Scheduling Jobs That Arrive Over Time

(Extended Abstract)

Cynthia Phillips* Clifford Stein† Joel Wein ‡

April 6, 1995

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.

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 most of these models polynomial-time algorithms were known to minimize average completion time when all the jobs were available at time 0; 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 algorithms to convert from

*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.

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 two approximation. For unrelated machines, we give a new integer programming formulation. We then solve the linear programming relaxation and give methods to round the fractional solution to valid preemptive schedules; it is possible that these methods will be of use in other settings.

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 can not 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 [8]. 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 [8]. Throughout this paper, unless specified otherwise, it is assumed that all jobs have release dates,

We will give algorithms for both preemptive and nonpreemptive scheduling. 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.

We will use the notation C_j to denote the completion time of job J_j in some schedule, and will often add a superscript to specify a particular schedule. 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 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, 3, 5, 6, 7, 9, 10, 11, 13].

We distinguish between *off-line* and *on-line* algorithms. In an off-line algorithm, all the input data associated with jobs (r_j and p_j) is known in advance, and the scheduler wishes to compute an optimal schedule. In an *on-line* algorithm, at time t, the scheduler only knows about jobs with release dates $r_j \leq t$, and must decide what job (if any) to be scheduling at that time, *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 ρ-approximation algorithm to be one which, in polynomial time, produces a schedule whose value is guaranteed to be at most ρ times the minimum possible value.

New Results: The results of this paper are as follows. We first focus on the problem of *nonpreemptively* scheduling jobs on one machine so as to minimize their average completion time. This problem is $\mathcal{NP}$-hard [9]; we give a simple 2-approximation algorithm for it, which works by transforming the optimum 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 [12]. Our proof demonstrates that for any set of jobs with release dates, the minimum average completion time of a preemptive one-machine schedule for these jobs is at most a factor of 2 smaller than the corresponding minimum average completion time of the nonpreemptive one-machine schedule.

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, and give a technique that converts preemptive

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]	$16+\epsilon$
Identical	6	$24+\epsilon$	2	$16+\epsilon$
unrelated	$O(\log^2 n)$[12]	$O(\log^2 n)$[12]	$24+\epsilon$	$32+\epsilon$

Figure 1: Summary of results. Unreferenced results are new results found in this paper, and $\epsilon = o(1)$.

parallel machine schedules to nonpreemptive schedules while at most tripling the completion time of any job. This technique applies even when the preemptive schedule is fractional (fractional parts of different jobs are assigned simultaneously to one machine) which proves useful in rounding linear program solutions to obtain valid schedules. This technique, however, does not by itself yield nonpreemptive approximation algorithms since the preemptive problems are $\mathcal{NP}$-hard. As a result, we next consider the preemptive scheduling of parallel machines.

When the release date of each job is 0 and the machines are identical, McNaughton [10] showed that 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, 5] solves it directly. When release dates are introduced, however, even the two machine problem becomes $\mathcal{NP}$-hard [3]. When one attempts, in addition, to minimize the average weighted completion time even the one machine version of the problem becomes $\mathcal{NP}$-hard [7]. 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 the problem of scheduling preemptive identical machines. These techniques, however, do not apply to weighted completion times or to unrelated machines. 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 rounding idea may be useful in other contexts as well. This formulation gives 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 the preemptive scheduling of unrelated machines so as to minimize average completion time, when all jobs are available at time 0. It is not known if this problem is $\mathcal{NP}$-hard; in [8] it states, "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 work we know of that studies this problem is [12] 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 sort of generalized matching problem. The type of generalization of matching, the rounding techniques, and the quality of approximation achieved, however, are quite different.

Whereas our algorithms in Sections 2 and 3 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 $C_j - r_j$. That optimality criterion is appropriate when trying to provide fair service to jobs, whereas the criterion we study is more appropriate in a situation where one does not care about fairness but rather getting as many jobs done as fast as possible.

