Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms

Dedicated to the memory of Gene Lawler

Leslie A. Hall[1] Andreas S. Schulz[2] David B. Shmoys[3] Joel Wein[4]

June 6, 1996

Abstract

In this paper we introduce two general techniques for the design and analysis of approximation algorithms for $\mathcal{NP}$-hard scheduling problems in which the objective is to minimize the weighted sum of the job completion times. For a variety of scheduling models, these techniques yield the first algorithms that are guaranteed to find schedules that have objective function value within a constant factor of the optimum. In the first approach, we use an optimal solution to a linear programming relaxation in order to guide a simple list-scheduling rule. Consequently, we also obtain results about the strength of the relaxation. Our second approach yields on-line algorithms for these problems: in this setting, we are scheduling jobs that continually arrive to be processed and, for each time t, we must construct the schedule until time t without any knowledge of the jobs that will arrive afterwards. Our on-line technique yields constant performance guarantees for a variety of scheduling environments, and in some cases essentially matches the performance of our off-line LP-based algorithms.

[1] `lah@jhu.edu`. Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, MD 21218. Research supported by NSF Research Initiation Award DMI-9496153.

[2] `schulz@math.tu-berlin.de`. Department of Mathematics, Technical University of Berlin, 10623 Berlin, Germany. Research partially supported by the graduate school Algorithmische Diskrete Mathematik, grant We 1265/2–1.

[3] `shmoys@cs.cornell.edu`. School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853. Research partially supported by NSF grant CCR-9307391.

[4] `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.

1 Introduction

In his seminal paper, Graham (1966) showed that when jobs are scheduled on identical parallel machines by a list-scheduling rule, then the algorithm is guaranteed to produce a schedule of length that is within a factor of two of optimal. This result is often viewed as the starting point for research on the design and analysis of approximation algorithms. A *ρ-approximation algorithm* is a polynomial-time algorithm that always finds a solution of objective function value within a factor of ρ of optimal; ρ is also referred to as the *performance guarantee* of the algorithm. In the following decades, there has been a great deal of work on approximation algorithms for $\mathcal{NP}$-hard optimization problems, and in particular, for scheduling problems with min-max objective functions. However, until recently much less was known about approximation algorithms for $\mathcal{NP}$-hard scheduling problems with min-sum objective functions.

In this paper we introduce two general techniques for the design of approximation algorithms for $\mathcal{NP}$-hard scheduling problems in which the goal is to minimize the weighted sum of the job completion times; these techniques yield the first constant performance guarantees for a variety of scheduling models. Whereas little was known about approximation algorithms for these problems, there is an extensive literature on their polyhedral structure; Queyranne & Schulz (1994) give a comprehensive survey of this area of research. For single-machine models, several linear programming relaxations have been considered, and they yield sufficiently strong lower bounds to allow instances of modest size to be solved by enumerative methods. Our first technique was motivated by this success: we show that Graham's list-scheduling algorithm, when guided by an optimal solution to a linear relaxation, is guaranteed to produce a schedule of near-optimal total weighted completion time. A consequence of these results is that the lower bound given by the linear programming relaxation is also guaranteed to be within a constant factor of the true optimum.

Our second technique is a general framework for designing on-line algorithms to minimize total weighted completion time in scheduling environments with release dates. In this setting, we are scheduling jobs that intermittently arrive to be processed and, for each time t, we must construct the schedule until time t without any knowledge of the jobs that will arrive after time t. Our on-line algorithm relies only on the existence of an (off-line) approximation algorithm for a problem that is closely related to finding a minimum-length schedule in that environment. For several of the problems we consider, the performance guarantee proved for this on-line technique asymptotically matches the guarantee proved for our off-line LP-based algorithms.

The problem of scheduling a single machine to minimize the total weighted job completion time is one of the most basic problems in the area of scheduling theory. We are given n jobs, and each job j has a specified positive weight w_j and a nonnegative processing time p_j, $j = 1, \ldots, n$. The jobs must be processed without interruption, and the machine can process at most one job at a time. We let C_j denote the completion time of job j and the goal is to minimize $\sum_j w_j C_j$, or equivalently, $(\sum_j w_j C_j)/n$. Consider some optimal schedule, and let C_j^* denote the completion time of job j in it; thus, $\sum_j w_j C_j^*$ denotes the optimal value. We shall present a number of approximation algorithms that are based upon solving a particular relaxation; throughout the paper, we shall use the notation $\overline{C}_j$ to denote the value assigned to job j for the relaxation, and so $\sum_j w_j \overline{C}_j$ is a lower bound on $\sum_j w_j C_j^*$. Furthermore, for each approximation algorithm that we shall consider, we use $\widetilde{C}_j$ to denote the completion time of job j in the schedule that it computes.

