mathrm{res}_t^J(j) = 0$. Finally we let $R_t^J$ be the ordered multiset of remaining processing times of all jobs that have been released by time $t$, i.e.

$$R_t^J = \{\mathrm{res}_t^J(j) : 0 \leq j \leq |J| \text{ and } r_j \leq t\}.$$

We make the analogous definitions for $J'$.

We now claim that

$$R_t^J \preceq R_t^{J'} \tag{2}$$

implies (1). This is because if $R_t^J \preceq R_t^{J'}$, then $R_t^J$ must contain at least as many zeroes as $R_t^{J'}$, and hence at least as many jobs have been completed for job set $J$ than for job set $J'$. We now prove (2) by induction on $t$.

Before time 0 (call this time $-1$), both $R_{-1}^J$ and $R_{-1}^{J'}$ are empty and (2) is trivially true. To form $R_t^J$ from $R_{t-1}^J$, we must perform two steps. First, for each job $J_j$ with $r_j = t$, we release the job, by adding an element with value $p_j$ to $R_t^J$. Second, we process one unit of the (up to) $m$ jobs with the smallest positive remaining processing times. We perform the same steps for $R_t^{J'}$.

The first step consists of adding an element $p_j$ to $R_t^J$ and an element $p'_j$ to $R_t^{J'}$ for each job $J_j$ and corresponding job $J'_j$ such that $r_j = t$. By the definitions of $J$ and $J'$, we know that either $p_j = p'_j$ or $p'_j = \infty$. In either case $p_j \leq p'_j$ and we can apply Lemma 4.1 to see that after the release of each job, it remains true that $R_t^J \preceq R_t^{J'}$.

The second step consists of applying $\mathrm{dec}_m(J)$ and $\mathrm{dec}_m(J')$ and hence, after processing the jobs, by Lemma 4.2, it remains true that $R_t^J \preceq R_t^{J'}$. ■

We are now ready to prove that SPT finishes as many jobs by time $2t$ as any schedule $S$ does by time $t$.

**Lemma 4.4** *For any job set $J$, any schedule $S$ for $J$ of length $T$ and any $0 \le t \le T$,*

$$f_{\mathrm{SPT}}(J, 2t) \ge f_{\mathrm{S}}(J, t).$$

**Proof:** Consider the set of jobs $J(S,t)$ finished by a schedule $S$ at time $t$. Since $J(S,t) \subseteq J$, by Lemma 4.3, it suffices to show that

$$f_{\mathrm{SPT}}(J(S,t), 2t) \ge f_{\mathrm{S}}(J, t).$$

Suppose that at time $t$, SPT has not finished all the jobs in $J(S,t)$. Let job $J_j$ be the job with the largest remaining processing time at time $t$, and call that processing time $x$. Then there are at least $x$ units of time when SPT was processing $m$ jobs other than $J_j$. This is because $r_j + p_j \le t$ or $J_j$ could not have been finished by time $t$. SPT always runs $m$ jobs when there are $m$ available, so during the $x$ units $J_j$ wasn't processed, SPT did $mx$ units of work on other jobs. Since $S$ can complete $J(S,t)$ using no more than $mt$ units of work in time $t$, there are no more than $m(t-x)$ units of work from $J(S,t)$ which have not been processed by SPT at time $t$. At time $t$ all jobs in $J(S,t)$ have been released and can be processed smallest-to-largest without interruption. Since all the remaining $m(t-x)$ units of work belong to job fragments of size less than $x$, they will be scheduled before the last $x$ units of job $J_j$, starting at time t. By standard averaging arguments, a machine will be available for job $J_j$ no later than time $t + m(t-x)/m = 2t - x$. Therefore job $J_j$ with remaining processing time $x$ will complete by time $2t$, and since $J_j$ is the last job to finish in the SPT schedule, all jobs in $J(S,t)$ finish by time $2t$. ■

Lemma 4.4 is essentially tight as illustrated by the following example. Suppose at time 0 one job with $p_j = t$ and $(m-1)t$ jobs with $p_j = 1$ are released. The makespan of this job set is $t$: run the long job on one machine and pack the unit jobs on the remaining $m-1$ machines. SPT will run the $(m-1)t$ unit jobs across all machines from time 0 to $t-1$. The long job will not complete until time $2t-1$.

We can now argue that SPT is a 2-approximation algorithm for $P|pmtn, r_j|\sum C_j$.

**Theorem 4.5** *SPT is an on-line 2-approximation algorithm for $P|r_j, pmtn|\sum C_j$.*

**Proof:** To see that SPT is a 2-approximation algorithm, we apply Lemma 4.4 for the case where $S$ is OPT. Given input $J$, if $C^{\mathrm{OPT}}(k)$ is the $k^{\mathrm{th}}$ completion time of OPT and $C^{\mathrm{SPT}}(k)$ is the $k^{\mathrm{th}}$ completion time in SPT, then by Lemma 4.4, we have $C^{\mathrm{SPT}}(k) \le 2C^{\mathrm{OPT}}(k)$. Therefore $\sum_{j \in J} C_j^{\mathrm{SPT}} \le 2 \sum_{j \in J} C_j^{\mathrm{OPT}}$.