2 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 discuss the consequences for the relationship between preemptive and nonpreemptive schedules. The algorithm we give is an on-line scheduling algorithm, so we prove a lower bound on the performance of *any* on-line scheduling algorithm for this problem.

Theorem 2.1 *There is a polynomial-time 2-approximation algorithm for scheduling jobs with release dates on one machine so as to minimize the average completion time of the jobs.*

Proof: It is well-known that the preemptive version of this problem is solvable in polynomial time by the shortest processing time rule: always be processing the job with the shortest remaining processing time [1]. Our algorithm for the nonpreemptive case transforms the preemptive schedule as follows.

Algorithm ONE-MACHINE:

1. Compute the optimum preemptive schedule.
2. Schedule the jobs nonpreemptively in the order of their completion time in the preemptive schedule, subject to the constraint that no job starts before its release date.

The second step of the algorithm can be visualized as follows. Consider the last scheduled piece of J_j, which is scheduled from time $C_j^p - k$ to C_j^p in the preemptive schedule. Insert $p_j - k$ 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 ONE-MACHINE. See Figure 2 for an example.

Let C_j^p be the completion time of J_j in the preemptive schedule. Then C_j^N, the completion time in the nonpreemptive schedule, is at most

$$C_j^p + \sum_{k:C_k^p \le C_j^p} p_k,$$

where the second term follows from the fact that job J_j can only be moved back by processing times associated with jobs that finish earlier in the schedule. However, since all this processing did occur before time C_j^p this sum is at most C_j^p and hence $C_j^N \le 2C_j^p$. ■

Theorem 2.2 *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 $C_N \le 2C_p$. Furthermore, there exist instances for which $C_N/C_p = \frac{4}{3} - \epsilon$, for any $\epsilon > 0$.*

Proof: The first claim of the theorem is an immediate consequence of the proof of Theorem 2.1. To prove the lower bound, 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 optimum preemptive schedule the total completion time is $(\frac{x}{2} + 1) + (x + 1) = \frac{3}{2}x + 2$. In the optimum nonpreemptive schedule the total completion time is $2x + 2$, yielding a ratio arbitrarily close to $\frac{4}{3}$. ■

This theorem generalizes to the optimality criterion of average *weighted* completion time.

Corollary 2.3 *Given a preemptive one-machine schedule of average weighted completion time W, there exists a linear-time algorithm to find a nonpreemptive schedule for the same instance of average weighted completion time $2W$.*

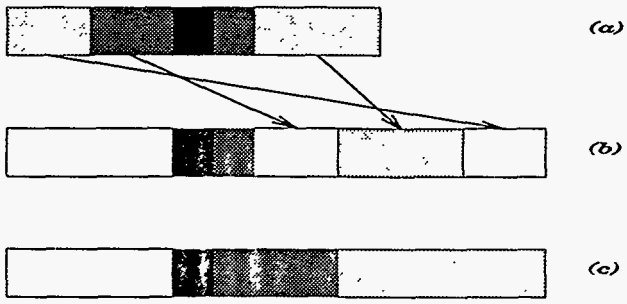

Figure 2: Illustration of the one machine algorithm. Schedule (a) is the optimal preemptive schedule. Schedule (b) has all but the last piece of each job removed from schedule (a), and empty space inserted in the schedule to make room for the pieces to be collected at the completion point of the job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

Proof: This is a direct consequence of Theorem 2.2 and the fact that the average weighted completion time is a linear function of the job completion times; doubling each completion time at most doubles the average weighted completion time as well. ■

Our analysis of the performance guarantee of algorithm ONE-MACHINE is tight, as is indicated by the following instance. At time 0 a job of size B is released, at time $B-2$ a job of size 1 is released, and at time B, x jobs of size 1 are released. Both the optimum preemptive schedule for this instance has optimum total completion time of $B(x+2)-1+\frac{(x+1)(x+2)}{2}$. The optimal nonpreemptive schedule, which has average completion time 1 greater, can be obtained by completing the job of size B first and then completing all of the size-1 jobs. Algorithm ONE-MACHINE, however, yields total completion time $3B+2Bx-2+\frac{x(x-1)}{2}$. Taking x sufficiently large and letting B go to infinity causes the ratio of these two quantities to go to 2.