For the single-machine problem stated above, Smith (1956) showed that sequencing in order of non-decreasing p_j/w_j ratio produces an optimal schedule. We shall be interested in more constrained, strongly $\mathcal{NP}$-hard problems, in which each job j cannot begin processing before a specified *release date* r_j, $j = 1, \ldots, n$, or there is a partial order $\prec$ on the jobs, where $j \prec k$ is a *precedence constraint* that requires job k to begin processing no earlier than the completion time of job j. We give a 2-approximation algorithm for the case in which there are precedence constraints, but no (non-trivial) release dates. In contrast, Ravi, Agrawal, & Klein (1991) gave an $O(\log n \log \sum_j w_j)$-approximation algorithm, and Even, Naor, Rao, & Schieber (1995) recently improved this to $O(\log n \log \log \sum_j w_j)$. For the case in which there are also release dates, we give a 3-approximation algorithm. In fact, with only slightly larger constants, these results extend to the model with m identical parallel machines, in which each job j must be processed without interruption for p_j time units on some machine. Furthermore, these results extend to models in which *preemption* is allowed, that is, the processing of a job may be interrupted and resumed at a later time, possibly on a different machine. Even for the special case of minimizing $\sum_j C_j$, these algorithms are the first shown to have sublogarithmic performance guarantees.

Our results were motivated by recent work using polyhedral methods for scheduling problems, and in particular, single-machine scheduling problems. There are a number of interesting papers in this area, both for characterizations of polynomially-solvable special cases and for computing optimal solutions; these include work of Balas (1985), Wolsey (1985,1990), Dyer & Wolsey (1990), Queyranne (1993), Queyranne & Wang (1991a,b), Lasserre & Queyranne (1992), Sousa & Wolsey (1992), von Arnim & Schulz (1994), Crama & Spieksma (1995), Van den Akker, Van Hoesel, & Savelsbergh (1993), Van den Akker (1994), Van den Akker, Hurkens, & Savelsbergh (1995), and von Arnim, Schrader, & Wang (1996).

Several of our algorithms are based on the work of Wolsey (1985) and Queyranne (1993), who proposed a linear programming relaxation in which the decision variable C_j, $j = 1, \ldots, n$, corresponds to the completion time of job j in a schedule. For the unconstrained single-machine scheduling problem solved by Smith (1956), Queyranne (1993) showed that this formulation provides an exact characterization. He also gave a polynomial-time separation algorithm, and so the relaxation can be solved in polynomial time, even if additional constraints are added to enforce release dates or precedence constraints. For these more constrained variants, we will show that an optimal solution to the linear programming formulation can be used to derive a schedule that is within a constant factor of the LP optimum. If a linear programming relaxation is shown to have an optimal value that is always within a factor of ρ of the true optimum, we shall call it a *ρ-relaxation* of the problem. For example, for the problem of minimizing total weighted completion time on a single machine subject to precedence constraints, we show that Queyranne's formulation is a 2-relaxation.

Our algorithm and its analysis are also inspired by recent work of Phillips, Stein, & Wein (1995) for the case in which there are release dates, but no precedence constraints. They introduced the notion of constructing a near-optimal nonpreemptive schedule by scheduling the jobs in order of their completion times in a preemptive schedule; this idea led to a simple 2-approximation algorithm to minimize (nonpreemptively) the average completion time of a set of jobs with release dates on one machine (i.e., in the special case where $w_j = 1$, $j = 1, \ldots, n$). They also introduced a time-indexed linear programming formulation from which they constructed near-optimal preemptive schedules for a variety of models in which the objective is minimize the average weighted completion time. Based on these ideas, they gave approximation algorithms for four models with this objective: scheduling preemptively or nonpreemptively on one machine or m identical parallel machines: let ϵ be an arbitrarily small positive constant; for both preemptive models their performance guarantee is $8 + \epsilon$; for one machine, their nonpreemptive guarantee is $16 + \epsilon$; and for m identical parallel machines their guarantee is $24 + \epsilon$. For all four scheduling models, our techniques significantly improve upon these performance guarantees.