SPT is an online algorithm, since the decisions about which jobs to run are made independent of the future. ■

We can apply algorithm Convert and Theorem 2.4 to the preemptive SPT schedule to obtain a 6-approximation algorithm for the nonpreemptive problem.

**Theorem 4.6** *An SPT schedule given as input to* Convert *is an on-line 6-approximation algorithm for $P|r_j|\sum C_j$.*

We now explain how to implement the off-line version of SPT on parallel machines in $O(n \log n)$ time. We will make use of a data structure called a *min-max heap*. A *min-max heap* is a data structure that supports INSERT, FIND-MIN, DELETE-MIN, FIND-MAX and DELETE-MAX. It is a simple exercise [4] to create a min-max heap using two traditional heaps, so that all operations run, in the worst case, in $O(\log n)$ time.

We will use two min-max heaps $H_1$ and $H_2$ and maintain, at all times, a current time. At any time, heap $H_1$ contains all jobs that are released, but have not completed and are not currently running. Note that $H_1$ may contain jobs that have run for a while but were then preempted. Heap $H_1$ indexes the jobs by their remaining processing time. Heap $H_2$ contains all jobs that are currently running. It indexes the jobs by their predicted completion times, that is, the current time plus their remaining processing time. We also maintain a list of the jobs that have not yet been released, sorted by release date.

Given these two heaps, we perform a discrete event simulation. There are two possible next events, a release date, or a job completion. We can decide which is next by comparing the next job to be released to the result of a FIND-MIN on $H_2$. If the next event is a job completion, we do a DELETE-MIN on heap $H_2$ to remove the job that completes. There is now an empty machine, so we do a DELETE-MIN on $H_1$ to obtain the job with shortest remaining processing time and INSERT that job into $H_2$.

If the next event is a release date, we must decide if any of the currently running jobs should be preempted. Assume that the list of jobs released at this time is sorted by processing time. We repeatedly compare the release date plus processing time of the first job on this list to the result of a FIND-MAX from $H_2$. If the just-released job's value is smaller, we do a DELETE-MAX from $H_2$ and INSERT that job into $H_1$. We then INSERT the just-released job into $H_2$, and repeat the procedure for the next job of the list of just-released jobs. Eventually a just-released job will have a smaller predicted completion time than that of any already-running job, and then we will INSERT the rest of the list into $H_1$.

We omit the details regarding what to do when various data structures are empty, and the proof of correctness, as they are straightforward. We now bound the running time. Observe that each DELETE-MIN or DELETE-MAX corresponds to a job completion or a preemption. Each job completes at most once, and there are at most $n$ preemptions overall, as each job can only be the preempting job at most once. The number of INSERT operations is equal to the number of DELETE-MIN and DELETE-MAX operations. Each FIND-MIN or FIND-MAX can be charged to a DELETE-MIN, DELETE-MAX or a release date. Hence the total running time is bound by $O(n \log n)$. Given the output of this algorithm, which includes the completion times in sorted order, CONVERT can then be implemented in $O(n)$ additional time.

## 5 Unrelated Machines, Weighted Completion Times and Linear Programming Relaxations

In this section we introduce an integer program that is closely related to the preemptive versions of our scheduling problem. We also give a method of rounding the fractional solutions generated by its linear programming relaxation to valid preemptive schedules, and obtain the first constant-factor approximation algorithms for a very general form of the preemptive scheduling problem, $R|r_j, pmtn| \sum w_j C_j$. By application of algorithm CONVERT from Section 2, we obtain the first constant-factor approximation algorithms for $1|r_j| \sum w_j C_j$ and $P|r_j| \sum w_j C_j$.

Our approach utilizes a generalization of bipartite matching, which we sketch here in the context of identical parallel machines. We will divide a job $J_j$ of processing time $p_j$ into $p_j$ *units*; on one side of the partition we will include a node for each unit of each job, and on the other side of the partition we will place a node $(M_i, t)$ for each unit of time $t$ on each machine $M_i$, where we include every possible unit of time during which a job might need to be processed in an optimal schedule. An edge will be placed in the bipartite graph between the $k$th unit of $J_j$ and $(M_i, t)$ if and only if $r_j \leq t-(k-1)$, i.e. the scheduling of that unit on that machine at that time is feasible with respect to the release date. We assign a cost of 0 to all edges except for those that represent the scheduling of the last unit of a job; to these we assign a cost of $w_j t$, namely the weighted completion time of $J_j$ if its last unit is scheduled using this edge. We seek the minimum-weight matching of all job units *subject to the constraint* that for a job of processing time $x$, $x-1$ units of it have been completed before the last unit is processed.

The resulting integer program will be quite large; in fact, unless the job processing times are polynomial in $n$ and $m$ this formulation will not be polynomial in the input size. In Section 5.2 we show that if the job processing times are not polynomial, we can still obtain schedules of good approximate quality by solving a number of subproblems of the original scheduling problem, each of which is scaled to yield a polynomial-size linear programming formulation.