2.1 On-Line Scheduling

Algorithm ONE-MACHINE can be easily modified to be an on-line algorithm without affecting its performance. 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 2.4 *There is an* on-line *nonpreemptive 2-approximation algorithm for scheduling jobs with release dates on one machine so as to minimize the average completion time of the jobs.*

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

Theorem 2.5 *No on-line nonpreemptive ρ-approximation algorithm exists for scheduling jobs with release dates on one machine so as to minimize the average completion time of the jobs with $\rho < \frac{3}{2}$.*

Proof: Assume that an on-line algorithm A has a performance guarantee of better than $1+c$ for some c. 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 is $(1+c)$. Thus we can assume that A must start J_0 before time cx.

Now consider the case when in addition to J_0, x additional 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 before 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 size-one jobs 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 $\frac{3}{2c+1}$. Choosing $c = \frac{1}{2}$, yields the theorem. ■

3 A Conversion Algorithm for Identical Parallel Machines

We now turn to the problem of scheduling jobs with release dates on parallel machines, so as to optimize 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 but the scheduling of even two identical machines is $\mathcal{NP}$-hard [3].

In this section we give a technique to convert from preemptive identical parallel machine schedules to nonpreemptive schedules. In the next two sections we will give a combinatorial algorithm for preemptive identical machines and then a linear-programming formulation that yields a constant-approximation algorithm for average *weighted* completion time on preemptive identical machines. Combining with the techniques of this section, we obtain the first constant approximation algorithms for the corresponding nonpreemptive problems.

Theorem 3.1 *Given a preemptive schedule S for a set of n jobs with release dates on parallel identical machines, construct a nonpreemptive schedule N for the same jobs by ordering the jobs in order of their completion times in the nonpreemptive schedule and then scheduling the jobs nonpreemptively in this order, respecting any release date constraints. Let C_j^N be the completion time of J_j in N. Then*

$$C_j^N \leq 3C_j^S.$$

Proof: Consider J_j which in S was released at r_j and completed at C_j^S. J_j is not started in N until $J_1, J_2, \ldots, J_{j-1}$ are started. Let N_{j-1} be the schedule of $J_1, J_2, \ldots, J_{j-1}$ in N.

Let r_k be the latest release date of any job in $J_1, \ldots, J_j$, for $k \leq j$. We know that $r_k + p_k \leq C_j^N$, since $C_k^S \leq C_j^S$.

By time r_k, jobs $J_1, \ldots, J_{j-1}$ have all been released. Thus, even if no processing happens before time r_k, at least one machine will have no more processing to do on any one of $J_1, \ldots, J_{j-1}$ by time $(\sum_{i=1}^{j-1} p_i)/m$, and so J_j starts in the nonpreemptive schedule by time

$$r_k + (\sum_{i=1}^{j-1} p_i)/m \leq r_k + C_j$$

This last inequality is true because $J_1, \ldots, J_j$ all completed by time C_j^S in S. Therefore, J_j completes by time $r_k + C_j + p_j \leq 3C_j$. ■

Corollary 3.2 *Given n jobs with release dates, let C_p be the minimum possible average weighted completion time of those jobs scheduled preemptively on m identical machines and let C_N be the minimum average weighted completion time of a nonpreemptive schedule. Then $C_N \leq 3C_p$.*

Theorem 3.1 is essentially tight, since we have an example in which one job finishes close to three times later in the nonpreemptive schedule. The corollary, however is not known by us to be tight; it is quite likely that a cumulative argument can improve the bound.