Our results also have implications for other well-studied formulations of these single-machine scheduling problems. For example, since the formulation in completion-time variables is weaker than both a linear-ordering formulation of Potts (1980) and a time-indexed formulation of Dyer &

Wolsey (1990), we see that each of these is also a 2-relaxation in the case mentioned above. Van den Akker (1994) evaluated the effectiveness of several heuristics for the model in which there are release dates but no precedence constraints, and concluded that the following one is particularly effective in practice: solve the time-indexed relaxation and schedule the jobs in the order in which they complete (in an average sense) in the optimal fractional solution. Our analysis implies that this procedure is a 3-approximation algorithm, and hence it can be viewed as a theoretical validation of this approach to finding a good schedule.

We also introduce a polynomial-size variant of the time-indexed formulation, called an *interval-indexed formulation.* We show that such formulations are effective in the design of approximation algorithms for scheduling jobs, constrained by release dates, on *unrelated parallel machines.* In this scheduling environment each job j must be assigned to some machine i, and requires p_{ij} time units when processed on machine $i = 1, \dots, m$. We introduce new rounding algorithms that yield the first constant performance guarantee for this problem. These results build on earlier research on computing near-optimal solutions for other scheduling models by rounding fractional solutions to linear relaxations. This research includes work by Lenstra, Shmoys, & Tardos (1990), Lin & Vitter (1992), Trick (1994), Munier & König (1993), and, most relevant to our work, that of Shmoys & Tardos (1993).

We then turn to our second technique: a general method for devising on-line algorithms to minimize the total weighted completion time in any scheduling environment with release dates. We show that if we assign jobs to intervals by applying a type of greedy strategy, then the resulting performance guarantee is within a factor of four of the performance guarantee of the subroutine used to make the greedy selection. This technique is similar to one used by Blum, Chalasani, Coppersmith, Pulleyblank, Raghavan, & Sudan (1994) to devise an approximation algorithm for the minimum latency problem, which is the variant of the traveling salesman problem in which one wishes to minimize the sum of the travel times to reach each city, rather than the time to reach the last city. We shall use this technique to devise on-line approximation algorithms, and in several cases, the resulting algorithm has nearly as good a performance guarantee as the off-line LP-based technique.

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). We summarize the most relevant features of this notation here. Each problem that we shall consider can be abbreviated $\alpha|\beta|\gamma$, where (i) α is either 1, P, or R, denoting that there is either one machine, m identical parallel machines, or m unrelated parallel machines; (ii) β contains some subset of r_j, $prec$, $pmtn$, and $p_j = 1$, where these denote respectively the presence of (non-trivial) release date constraints, precedence constraints, the ability to schedule preemptively, and the restriction that all jobs are of unit size; and (iii) γ is $\sum w_j C_j$, indicating that we are minimizing the total weighted job completion time. For example, $1|r_j, prec|\sum w_j C_j$ refers to the problem of minimizing (nonpreemptively) the total weighted completion time on one machine subject to release-date and precedence constraints. We shall assume, without loss of generality, that the data for each instance is integral and that the data is preprocessed so that in any feasible schedule, there does not exist a job that can be completed at time 0; hence, no job can complete before time 1. Finally, note that we have assumed that no job has weight 0, primarily to ensure that the p_j/w_j ordering is well-defined; however this assumption is be made without loss of generality, since a job of weight 0 does not affect the objective function value.

The approach of applying a list-scheduling rule in which the jobs are ordered based on solving a linear program can easily be extended to a wide spectrum of scheduling problems, and we believe that it will have further consequences for the design of approximation algorithms. For several other basic scheduling models, we have considered analogous formulations, and conjecture them to yield substantially stronger guarantees than are presently known. Motivated by our work, Chudak & Shmoys (1996) have given $O(\log m)$-approximation algorithms for the $C_{\max} = \max_j C_j$ and $\sum w_j C_j$ objectives in the setting in which the parallel machines run at different speeds; this improves upon the best known performance guarantees of $O(\sqrt{m})$ due to Jaffe (1980) (for $C_{\max}$) and Schulz (1995) (for $\sum w_j C_j$). Möhring, Schäffter & Schulz (1996) considered the problem of scheduling with communication delays to minimize the average weighted completion time. Specifically, they present the first constant-factor approximation algorithms for scheduling identical parallel machines subject to release dates and small communication delays. Our on-line technique has also already inspired several different directions. Chakrabarti, Phillips, Schulz, Shmoys, Stein & Wein (1996) have given a version with an improved performance guarantee and extended the technique to a variety of other scheduling models. In addition, they show that this on-line technique finds schedules that are *simultaneously* near-optimal with respect to both the maximum completion time and the total weighted completion time objectives. Further extensions to other models are given by Chakrabarti & Muthukrishnan (1996).