Note that this formulation models schedules in which preemptions are only allowed between units, and does not capture the possibility of preemption within a unit. We will show that the fractional solutions to the linear programming relaxation can capture this possibility with only a slight degradation in quality of solution. Intuitively, this is because there is little difference between a fractional unit of a job and a preempted unit of a job.

For convenience, we will scale the data so that $C_j \geq 1$ for all $j$. In particular, for the rest of this section we assume that $p_j/s_{ij} \geq 1$. We also assume $s_{ij} \geq 1$ for every job $J_j$ and machine $M_i$, and that all the $p_j$ and $r_j$ are integral. If the input is not in this form, it can easily be scaled. Although all of our algorithms are polynomial-time algorithms, the polynomials involved can be quite large. Therefore, for the rest of this section we do not discuss running times.

## 5.1 Jobs of Polynomial Processing Time

Throughout this section we assume without loss of generality that $m \leq n$. We also assume that all $p_j$ and $s_{ij}$ are bounded by a polynomial in $n$; in Section 5.2 we show how to remove this assumption.

We introduce the following integer program $\mathcal{IP}$. Let $T = \{t | \exists\, j \text{ s.t. } r_j \leq t \leq r_j + np_{\max}\}$; that is, $T$ is a set of times that includes all the times that any job might ever be running. Note that by the assumptions of this section, $T$ is of polynomial size. We will use variables $\{x_{ijkt}\}, 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq p_j, t \in T$; $x_{ijkt} = 1$ will represent the scheduling of unit $k$ of job $J_j$ on machine $M_i$ during $(t-1, t]$; note that the range of $k$ depends on $j$.

$\mathcal{IP}$ is defined as follows.

$$\text{minimize} \sum_{j=1}^{n} w_j \sum_{i=1}^{m} \sum_{\substack{t \in T \\ k=p_j}} t \cdot x_{ijkt}$$

subject to

$$x_{ijkt} = 0 \qquad \text{if } t \leq r_j + k - 1 \tag{3}$$

$$\sum_{i=1}^{m} \sum_{t \in T} x_{ijkt} = 1 \qquad j = 1, \ldots, n;\ k = 1, \ldots, p_j \tag{4}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{p_j}\frac{x_{ijkt}}{s_{ij}} \leq 1 \qquad j=1,\ldots,n;\ t\in T \tag{5}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{p_j}\frac{x_{ijkt}}{s_{ij}} \leq 1 \qquad i=1,\ldots,m;\ t\in T \tag{6}$$

$$\sum_{i'=1}^{m}\sum_{\ell=1}^{k-1}\sum_{t'=1}^{t-1} x_{i'j\ell t'} \geq (k-1)x_{ijkt} \qquad k=p_j; i=1,\ldots,m; j=1,\ldots,n, t\in T \tag{7}$$

$$x_{ijkt} \in \{0,1\} \tag{8}$$

To understand the objective function, note that if the $x_{ijkt}$ are indeed $0-1$ variables then $\sum t \cdot x_{ijkt}$ will evaluate to 0 if unit $k$ of job $J_j$ is not scheduled on machine $M_i$ and will otherwise evaluate to the time unit during which unit $k$ is scheduled. Constraints (4)-(7) can be understood by viewing the $x_{ijkt}$ as indicator variables; by summing over different combinations of subscripts we can constrain the number of units that are processed from one job in a particular time unit (5), or the number of units that are processed on one machine in a particular time unit (6). Alternatively, as in constraints (4) and (7), we may write a constraint that ensures at least a certain amount of processing of a certain kind is done in a time period of interest.

The following lemma, which formalizes the intuition behind this formulation, establishes a precise relationship between solutions to $\mathcal{IP}$ and slightly restricted solutions to $1|r_j, pmtn|\sum w_jC_j$ and $P|r_j, pmtn|\sum w_jC_j$.

**Lemma 5.1** *Given an instance of $1|r_j, pmtn|\sum w_jC_j$ or $P|r_j, pmtn|\sum w_jC_j$, in which preemption of a single unit of a job is not allowed, there is a schedule of total weighted completion time $z$ if and only if there is a feasible solution to $\mathcal{IP}$ of cost $z$.*

**Proof:** Note that in these two environments, we have $s_{ij} = 1$ for all $i$ and $j$. Clearly any preemptive schedule is a feasible solution to $\mathcal{IP}$. Given a feasible solution to $\mathcal{IP}$, constraint (4) guarantees that every unit of every job is scheduled. Constraints (5) and (8) guarantee that at most one unit of a job is processed during any time unit, whereas constraints (6) and (8) guarantee that at most one unit of one job is assigned to one unit of time on any machine. Constraint (7) guarantees that when the last unit of a job is scheduled the rest of the job has been scheduled already. This guarantees that the objective function accurately reflects the total weighted completion time of the schedule. Note that as long as (5) and (6) are satisfied, the order in which the first $p_j - 1$ units of $J_j$ are scheduled is not important. ∎

When we consider the more general case of unrelated machines, and we allow a job to be preempted within a unit, the relationship between solutions to $\mathcal{IP}$ and valid schedules is slightly more complicated; in fact, we will establish a relationship between schedules in this setting and *fractional* solutions to $\mathcal{LP}$, a linear programming relaxation of $\mathcal{IP}$, which is defined by constraints (3) - (7) and the linear relaxation of (8):

$$0 \leq x_{ijkt} \leq 1, \quad 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq p_j, t \in T.$$

In establishing this relationship we must handle several technical difficulties, including preemption within a unit, jobs preemptively completing at fractional times (our objective function only models integral completion times), and the processing of a unit of a job extending from one time unit to the next in the unrelated machines model. In addition, solutions to $\mathcal{LP}$ cannot immediately be understood as schedules, since a number of jobs may be fractionally spread over a number of machines in one unit, and further scheduling may be required to insure that different fractions of a job do not run simultaneously in that unit of time. We resolve these potential difficulties with the following lemma.