3.1 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 $f_j^i(t)$ be the fractional amount of machine M_i assigned to J_j at time t, with time taken as continuous, and let t_f be the completion time of a schedule. A fractional schedule satisfies $\sum_j f_j^i(t) \leq 1$ for all machines i and times t; $\sum_i \int_{t=0}^{t_f} s_{ij} f_j^i(t) = p_j$ for all jobs j, and for any pair of t, j only one of the $f_j^i(t)$ is non-zero. The following corollary will be useful in rounding fractional linear program solutions to valid schedules.

Corollary 3.3 *Given a fractional preemptive schedule for a [one machine/parallel identical machine] problem of average weighted completion time W, there exists a nonpreemptive schedule for the problem of average weighted completion time [$2W$/$3W$].*

Proof: Immediate from proofs of Theorems 2.1 and 3.1.

4 Preemptive Scheduling on Identical Machines

In this section we give a 2-approximation algorithm for minimizing the average completion time of a set of jobs with release dates run with preemption on a set of m identical parallel machines. For one machine, the shortest-processing-time (SPT) algorithm (always run the job that has the least amount of work remaining) is optimal. For more than one machine, there is a generalization of this approach: always run the m jobs with the shortest remaining amount of work. However, even for the case of 2 machines, this approach does not yield the optimal preemptive schedule. Intuitively, SPT finishes jobs quickly, but it can increase the schedule length, which forces other jobs to finish later than they would in the optimal schedule.

We will show that a variation on SPT is a 2-approximation for this problem. Fix an optimal schedule OPT. Let $f_{\text{OPT}}(t)$ denote the number of jobs finished in this optimal schedule by time t. We describe a BIN algorithm that computes a *pseudoschedule*, an assignment of sets of jobs to a block of time without explicit scheduling of times and machines for individual jobs. Let $f_{\text{BIN}}(t)$ be the number of jobs finished by the pseudoschedule at time t. This pseudoschedule will have the property that $f_{\text{BIN}}(t) \geq f_{\text{OPT}}(t)$ for $1 \leq t \leq T$, where T is the length of the optimal schedule. Thus if the xth job finishes at time t in the BIN algorithm, we have that t is a lower bound on the finish time of the xth job in the optimal schedule. We will then show how to convert this pseudoschedule BIN into an actual schedule RSPT (*revised shortest processing time*) such that $f_{\text{RSPT}}(2t) \geq f_{\text{BIN}}(t)$, for $1 \leq t \leq T$. That is, if the xth job finishes at time t in the BIN algorithm, then the xth job to finish in the RSPT schedule (not necessarily the same job) will finish by time $2t$. Thus we can

conclude that the average completion time of the RSPT schedule is at most 2 times the average completion time of the optimal schedule for the preemptive identical parallel machine problem.

We now describe how to construct the pseudoschedule BIN that will yield an upper bound on the number of jobs that can be completed by time t in *any* schedule for the preemptive identical parallel machine problem. Let $r_1, r_2, \ldots, r_n$ be the release dates of the input jobs in order. We can assume that $r_1 = 0$. We construct the pseudoschedule as follows: We construct n "bins" B_j of size $I_j = r_{j+1} - r_j$ for $j = 1, \ldots, n-1$ and the final bin B_n of size $I_n = T_B - r_n$, where $T_B = \lceil \sum_{i=1}^{n} p_i/m \rceil + \max_i p_i$ (note that some of the bins may be of size 0 and can be ignored). We consider the following greedy procedure. Sort all the jobs released at time 0 by processing time. Place the first mI_1 units of work from this list into bin B_1 with the restriction that no job can have more than I_1 units of work in the bin (but making no attempt to pack things onto m processors). Decrease all processing times of jobs in B_1 by the amount of processing they contribute to B_1. If this sets a processing time to zero, we say the job finishes in B_1. Now we repeat the process for B_2, a bin of size $r_3 - r_2$, considering all the jobs released at or before time r_2, that still have positive processing times. Continue placing jobs in this manner until the last bin which has size $m(T_B - r_n)$.