2 Single-machine scheduling problems

In this section we present approximation algorithms for several single-machine scheduling problems; we consider variants in which the set of jobs may be precedence constrained, and in which additionally each job j may have a release date r_j. We use $j \prec k$ to denote the constraint that job j must be completed before job k starts. We denote the entire set of jobs $\{1, \ldots, n\}$ as N, and, for any subset $S \subseteq N$, we use the following shorthand notation:

$$p(S) = \sum_{j \in S} p_j, \quad r_{\min}(S) = \min_{j \in S} r_j, \quad \text{and} \quad r_{\max}(S) = \max_{j \in S} r_j.$$

We shall also require the quantity $\sum_{j \in S} p_j^2$, which we shall denote by $p^2(S)$.

The basis of our approximation algorithms is a linear programming relaxation that uses as variables the completion times C_j. We can formulate the problem $1|r_j, prec|\sum w_j C_j$ in the following way, where the constraints ensure that the variables $C_1, \ldots, C_n$ specify a feasible set of completion times:

$$\text{minimize} \sum_{j=1}^{n} w_j C_j \tag{1}$$

subject to

$$C_j \geq r_j + p_j, \qquad j = 1, \ldots, n, \tag{2}$$

$$C_k \geq C_j + p_k, \qquad \text{for each pair } j, k \text{ such that } j \prec k, \tag{3}$$

$$C_k \geq C_j + p_k \quad \text{or} \quad C_j \geq C_k + p_j, \qquad \text{for each pair } j, k. \tag{4}$$

The difficulty with this characterization is that the so-called "disjunctive" constraints (4) are not linear inequalities and cannot be modeled using linear inequalities. Instead, we use a class of valid inequalities, introduced by Queyranne (1993), that are motivated by considering Smith's rule for scheduling the jobs when there are no release dates or precedence constraints. Smith (1956) proved that a schedule is optimal if and only if the jobs are scheduled in order of non-decreasing ratio p_j/w_j. As a result, if we set $w_j = p_j$ for all j, then the sum $\sum_j p_j C_j$ is invariant for any ordering of the jobs. In particular, for the ordering $1, \ldots, n$, if there is no idle time in the schedule then $C_j = \sum_{k=1}^{j} p_k$; therefore, for any schedule we can write down the valid constraint

$$\sum_{j=1}^{n} p_j C_j \geq \sum_{j=1}^{n} p_j (\sum_{k=1}^{j} p_k) = \sum_{j=1}^{n} \sum_{k=1}^{j} p_k p_j = \frac{1}{2}(p^2(N) + p(N)^2),$$

where the inequality results from the possibility of idle time in the schedule.

Since a schedule for the entire set of jobs can be interpreted as a schedule for any subset, we also have the following valid inequalities:

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2}(p^2(S) + p(S)^2), \quad \text{for each } S \subseteq N. \tag{5}$$

We note that these inequalities remain valid even if we allow the schedule to be preemptive; that is, the processing of a job may be interrupted and continued at later point in time. Furthermore, Queyranne (1993) has shown that constraints (5) are sufficient to describe the convex hull of completion-time vectors of feasible schedules for instances of $1||\sum w_j C_j$. These constraints are no longer sufficient, however, if we add constraints such as (2) and (3) that enforce release dates and precedence constraints, respectively. Although we do not have exact characterizations for either $1|prec|\sum w_j C_j$ or $1|r_j, prec|\sum w_j C_j$, we will show that this linear relaxation can be used to find near-optimal solutions for each of them.

The key to the quality of approximation deriving from these relaxations is the following lemma.