**Lemma 5.2** *Given a schedule $S$ for an instance of $R|r_j, pmtn|\sum w_jC_j$ in which $J_j$ completes at time $C_j^S$, there exists a solution to $\mathcal{LP}$ in which $x_{ijkt} = 0$ if $t > \lceil C_j^S \rceil$. Furthermore, given a fractional solution to $\mathcal{LP}$, let $t_j = \max\{t|x_{ijkt} > 0$ for any $i, k\}$. There exists a schedule for the corresponding instance of $R|r_j, pmtn|\sum w_jC_j$ in which $J_j$ finishes by time $t_j$, $j = 1, \ldots, n$.*

**Proof:** We first establish the conversion from $S$ to a (fractional) solution for $\mathcal{LP}$. Consider job $J_j$ with size $p_j$, and define $F(j,t)$ to be the amount of processing done on job $J_j$ by time $t$, where $t$ is a positive real number that is at most the length of $S$. Let $t_k^{'}$ be the smallest $t$ such that $F(j,t) \geq k$ and let $t_k^{''}$ be the largest $t$ such that $F(j,t) \leq k+1$. The $k$th unit of job $J_j$ is precisely the processing of job $J_j$ that occurs between times $t_k^{'}$ and $t_k^{''}$. We then set the appropriate $x_{ijkt}$'s to correspond to the fractions of the $k$th unit processed in this time interval, that is, $x_{ijkt} = \alpha$ if an $\alpha$ fraction of the $k$th unit of job $J_j$ is scheduled on machine $M_i$ during time interval $(t-1, t]$. It is clear that these $x_{ijkt}$ are a valid solution with the desired properties of the lemma.

Conversely, given a fractional solution to $\mathcal{LP}$ in which $x_{ijkt} = 0$ for $t > t_j$, $j = 1, \ldots, n$, we build a schedule as follows. Consider a unit of time $t$ on a machine $i$. Let $O_{ijt} = \frac{1}{s_{ij}}\sum_{k=1}^{p_j} x_{ijkt}$; $O_{ijt}$ is the total processing time allotted to $J_j$ on $M_i$ during $(t-1, t]$ by the linear program. We seek to schedule $J_j$ for $O_{ijt}$ time during time $(t-1, t]$ on machine $M_i$. By constraint (6) of $\mathcal{LP}$ $\sum_{j=1}^{n} O_{ijt} \leq 1$, but this is not sufficient to guarantee the existence of a valid schedule; we must also construct a schedule in which no two machines are running pieces of $J_j$ simultaneously, for any $j$. We can accomplish this by an application of open shop theory [6, 16].

In the open-shop scheduling environment each job is made up of a number of *operations*, and each operation must be scheduled on a specific one of $m$ machines; no two operations of one job may be scheduled simultaneously. The $O_{ijt}$ define an open-shop problem for each unit of time; $O_{ijt}$ is $J_j$'s operation on machine $M_i$. Using a result of Gonzales and Sahni [6], we can create a preemptive schedule with at most $m^4$ preemptions per processor , with length $\max\{\max_{j'}\sum_{i'} O_{i'j't}, \max_{i'}\sum_{j'} O_{i'j't}\}$, which by constraints (5) and (6) is at most 1. Thus the jobs can be scheduled in the time unit to which they are assigned. In this fashion, unit by unit, we create a valid schedule for each unit of time; it is clear that $J_j$ finishes by time $t_j$ for all $j = 1, \ldots, n$. ■

Note that in Lemma 5.2 we make no claims about the relationship between OPT($\mathcal{LP}$), the optimal value of the objective function of $\mathcal{LP}$, and the total weighted completion time of the schedule that is derived from the optimal solution to $\mathcal{LP}$. The latter may in fact be much larger than the former. We now show how to obtain schedules from $\mathcal{LP}$ of total weighted completion time within a constant factor of OPT($\mathcal{LP}$).

Consider a solution to $\mathcal{LP}$; in this (possibly) fractional solution each job contributes to the objective function the weighted sum of the different fractionally scheduled pieces of the last unit; we call this quantity the job's *fractional completion time*, and define it precisely as

$$\bar{C}_j = \sum_{t \in T}\sum_{i=1}^{m} t \cdot x_{ijp_jt},$$

and observe that the value of the objective function of $\mathcal{LP}$ is precisely $\sum_j w_j\bar{C}_j$ .