Now now define $f_{\text{BIN}}(t)$, the number of jobs finished by the BIN pseudoschedule for any time $0 \leq t \leq T_B$. Suppose $r_i \leq t < r_{i+1}$ (assume a dummy $r_{n+1} \equiv \infty$). The number of jobs finished by the BIN procedure, $f_{\text{BIN}}(t)$, is the number of jobs completely placed in the first $i-1$ bins, plus the number of jobs that would be completely placed in a modified ith bin B_i' of size $I_i' = t - r_i$. In other words, we consider what the procedure BIN would pack if forced to stop at time t. This new ith bin can only accept I_i' units from each job and a total of mI_i' work.

Lemma 4.1 $f_{\text{BIN}}(t) \geq f_{\text{OPT}}(t)$.

Proof: **(Sketch)** Suppose there is some schedule S that completes more jobs by time t than BIN. Let J be the set of jobs not finished in BIN that finished by schedule S. If we wish to put the set J of jobs into the BIN schedule, we must remove an equal amount of work that is already "scheduled". This entering set will complete fewer jobs per unit of work (since the individual jobs will be larger than the ones they are replacing) and therefore will force the removal of at least as many jobs are entering, violating the assumption that schedule S completed more jobs than the greedy strategy. ∎

Our revised shortest-processing-time (RSPT) algorithm schedules the work assigned to bin B_j from time $2r_j$ to time $2r_{j+1}$. The RSPT algorithm takes all jobs placed in bin B_j by the BIN pseudoschedule, sorts the pieces by processing time and then list schedules the job pieces on m processors. Since we are list processing the jobs (or job pieces) within each bin, these jobs will clearly all finish by time $2I_j$. However, we can claim something slightly stronger.

Lemma 4.2 $f_{\text{RSPT}}(2t) \geq f_{\text{BIN}}(t)$.

Proof: Let J be the set of jobs completed at time $r_i \leq t < r_{i+1}$ by the BIN pseudoschedule described above. We show that the RSPT algorithm finishes the set J of jobs by time $2t$. Let $j_1, j_2, \ldots, j_x$ be the jobs to be list scheduled into bin B_j, for $j \leq i$. Starting at time $2r_j$, processor P_i will process jobs $j_i, j_{m+i}, j_{2m+i}, \ldots, j_{km+i}$, in order, possibly not completing job j_{km+i}. At time $2r_j + I_j = r_{j+1} + r_j$, all the jobs placed in bin B_j by the pseudoschedule are done by RSPT or are being worked on by RSPT. To see this, notice that there are no preemptions between time $2r_j$ and $r_j + r_{j+1}$ because there are no new releases considered (we are scheduling a block of time that is between releases in the BIN pseudoschedule). At time $2r_j$ the processors $P_1, \ldots P_m$ will start on the smallest m jobs $j_1, \ldots, j_m$ in order. Processor P_1 will become available no later than any other processor and will start job j_{m+1}. In fact, the jobs will be tiled across the processors in order since, for example, $\sum_{i=0}^{j} p_{im+1} > \sum_{i=1}^{j} p_{im}$ (because $p_{im+1} \geq p_{im}$ and the left side also has

p_1). This shows that the first processor (which is always getting the shortest first job, the shortest second, job, etc) can never be more than one job ahead of the last processor (which gets the largest first job, the largest second job, etc). There are at most m jobs that are placed in B_j by the BIN strategy but are not completed by RSPT by time $2r_j + I_j$, the longest (at most) m jobs placed in B_j, each of length at most I_j. We now let these m jobs finish (to the extent that BIN finished them), which happens by time $2r_{j+1}$. This argument shows that by time $2r_i$, RSPT has finished all the work that BIN finishes by time r_i. We now argue that over the next $2(t - r_i)$ units, RSPT finishes all the work that would be packed by BIN into a modified bin B_i' of size $t - r_i$. RSPT list schedules jobs from the original bin B_i from time $2r_i$ to $2r_{i+1}$. However, by the above argument, at time $2r_i + I_i'$, RSPT has done all the work that BIN would assign to bin B_i' except perhaps the jobs it is currently processing. No jobs are preempted during the list schedule, so the current jobs will continue running (or finish) in the next $t - r_i$ time units just as though there had been an actual bin boundary at time t. ■