Lemma 2.1 *Let $C_1, \ldots, C_n$ satisfy (5), and assume without loss of generality that $C_1 \leq \cdots \leq C_n$. Then, for each $j = 1, \ldots, n$,*

$$C_j \geq \frac{1}{2} \sum_{k=1}^{j} p_k \ .$$

Proof: Inequality (5) for $S = \{1, 2, \ldots, j\}$ implies that

$$\sum_{k=1}^{j} p_k C_k \geq \frac{1}{2}(p^2(S) + p(S)^2) \geq \frac{1}{2} p(S)^2. \tag{6}$$

Since $C_k \leq C_j$, for each $k = 1, \ldots, j$, we have

$$C_j \cdot p(S) = C_j \sum_{k=1}^{j} p_k \geq \sum_{k=1}^{j} C_k p_k \geq \frac{1}{2} p(S)^2,$$

or equivalently, $C_j \geq \sum_{k=1}^{j} p_k / 2$. ■

A feasible solution $C_1 \leq \cdots \leq C_n$ to (5) need not correspond to a feasible schedule: the intervals $(C_j - p_j, C_j]$, $j = 1, \ldots, n$, are not constrained to be disjoint. If this solution actually corresponds a feasible schedule, then $C_j \geq \sum_{k=1}^{j} p_k$, $j = 1, \ldots, n$. Lemma 2.1 states that merely satisfying the constraints (5) is sufficient to obtain a relaxation of this: $C_j \geq (1/2) \sum_{k=1}^{j} p_k$, $j = 1, \ldots, n$. It is this intuition that underlies the approximation algorithms of this section.

2.1 Single-machine scheduling with precedence constraints

We begin by presenting a 2-approximation algorithm for $1|prec|\sum w_j C_j$ based on the linear programming formulation that minimizes $\sum_{j=1}^{n} w_j C_j$ subject to constraints (3) and (5). Consider the following heuristic for producing a schedule: first, we obtain an optimal solution to the linear program, $\overline{C}_1, \ldots, \overline{C}_n$; and then we schedule the jobs in order of non-decreasing $\overline{C}_j$, where ties are broken by choosing an order that is consistent with the precedence relation. We refer to this algorithm as Schedule-by-$\overline{C}_j$, since the jobs are ordered according to their completion times in the linear programming solution. Observe that constraints (3) ensure that the resulting schedule will be consistent with the precedence constraints.

Lemma 2.2 *Let $C_1^*, \ldots, C_n^*$ denote the completion times in some optimal schedule, and $\widetilde{C}_1, \ldots, \widetilde{C}_n$ denote the completion times in the schedule found by* Schedule-by-$\overline{C}_j$. *Then $\sum_j w_j \widetilde{C}_j \leq 2 \sum_j w_j C_j^*$.*

Proof: For simplicity we assume that the jobs have been renumbered so that $\overline{C}_1 \leq \cdots \leq \overline{C}_n$; therefore, for $S = \{1, \ldots, j\}$,

$$\widetilde{C}_j = p(S).$$

By Lemma 2.1, we immediately obtain $\widetilde{C}_j \leq 2\overline{C}_j$. Since $\sum_j w_j \overline{C}_j \leq \sum_j w_j C_j^*$, the result follows. ■

Queyranne (1993) has shown that the linear program given by (1), (3) and (5) is solvable in polynomial time via the ellipsoid algorithm; the key observation is that there is a polynomial-time separation algorithm for the exponentially large class of constraints (5). Hence we have established the following theorem.

Theorem 2.3 Schedule-by-$\overline{C}_j$ *is a 2-approximation algorithm for $1|prec|\sum w_j C_j$.*

Next, we present a set of instances, suggested by Margot and Queyranne, which show that our analysis of this heuristic is tight. Consider an instance with $2k$ jobs with

$$p_j = \begin{cases} 1, & j = 1, \ldots, k; \\ 0, & j = k+1, \ldots, 2k; \end{cases}$$

$$w_j = \begin{cases} 0, & j = 1, \ldots, k-1; \\ 1, & j = k, k+1; \\ 2, & j = k+2, \ldots, 2k. \end{cases}$$

Also, $j \prec k+j$ and $j \prec k+j+1$, for $j = 1, \ldots, k-1$, and $k \prec 2k$. If $\overline{C}_j$ denotes the optimal LP "completion time," then $\overline{C}_j = \alpha$, $j = 1, \ldots, 2k$, where α is chosen so that $\sum_{j=1}^{2k} p_j \overline{C}_j = (1/2)(p^2(N) + p(N)^2)$, for $N = \{1, \ldots, 2k\}$. In the optimal schedule, the jobs are processed in the order $1, k+1, 2, k+2, \ldots, k, 2k$. Its value is $k^2 + 2k - 1$. On the other hand, one possible ordering generated by the algorithm Schedule-by-$\overline{C}_j$ is $1, \ldots, 2k$, with objective function value equal to $2k^2$. Hence, the ratio between heuristic value and the optimal value approaches 2 as $k \to \infty$. In this example, the bad behavior of the heuristic results from an unlucky breaking of ties; in fact, by perturbing the data, it is possible to force the algorithm to choose an equivalently bad solution.