The fractional completion time of $J_j$ is a weighted sum of fractional completion times of different pieces of $J_j$'s last unit, and does not represent the actual point in the schedule at which the last fractional unit of the job finishes processing. In fact, it is possible that the completion time of $J_j$ is much larger than its *fractional* completion time. We now show how to round fractional solutions to $\mathcal{LP}$ of total fractional weighted completion time $\sum_j w_j \bar{C}_j$ to schedules with total weighted completion time at most a factor of 4 larger.

**Lemma 5.3** *Given an instance of $R|r_j, pmtn|\sum w_j C_j$, and the solution to the corresponding linear program $\mathcal{LP}$ of total fractional weighted completion time $\sum_j w_j \bar{C}_j$, there is a polynomial time algorithm which produces a schedule with total weighted completion time at most $4\sum_j w_j \bar{C}_j$.*

**Proof:** Consider a job $J_j$ of size $p_j$ and consider the $p_j$th unit of $J_j$. In the solution to $\mathcal{LP}$, this unit may be assigned fractionally to a number of machines at different times. Let time $t_j^*$ be the unit of time at the end of which, for the first time, $1/2$ of this $p_j$th unit is assigned; specifically,

$$t_j^* = \min\{t | \sum_{\ell=1}^{t} \sum_{i=1}^{m} x_{ijp_j\ell} \geq \frac{1}{2}\}.$$

At least $\frac{1}{2}$ of the processing of the last unit of job $J_j$ occurs after time $t_j^*$ and hence we can conclude that

$$\bar{C}_j \geq \frac{1}{2} t_j^*. \tag{9}$$

By constraint (7) we know that by time $t_j^* - 1$ at least half of the first $k-1$ units of $J_j$ have been assigned. We now show how to build a preemptive schedule in which job $J_j$ completes by time $2t_j^*$. Consider the schedule $S$ that, by Lemma 5.2, corresponds to the solution to $\mathcal{LP}$. We first remove from $S$ all pieces of the first $k-1$ units of $J_j$ that were scheduled in $S$ after time $t_j^* - 1$, and any pieces of the $k$th unit that were scheduled after time $t_j^*$. In the schedule that remains we have scheduled at least half of each unit of $J_j$ by time $t_j^*$. We now take each unit of time and double its size, and double the processing time of each scheduled piece of $j$ as well. As a result, a piece of a job that ran from time $s$ to $t$ now runs from time $2s$ to $2t$. We now have a schedule in which all of each unit of $J_j$ is scheduled, and in which all units of $J_j$ complete by time $2t_j^*$. Note in fact that we may have reserved more space in the schedule than is necessary for some units of $J_j$, but if we remove any "excess" processing, this only decreases $J_j$'s completion time. The fact that $\frac{t_j^*}{2}$ is a lower bound on the fractional completion time of job $J_j$ completes the proof of the lemma. ■

Due to the rounding of completion times to integral values (see Lemma 5.2), OPT($\mathcal{LP}$) is not necessarily a lower bound on the weighted completion time of the optimal schedule. However, we can show that it is close, and thus we can use Lemma 5.3 to obtain an approximation algorithm.

**Theorem 5.4** *If $p_{\max}$ is bounded above by a polynomial in $n$, there exists a polynomial-time $(4+\epsilon)$-approximation algorithm for $R|r_j, pmtn|\sum w_j C_j$ for any constant $\epsilon$.*

**Proof:** Consider an optimal schedule $S$ in which $J_j$ finishes at time $C_j^S$. By Lemma 5.2 there exists a solution $\{x_{ijkt}\}$ to $\mathcal{LP}$ in which $t_j = \lceil C_j^S \rceil$; therefore

$$\mathrm{OPT}(\mathcal{LP}) \leq \sum_{j=1}^{n} w_j (C_j^S + 1) \leq \sum_{j=1}^{n} w_j C_j^S + \sum_{j=1}^{n} w_j.$$

Using Lemma 5.3 we construct a schedule $S'$ such that

$$\sum_j w_j C_j^{S'} \leq 4\mathrm{OPT}(\mathcal{LP}) \leq 4(\sum_j w_j C_j^S + \sum_j w_j).$$

By the assumption that all $C_j \geq 1$, this immediately yields a 8-approximation algorithm. Note, however, that $\sum w_j$ is merely a constant; in order to obtain an improved bound we will scale up the $p_j$ and $r_j$ by a larger amount, or, equivalently, subdivide time more finely. This will increase the minimum non-zero $C_j$ value and ensure that rounding up to an integral time unit has a smaller effect on the final solution.

To achieve the quality of approximation guarantee claimed in the statement of the Theorem, we wish to ensure that

$$\sum_j w_j \leq \frac{\epsilon}{4} \sum_j w_j C_j^S.$$

We have assumed that $C_j^S \geq 1$; we may further scale up every $p_j$ and $r_j$ so that $C_j^S \geq \frac{4}{\epsilon}$ for all $j$, or equivalently refine our integer programming formulation by subdividing time into units of $\frac{\epsilon}{4}$. Whichever way we view the process, we obtain

$$\sum_j w_j \leq \frac{\epsilon}{4} \sum_j w_j C_j^S$$

and therefore

$$\sum_j w_j C_j^{S'} \leq (4+\epsilon) \sum_j w_j C_j^S.$$

■

Note that, due to the subdivision of time into units of $\frac{\epsilon}{4}$, the running time of the algorithm is polynomial in $\frac{1}{\epsilon}$.