Theorem 4.3 *There is an $O(n \lg n)$-time online 2-approximation algorithm for preemptively scheduling n jobs with release dates on parallel identical machines.*

Proof: **(Sketch)** The 2- approximation follows from the previous two lemmas. The running time is dominated by sorting and the algorithm can trivially be modified to be on-line so that the schedule at time t depends only upon the jobs that have been released at times $t' \leq t$.

■

5 Preemptive Schedules 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 utilize this integer program in two ways. First, the solutions of its linear-programming relaxation can be immediately converted, using the techniques of Section 3 , to nonpreemptive schedules for identical machines. Second, we give a method of rounding the fractional solutions generated by the linear program to valid preemptive schedules, and obtain the first constant-factor approximation algorithms for a very general form of the preemptive scheduling problem, namely average weighted completion time on unrelated parallel machines. Unless otherwise stated, all results in this section apply to this most general case.

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 size 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 kth 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 of 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 size 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 sizes are polynomial in n and m this formulation will not be polynomial in the input size. However, in Section 5.3 we will show how to scale the input data to be of size polynomial in n and m, and how to interpret the scaled solution as a solution for the original instance with little degradation in the quality of approximation.

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. Its solution, however, gives a lower bound on *nonpreemptive* schedules. In addition we will show that its solutions are at most a constant factor worse than those of the optimal preemptive schedule, and therefore do yield preemptive approximations of high quality.

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 Polynomial Size Jobs

Throughout this section we assume that no preemption is allowed within single units of a job, $m \leq n$, and $p_j \leq n^4$. In Sections 5.2 and 5.3 we show how to remove these assumptions. Note that although preempting an individual unit is not allowed, in the unrelated machine model, due to the speeds of the machines, units may take fractional amounts of time to complete.

We introduce the following $\{0,1\}$-integer program $\mathcal{IP}$. Let $T = \{t | \exists j \text{s.t.} r_j \leq t \leq t + np_{\max}\}$, denote a set of times that includes all the times that any job will ever be running. Note that by the assumptions of this section, T is of polynomial size. We will use the variable $x_{ijkt}, 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq p_j, t \in T$. $x_{ijkt} = 1$ will represent the scheduling of unit k of job J_j on machine m_i at time t. Note that for convenience of notation, the range of k depends on j. Recall that s_{ij} is the speed of job J_j on machine M_i.

$\mathcal{IP}$ will be made up of the following constraints.

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

$$\sum_{i,t} x_{ijkt} = 1 \qquad j = 1,\ldots,n;\ k = 1,\ldots,p_j \tag{2}$$

$$\sum_{i,k} \frac{x_{ijkt}}{s_{ij}} \leq 1 \qquad j = 1,\ldots,n;\ t \in T \tag{3}$$

$$\sum_{j,k} \frac{x_{ijkt}}{s_{ij}} \leq 1 \qquad i = 1,\ldots,m;\ t \in T. \tag{4}$$

In addition, for all k, j such that k is the last unit of J_j, and for all t

$$\sum_{i,t'<t,\ell<k} x_{ij\ell t'} \geq (k-1)x_{ijkt} \tag{5}$$

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

For all k, j such that k is the last unit of J_j, the cost associated with x_{ijkt} is $w_j t$; otherwise the cost is 0. The goal is to minimize the total cost subject to the above constraints.

Lemmas 5.1 and 5.2 characterize the relationship between $\mathcal{IP}$ and optimal solutions to variants of our scheduling problems.