One manner in which the linear program given by (3) and (5) can be strengthened is by adding a set of so-called series constraints (see Queyranne & Schulz (1994)). When these are added to the model, Queyranne & Wang (1991a) showed that this gives an exact characterization of the feasible completion-time vectors in the case that the partial order associated with the precedence relation is series-parallel. It is interesting to note, however, that these constraints cannot in general strengthen our approximation result, since the preceding example remains unaffected when these new inequalities are added.

We conclude this section with a few additional observations. First, notice that in equation (6) we have discarded the term $\frac{1}{2}p^2(S)$. By analyzing the inequality more carefully it is possible to show that Schedule-by-$\overline{C}_j$ is a $(2 - \frac{2}{n+1})$-approximation algorithm; see Schulz (1995) for the details.

Second, notice that our algorithmic results yield the following corollary concerning the quality of the optimal value of the linear program.

Corollary 2.4 *The linear program (1), (3) and (5) is a 2-relaxation of $1|prec|\sum w_j C_j$.*

In fact, by the observations just made, this linear program is actually a $(2 - \frac{2}{n+1})$-relaxation. Furthermore, we now give an example that shows that this analysis of the quality of the linear program is asymptotically tight as well. Consider an instance with n unit-length jobs in which the first $n-1$ jobs must precede job n but are otherwise independent. Let $w_j = 0$ for $j = 1, \ldots, n-1$, and $w_n = 1$. The optimal LP solution will set $\overline{C}_j = (n+1)/2 - 1/n$ for $j = 1, \ldots, n-1$, and $\overline{C}_n = (n+3)/2 - 1/n$; thus the overall LP objective value is $(n+3)/2 - 1/n$. On the other hand, the heuristic schedule, which is in fact an optimal schedule, has value n; thus, as $n \to \infty$, the ratio between the two values approaches 2. Note that this example is not a "bad" example for the algorithm, and the earlier example is not a "bad" example for the linear program. Moreover, this second example has a series-parallel precedence partial order, and so by adding the series inequalities to the linear program (1), (3) and (5), we would ensure that its extreme-point solutions also satisfy the disjunctive constraints (4).

The results of this section have implications for other LP formulations, as well; we give two basic examples here. The first formulation, which was given by Potts (1980), uses linear ordering variables δ_{ij}, where $\delta_{ij} = 1$ implies that job i precedes job j in the chosen schedule:

$$\text{minimize} \sum_{j=1}^{n} w_j C_j$$

subject to

$$\begin{aligned}
C_j &= \sum_{i=1}^{n} p_i \delta_{ij} + p_j, && j = 1, \ldots, n; \\
\delta_{ij} + \delta_{ji} &= 1, && i, j = 1, \ldots, n,\ i < j; \\
\delta_{ij} + \delta_{jk} + \delta_{ki} &\leq 2, && i, j, k = 1, \ldots, n,\ i < j < k \text{ or } i > j > k; \\
\delta_{ij} &= 1, && i, j = 1, \ldots, n,\ i \prec j; \\
\delta_{ij} &\geq 0, && i, j = 1, \ldots, n,\ i \neq j.
\end{aligned}$$

Notice, of course, that the C_j may be made implicit in this formulation, and from a set of δ_{ij} one could construct $C_j = \sum_{i=1}^{n} p_i \delta_{ij} + p_j$. Schulz (1995) has shown that these C_j are feasible for the linear program given by (3) and (5); consequently, the optimal value for the formulation in linear-ordering variables is at least the optimal value for the one given by the C_j decision variables. Hence, the linear-ordering formulation is also a 2-relaxation of $1|prec|\sum w_j C_j$. In addition, the linear ordering formulation is polynomial in size, and thus by using it in conjunction with our algorithm we can actually avoid the use of the ellipsoid algorithm in obtaining an optimal LP solution.