## 5.2 Large Jobs

Since the $p_j$ and $s_{ij}$ are input in binary and in general need not be polynomial in $n$ and $m$, the algorithm of Theorem 5.4 cannot be applied directly to all instances, since it would yield superpolynomial-size formulations. Therefore we must find a way to handle very large jobs without impacting significantly on the quality of solution.

One technique for dealing with jobs of greatly varying processing times is to partition the jobs into classes, in which each class contains jobs of similar size. In both the current best approximation algorithms for minimizing makespan on parallel machines [9], and minimizing makespan in a job shop [21], jobs are partitioned into two classes, "big" and "small." These algorithms first schedule the large jobs, which are scaled to be in a polynomially-bounded range, and then schedule the small jobs in a different manner and show that their net contribution is not significant. In the minimization of average weighted completion time, however, we must be more careful, since the small jobs may have large weights and make a significant contribution to the weighted completion time.

To handle the case of large jobs we employ several steps, each of which increases the average weighted completion time by a small constant factor; the overall impact on the quality of approximation is a factor of $(2+\epsilon)$. The basic idea is to characterize each job by the minimum value, taken over all machines, of its (release date + processing time) on that machine. Recall that $p_{ij}$ denotes the processing time of job $J_j$ on machine $M_i$, i.e. $p_{ij} = p_j/s_{ij}$. Let $m(J_j) = \min_i(p_{ij} + r_j)$. We will group the jobs together based on the size of their $m(J_j)$. The jobs in each group can be scaled down to be of polynomial size and thus we can construct a schedule for the scaled down versions of each group. We then scale the schedules back up, correct for the rounding error, and show that this does not affect the quality of approximation by more than a constant factor. We then apply Lemma 5.5 (see below) to show that the makespan can be kept short simultaneously.

The schedule for the complete instance will be constructed by constructing two schedules in which specific groups are scheduled consecutively, and then interleaving these two schedules. The key point will be that we can, for the schedules for each group, keep both the average completion time and the makespan small *simultaneously*. Thus the jobs in the groups that are scheduled early will only delay the groups scheduled later by a small amount.

We now proceed in greater detail. For much of the section we will let $\alpha > 1$ denote an arbitrary parameter, whose value we will specify at the end of the section.

Let $\mathcal{J}^\ell = \{J_j | n^{2\ell-2} \leq m(J_j) \leq n^{2\ell}\}$. Note that there are at most $n$ nonempty $\mathcal{J}^\ell$, one for each of the $n$ jobs. We will employ the following lemma in order to keep the makespan of a schedule for $\mathcal{J}^\ell$ from growing too large.

**Lemma 5.5** *A schedule $S$ for $\mathcal{J}^\ell$ can be converted, in polynomial time, to a schedule $T$ of makespan at most $\alpha n^{2\ell+1}$ such that $C_j^T \leq (1 + \frac{1}{\alpha-1}) C_j^S \; \forall j \in \mathcal{J}^\ell$, where $\alpha > 1$.*

**Proof:** Consider all jobs that complete after time $(\alpha-1)n^{2\ell+1}$; remove them from the schedule, and, starting at time $(\alpha-1)n^{2\ell+1}$ schedule them arbitrarily on the machine on which they run most quickly. There are at most $n$ such jobs and each one has, for the machine $j$ on which they run most quickly, $r_j + p_{ij} \leq n^{2\ell}$. Clearly on any machine, the $k$th such job scheduled finishes by time $kn^{2\ell}$, and hence scheduling all the jobs in this manner adds at most $n(n^{2\ell}) = n^{2\ell+1}$ to the makespan of the schedule. Therefore any rescheduled job $J_j$ satisfies $C_j^T \leq \alpha n^{2\ell+1}$ and $C_j^S \geq (\alpha-1)n^{2\ell+1}$, and hence $C_j^T \leq (1 + \frac{1}{\alpha-1}) C_j^S \; \forall J_j \in \mathcal{J}^\ell$. ■

This lemma implies that there exist schedules which simultaneously have an average completion time close to optimal and whose makespan is bounded. Thus, we can modify $\mathcal{LP}$ to find such solutions of limited makespan without degrading the average weighted completion time by too much.

**Corollary 5.6** *Let $\mathcal{J}^p$ be a set of jobs such that $m(J_j) \leq n^{2p}$ for all $J_j \in \mathcal{J}^p$. Consider the linear program obtained by forming $\mathcal{LP}$ for the scheduling instance with job set $\mathcal{J}^p$, removing all variables $x_{ijkt}$ for which $t > \alpha n^{2p+1}$. Then the optimal solution to this modified $\mathcal{LP}$ has objective function value at most $(1 + \frac{1}{\alpha-1})$ times that of the optimal solution to $\mathcal{LP}$. Further, the modified $\mathcal{LP}$ is of polynomial size.*

We now turn to the problem of scheduling each $\mathcal{J}^\ell$ with a bounded guarantee on the average weighted completion time.