Lemma 5.1 *Given an instance of a single machine or parallel identical machines scheduling problem, and assuming preemption of a single unit 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: Clearly any preemptive schedule is a feasible solution to $\mathcal{IP}$. Given a feasible solution to $\mathcal{IP}$, equation (2) guarantees that every unit of every job is scheduled; inequality (3) guarantees that the total amount of time taken up by all any pieces of any units of a job during one time unit is no more than one. Inequality (4) guarantees that the total time of all the work assigned to any machine for one unit of time is at most 1. Equation (5) guarantees that when the last unit of a

job is scheduled, the one that contributes to the weighted completion time, the rest of the job has been scheduled already. Note that as long as (3) and (4) are satisfied, the order in which the first $p_j - 1$ units of J_j are scheduled is not important. ■

For unrelated machines the relationship between $\mathcal{IP}$ and optimal preemptive schedules is more complex for two reasons. First, a unit may run for a fractional amount of time, and in an optimal schedule a job might finish at, say, time 2.5; whereas our formulation can only capture finishes at integral times. Secondly, a preemptive schedule that only allows preemption within units can still have situations in which a job, for example, starts at time .3 and ends at time 1.2. This overlap between units of time is not captured by $\mathcal{IP}$ either.

However, these two problems can be remedied. First, note that if J_j could be scheduled to complete at time t, e.g. 2.5, in $\mathcal{IP}$ its last piece can be scheduled into the $t = 3$ unit of time, and be "declared" to finish at time 3, which will make a contribution to the objective function of 3, or in general at most $t+1$. Secondly, a schedule with "overlap" can be captured by $\mathcal{IP}$ by associating the entire overlapping piece with the time unit in which it ends. This means relaxing (4) to ≤ 2 instead of 1; call this modified program $\mathcal{IP}^2$. It is not hard to see that given a solution with at most 2 units of work/machine in any time unit, it can be converted to a solution with at most one, while at most doubling the completion time of the last piece of any job.

Lemma 5.2 *If there is a schedule for an unrelated machine problem in which J_j, $j = 1, \ldots, n$ finishes at time t_j, then there is a solution to $\mathcal{IP}^2$ in which the last unit of J_j finishes at time $2t + 1$.*

For the rest of this section we'll focus on $\mathcal{IP}$; our results apply, paying a factor of $2 + \frac{1}{t}$ in the completion time of each job, to $\mathcal{IP}^2$ as well. To obtain schedules of good approximate quality from $\mathcal{IP}$ we relax $x_{ijkt} \in \{0, 1\}$ to $0 \leq x_{ijkt} \leq 1$ and solve the resulting linear program; call this linear program $\mathcal{LP}$. In the resulting 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 weighted completion time.*

The solution to the linear program can not immediately be interpreted as a schedule for two reasons. First, a particular unit may be fractionally assigned across several machines during the same time unit; we need to find a way to schedule these fractional pieces of the job so that they do not run simultaneously. Secondly, the fractional weighted 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 finish processing. In fact, it is possible that the weighted completion time of J_j is much larger than the *fractional* weighted completion time.

We now show how to deal with these two concerns and produce a valid schedule. The first we can get rid of via an application of open shop theory [4, 8]; the latter problem we cope with by introducing a new rounding idea that may have applications in other contexts.

Lemma 5.3 *Consider one unit of time t' in an assignment of fractional pieces of jobs to units of time. There exists a valid schedule for that unit of time, in which each machine makes at most m^4 preemptions.*

Proof: Let $p'_{i'j'} = \sum_{k,t} x_{i'j'kt'}/s_{i'j'}$ denote the amount of processing time that job J_j performs on machine M_i. Create an operation $o'_{i'j'}$ of length $p'_{i'j'}$. This defines an *open shop* scheduling problem[4, 8]. 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. Using a result of [4], we know that we can create a preemptive schedule with at most m^4 preemptions per processor , with length