The next formulation, which was given by Dyer & Wolsey (1990), uses time-indexed variables. In this formulation we fix a time horizon $T = p(N)$ by which all jobs will be completed in any feasible schedule without unnecessary idle time. For each job $j = 1, \ldots, n$ and each $t = 1, \ldots, T$, we define $x_{jt} = 1$ if job j completes processing at time t. We then have the following LP relaxation:

$$\text{minimize} \sum_{j=1}^{n} w_j \sum_{t=1}^{T} t \cdot x_{jt}$$

subject to

$$\begin{aligned}
\sum_{t=1}^{T} x_{jt} &= 1, && j = 1, \ldots, n; && (7) \\
\sum_{s=1}^{t} x_{js} &\geq \sum_{s=1}^{t+p_k} x_{ks}, && \text{if } j \prec k,\ t = p_j, \ldots, T - p_k; && (8)
\end{aligned}$$

$$\sum_{j=1}^{n} \sum_{s=t}^{\min\{t+p_j-1,T\}} x_{js} \leq 1, \qquad t = 1, \ldots, T; \tag{9}$$

$$x_{jt} \geq 0, \qquad j = 1, \ldots, n,\ t = 1, \ldots, T; \tag{10}$$

$$x_{jt} = 0, \qquad t = 1, \ldots, p_j - 1. \tag{11}$$

Equation (7) says that each job must be assigned to some time slot; inequality (9) ensures that there is at most one job undergoing processing in the time interval $[t-1, t]$; and inequalities (8) enforce the precedence constraints, since they say that, for $j \prec k$, in order for k to be completed by time $t + p_k$, job j must be completed by time t, for all t.

This formulation has been reported to be quite strong in practice. However, it has both an exponential number of variables and constraints, and so significant effort has been devoted to developing efficient computational techniques to compute its solution (see, for example, Sousa & Wolsey (1992), Van den Akker (1994), Van den Akker, Hurkens, Savelsbergh (1994)). In Section 4 we will also introduce a closely related formulation that is polynomial in size and show how to use it to design approximation algorithms.

If we define $C_j = \sum_{t=p_j}^{T} t \cdot x_{jt}$ where x is a feasible solution to the linear program (7) – (10), then $C_1, \ldots, C_n$ are guaranteed to be feasible for (3) and (5) (Schulz, 1995); hence, the time-indexed formulation is a 2-relaxation as well. Van den Akker (1994) reports that the heuristic Schedule-by-$\overline{C}_j$ that uses $\overline{C}_j$ computed from the optimal solution to the time-indexed formulation is the best heuristic in practice for $1|r_j|\sum w_j C_j$. Therefore, our analysis of Schedule-by-$\overline{C}_j$ (and its extension to $1|r_j, prec|\sum w_j C_j$ in the next section) gives the first evidence from a worst-case perspective of the computational efficacy of this heuristic and the quality of lower bounds provided by these formulations. Although our analysis provides an identical performance guarantee for each of these three formulations, its seems likely that these formulations are not equivalently strong; for neither the linear-ordering formulation, nor the time-indexed formulation, have we been able to show that our analysis is tight. Clearly, for any LP-based approximation algorithm, the choice of formulation can have a big impact on the performance guarantee that one can hope to prove.

2.2 Single-machine scheduling with precedence constraints and release dates

Next we consider a more general model in which, in addition to precedence constraints, each job j has a release date r_j when it first becomes available for processing. We will demonstrate that an algorithm analogous to the one of the previous section is a 3-approximation algorithm.

Consider the following linear program given by (1), (2), (3), and (5). Suppose we solve the linear program to obtain an optimal solution $\overline{C}_1, \ldots, \overline{C}_n$; for simplicity we assume, as before, that $\overline{C}_1 \leq \cdots \leq \overline{C}_n$. Given the $\overline{C}_j$, we use the same heuristic, Schedule-by-$\overline{C}_j$: construct a feasible schedule by ordering the jobs according to non-decreasing $\overline{C}_j$. In this case, we might introduce idle time before the start of job j: if r_j is greater than the time at which job $j-1$ completes, then job j begins processing at time r_j.