**Lemma 5.7** *There is a polynomial-time $(4 + \delta)$-approximation algorithm to minimize the total weighted completion time of a preemptive schedule for each $\mathcal{J}^\ell$, for any positive constant $\delta$. Furthermore, the makespan of the resulting schedule is $\alpha n^{2\ell+1}$.*

**Proof:** For $\ell \leq 3$ (actually any constant), we can carry out the following algorithm, which we call $\mathcal{A}$. We first apply Corollary 5.6, yielding a modified $\mathcal{LP}$ of polynomial size. In particular $|T| \leq n^{2\ell+2} \leq n^8$, and $p_{\max}$ is effectively set to $\alpha n^{2\ell+1}$. We then solve this modified $\mathcal{LP}$ and apply Theorem 5.4 to obtain a schedule with total completion time at most $(4+\epsilon)$ times optimal. We then apply Lemma 5.5 to bound the makespan by $\alpha n^{2\ell+1}$ while not increasing the average completion time by more than a factor of $(1 + \frac{1}{\alpha-1})$.

For $\ell > 3$ we would like to apply $\mathcal{A}$, to find an approximately optimal schedule $S^\ell$ for each $\mathcal{J}^\ell$. However, $\mathcal{A}$ cannot be applied directly to $\mathcal{J}^\ell$ since the processing times of the jobs involved may not be polynomial in $n$; we must apply $\mathcal{A}$ to a scaled version of $\mathcal{J}^\ell$.

For all $j$ such that $J_j \in \mathcal{J}^\ell$ and for all $i$, we create $J_j'$ by setting $p'_{ij} = \lfloor \frac{p_{ij}}{n^{2\ell-6}} \rfloor$ and $r'_j = \lfloor \frac{r_j}{n^{2\ell-6}} \rfloor$. Note that on at least one machine $i$, for each job $J_j$, $p'_{ij} \in [0, n^6]$ and $r'_j \in [0, n^6]$, and hence $m(J'_j) \leq 2n^6$ for all $j$.

We use $\mathcal{A}$ to obtain an approximately optimal schedule for the scaled version of $\mathcal{J}^\ell$, which we call $\mathcal{J}'^\ell$. Although some of the $p'_{ij}$ may still be large, we can apply Corollary 5.6 (with $p = 3$) to limit the size of the formulation to be polynomial in $n$. (Note that some of the $p'_{ij}$ may be 0, but it is still important to include a variable in the formulation for each job of processing time 0.) We then use the algorithm $\mathcal{A}$ to return a schedule.

We now interpret the schedule for $\mathcal{J}'^\ell$ as a schedule for $\mathcal{J}^\ell$; we schedule the jobs in $\mathcal{J}^\ell$ in the order dictated by the schedule for $\mathcal{J}'^\ell$, making sure to not start the processing of a job until its release date. We will show that this does not degrade the quality of approximation by more than a constant factor.

Let $p^*_j = n^{2\ell-6} p'_j$ and $r^*_j = n^{2\ell-6} r'_j$ be scaled-up versions of $p'_j$ and $r'_j$. Let $C^*_j$ be the completion time of job $J_j$ in the corresponding scaled up schedule and $C'_j$ be the completion time of job $J_j$ in the schedule for $\mathcal{J}'^\ell$. Since we have just, in effect, changed the "units," the quality of approximation provided by this scaled-up-schedule for its instance is just as good as that of the schedule for $\mathcal{J}'^\ell$. We now show that by further increasing the processing time of job $J_j$ from $p^*_j$ back up to $p_j$, yielding a schedule valid for $\mathcal{J}^\ell$, does not degrade the quality of approximation by more than a factor of $(1 + o(1))$. We denote the completion times of $J_j$ in this final schedule by $C_j$.

By definition, we know that $|r_j - r^*_j| < n^{2\ell-6}$ and $|p_{ij} - p^*_{ij}| < n^{2\ell-6}$. Let $B_j$ be the set of jobs that run earlier than job $J_j$ on the same machine. Then

$$C_j - C^*_j \leq \sum_{x \in B_j} (|r_x - r^*_x| + |p_{ix} - p^*_{ix}|) \leq n(n^{2\ell-6} + n^{2\ell-6}) = 2n^{2\ell-5}.$$