$\max\{\max_{j'} \sum_{i'} p_{i'j'}, \max_{i'} \sum_{j'} p_{i'j'}\}$, which by constraints (Y) and (Z) is at most 1. Thus the jobs can be scheduled in the time unit they are assigned. ■

Applying Lemma 5.3 to each time unit of the solution to $\mathcal{LP}$, we obtain:

Lemma 5.4 *Given a solution to $\mathcal{LP}$ in which the last fractional piece of the last unit of J_j, $j = 1, \ldots, n$ is assigned to time t_j, there exists a valid schedule in which each J_j finishes by time t_j.*

Lemma 5.5 *There exists a polynomial-time algorithm, which, given a solution to $\mathcal{LP}$, converted to a valid schedule S using Lemma 5.4, produces a schedule with average completion time at most 4 times the weighted fractional completion time of S.*

Proof: Consider a job J_j of size $p_j = k$ and consider the kth unit of J_j. This unit may be assigned fractionally on a number of machines at different times. Let time t^* be the unit of time at the end of which, for the first time, 1/2 of this kth unit is assigned.

By constraint 5 we know that by time $t^* - 1$ at least half of the first $k-1$ units of J_j have been processed. We double the time scale of the entire schedule, remove all pieces of the first $k-1$ units of J_j that were scheduled in S after $t^* - 1$, any pieces of the kth unit that were scheduled after time t^*, and schedule these pieces in the "doubles" of the pieces of J_j that were scheduled before $t^* - 1$ or t^*, respectively. Therefore, by time $2t^*$ all of J_j has been processed. Noting that $\frac{t^*}{2}$ is a lower bound on the fractional completion time gives the lemma. ■

Theorem 5.6 *Under the assumptions that $p_{\max} \leq n^4$ and that preemption is not allowed within units of a job, there exists a polynomial-time 4-approximation algorithm for preemptive scheduling of identical machines with release dates to minimize average weighted completion time, and a $(8+\epsilon)$-approximation algorithms for the corresponding unrelated machines problem.*

Proof: Combine Lemmas 5.1, 5.2, 5.4 and 5.5.

5.2 Allowing Arbitrary Preemption

We can prove that if one restricts preemptions to happen only at time intervals which are integral multiples of $1/m^4$, the weighted average completion time can only increase by at most a factor of $(2 + o(1))$, and in many cases, such as the presence of large jobs, by a $1 + o(1)$ factor. The details are omitted in this extended abstract.

Lemma 5.7 *Let C be the weighted average completion time of the optimal schedule. A linear program of the form $\mathcal{LP}$ in which we split each job and each time slot on each machine into m^5 equal-size pieces, will have an optimal weighted average completion time of at most $(1+1/m)C + \frac{1}{n}\sum w_j$.*

We note that this is always at most $2 + o(1)$, and when $p_{\max} > n^2$ and $w_j = 1\ \forall j$ it is $1 + o(1)$.

5.3 Large Jobs

One can round and scale to take care of large jobs, the details are omitted in this extended abstract.

Lemma 5.8 *Given a ρ-approximation algorithm for the preemptive scheduling of jobs on unrelated machines so as to minimize average weighted completion time that assumes $p_{\max} \leq n^4$, there exists a $(2 + \epsilon)\rho$-approximation algorithm for arbitrary job sizes.*

Proof: Omitted.

Combining Lemma 5.8 with Lemma 5.7 and Theorem 5.6 with Lemma 5.8, Lemma 5.7 and Corollary 3.3, along with a bit of balancing of different cases yields the results detailed in Figure 1.

References

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

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

[3] J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraints. Technical report, University of Texas at Dallas, 1988. Technical Report.

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

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

[6] 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.

[7] 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.

[8] 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 pp 445–522. 1993.

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

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

[11] 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.

[12] C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. In *Proceedings of Fourth Scandinavian Workshop on Algorithm Theory*, pages 290–301, 1994.

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

DISCLAIMER