Thus in the conversion from $p^*_{ij}, r^*_j$ to $p_{ij}, r_j$ we add at most $2n^{2\ell-5}$ to the completion time of each job. However, by the definition of $\mathcal{J}^\ell$, we know that $n^{2\ell-2}$ is a lower bound on the completion time of job $J_j$. Thus by adding $2n^{2\ell-5}$ to the completion time of a job, we are multiplying that completion time by at most a factor of $(1 + \frac{2}{n^3})$, and hence

$$C_j \leq C^*_j \left(1 + \frac{2}{n^3}\right).$$

Also note that since the makespan of a schedule is equal to the completion time of the last job to finish, this conversion multiplies the makespan by at most a factor of $(1 + \frac{2}{n^3})$. However, we do not yet have a bound on the makespan of the schedule constructed, so we apply Lemma 5.5, which multiplies the completion times by another factor of $(1 + \frac{1}{\alpha-1})$, while bounding the makespan by $\alpha n^{2\ell+1}$. Thus the overall performance guarantee on the average completion time is thus $(4 + \epsilon) \times (1 + \frac{2}{n^3}) \times (1 + \frac{1}{\alpha-1})^2$ times optimal, which, by appropriate choice of $\alpha$ and $\epsilon$ can be made less than $(4 + \delta)$. Note that the running time will be polynomial in $\frac{1}{\delta}$. ∎

We now construct two schedules $S^{\text{o}}$ and $S^{\text{e}}$. In $S^{\text{o}}$ we consecutively schedule $S^1, S^3, S^5, \ldots$, the schedules constructed for $\mathcal{J}^1, \mathcal{J}^3, \ldots$, and in $S^{\text{e}}$ we consecutively schedule $S^2, S^4, S^6, \ldots$. The following lemma applies to both $S^{\text{o}}$and $S^{\text{e}}$; for the sake of clarity it is stated only in terms of $S^{\text{o}}$. The schedule we describe has $\alpha n^{2i+1}$ time dedicated to each $S^i$ even if $S^i$ has no jobs; however, only $n$ of the $S^i$ are nonempty.

**Lemma 5.8** *Let $J_j \in \mathcal{J}^\ell$ be a job scheduled in $S^O$. $J_j$'s completion time in $S^O$ is at most a factor of $(1+o(1))$ larger than its completion time in the schedule constructed by the algorithm of Lemma 5.7 for $\mathcal{J}^\ell$.*

**Proof:** The subschedule for any set $\mathcal{J}^\ell$ scheduled in $S^{\mathrm{O}}$ begins after $\mathcal{J}^1, \mathcal{J}^3, \ldots, \mathcal{J}^{\ell-2}$ are all scheduled . By Lemma 5.7, the makespan of $\mathcal{J}^\ell$ is bounded by $\alpha n^{2\ell+1}$, and thus $S^\ell$ starts by time $\alpha(n^3+n^5+\ldots+n^{2\ell-3}) \leq 2\alpha n^{2\ell-3}$. Since $n^{2\ell-2}$ is a lower bound on the completion time of any job in $\mathcal{J}^\ell$, increasing the completion time of a job by an additive term of $O(n^{2\ell-3})$, does not increase its completion time by more than a multiplicative factor of $(1+o(1))$. Thus, in the combined schedule $S^{\mathrm{O}}$ each job completes within a factor of $(1+o(1))$ of its completion time in $S^\ell$. ∎

Note that Lemma 5.8 would not be true for the schedule in which we schedule consecutively $S^1, S^2, S^3, \ldots$.

We now combine $S^{\mathrm{O}}$ and $S^{\mathrm{e}}$ by superimposing them over the same time slots. This creates an infeasible schedule in which the sum of completion times is just the sum of the completion times in $S^{\mathrm{O}}$ and $S^{\mathrm{e}}$, but in which there may be two jobs scheduled simultaneously. However, each job is only scheduled once in this combined schedule, so if we just replace each unit of time by two units of time and share the unit between the two jobs, we see that we can reschedule to a valid schedule while at most doubling the completion time of any job. We thus obtain the following theorem.

**Theorem 5.9** *There exists a polynomial-time $(8+\epsilon)$-approximation algorithm for $R|r_j, pmtn|\sum w_jC_j$.*

**Corollary 5.10**

- *There exists a polynomial-time $(16+\epsilon)$-approximation algorithm for $1|r_j|\sum w_jC_j$.*
- *There exists a polynomial-time $(24+\epsilon)$-approximation algorithm for $P|r_j|\sum w_jC_j$.*

**Proof:** We apply algorithm CONVERT from Section 2 to the preemptive schedule of Theorem 5.9 and invoke Theorems 2.1 and 2.4.

## Acknowledgements

We thank the anonymous referees for helpful comments.

## References

[1] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

[2] J.L. Bruno, E.G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, 1974.

[3] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming*, number 1099 in Lecture Notes in Computer Science. Springer, Berlin, 1996. Proceedings of the 23rd International Colloquium (ICALP'96).

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[5] J. Du, J.Y.T. Leung, and G.H Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75, 1990.

[6] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23:665–679, 1976.

[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[8] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, August 1997.

[9] D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[10] J.A. Hoogeveen and A.P.A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization*, pages 404–414, 1996. Published as Lecture Notes in Computer Science 1084, Springer-Verlag.

[11] W. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:846–847, 1973.

[12] T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15:1119–1129, 1986.

[13] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 418–426, May 1996.

[14] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W.R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[15] T. C. Lai, Personal communication. May, 1995.

[16] 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.

[17] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[18] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[19] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 422–431, January 1993.

[20] C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM Journal on Discrete Mathematics*, 10(4):573–598, 1997.

[21] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, June 1994.

[22] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[23] L. Stougie, 1995. Personal communication cited in [10].