Lemma 2.5 *Let $\overline{C}_1 \leq \cdots \leq \overline{C}_n$ be an optimal solution to the linear program defined by* (1), (2), (3), *and* (5), *and let $\widetilde{C}_1, \ldots, \widetilde{C}_n$ denote the completion times in the schedule found by* Schedule-by-$\overline{C}_j$. *Then, for $j = 1, \ldots, n$, $\widetilde{C}_j \leq 3\overline{C}_j$.*

Proof: Let us fix j and define $S = \{1, \ldots, j\}$. Since no idle time is introduced between $r_{\max}(S)$ and $\widetilde{C}_j$,

$$\widetilde{C}_j \leq r_{\max}(S) + p(S).$$

Moreover, by (2) and the ordering of the jobs we have that $r_{\max}(S) \leq \max_{k=1,\ldots,j} \overline{C}_k = \overline{C}_j$, and so

$$\widetilde{C}_j \leq \overline{C}_j + p(S).$$

Finally, by applying Lemma 2.1, we obtain our result. ■

Since this linear program can also be solved in polynomial time via the ellipsoid algorithm, we have the following theorem.

Theorem 2.6 Schedule-by-$\overline{C}_j$ *is a 3-approximation algorithm for* $1|r_j, prec|\sum w_j C_j$.

Moreover, the optimal value of the linear program is guaranteed to be within a factor of three of the optimal schedule value.

Corollary 2.7 *The linear program given by* (1), (2), (3), *and* (5) *is a* 3*-relaxation of* $1|r_j, prec|\sum w_j C_j$.

Again, these results have implications for other LP relaxations. We obtain a time-indexed formulation for this model by simply changing constraints (11) to

$$x_{jt} = 0, \qquad t = 1, \ldots, r_j + p_j - 1. \tag{12}$$

Then, if x satisfies (7) – (10) and (12), then $C_j = \sum_{t=p_j}^{T} t \cdot x_{jt}$ also satisfies the release-date constraints (2). Consequently, the time-indexed formulation is a 3-relaxation of $1|r_j, prec|\sum w_j C_j$. In the absence of precedence constraints, Dyer & Wolsey (1990) proposed a formulation in completion time variables C_j and another kind of time-indexed variables y_{jt}. Here, $y_{jt} = 1$ if job j is being processed in the time period $[t-1, t]$ and $y_{jt} = 0$, otherwise. The relaxation is as follows:

$$\text{minimize} \sum_{j=1}^{n} w_j C_j$$

subject to

$$\begin{aligned}
\sum_{j=1}^{n} y_{jt} &\leq 1, && t = 1, \ldots, T;\\
\sum_{t=1}^{T} y_{jt} &= p_j, && j = 1, \ldots, n;\\
\frac{p_j}{2} + \frac{1}{p_j}\sum_{t=1}^{T}(t - \frac{1}{2}) y_{jt} &= C_j, && j = 1, \ldots, n;\\
y_{jt} &\geq 0, && j = 1, \ldots, n, \ t = r_j, \ldots, T.
\end{aligned}$$

Goemans (1996) showed that this relaxation is equivalent to the following relaxation which solely uses completion time variables:

$$\text{minimize} \sum_{j=1}^{n} w_j C_j$$

subject to

$$\sum_{j \in S} p_j C_j \geq \ell(S), \quad \text{for each } S \subseteq N, \tag{13}$$

where

$$\ell(S) = r_{\min}(S) p(S) + \frac{1}{2}(p^2(S) + p(S)^2).$$

The valid inequalities (13) are a strengthened variant of (5) (see, e.g., Queyranne & Schulz (1995)). This implies that the linear program (1) and (13) also is a 3-relaxation of $1|r_j|\sum w_j C_j$. Since the polyhedron defined by constraints (13) is a linear transformation of a supermodular polyhedron (Goemans (1996)), we may apply the greedy algorithm for supermodular polyhedra to solve this particular relaxation. Combining this with algorithm Schedule-by-$\overline{C}_j$, we obtain a combinatorial 3-approximation algorithm for $1|r_j|\sum w_j C_j$ that runs in $O(n \log n)$ time.

2.3 Single-machine scheduling with preemption

The third model we consider is $1|r_j, prec, pmtn|\sum w_jC_j$, that is, the scheduling model in which jobs have release dates and precedence constraints, but the processing of a job may be interrupted and continued at a later point in time. Since the preemptive problem is a relaxation of the non-preemptive problem and constraints (2), (3), and (5) are all valid for the preemptive version of the problem, Theorem 2.6 immediately implies a 3-approximation algorithm for the preemptive problem. In this section we give a 2-approximation algorithm based on a strengthened linear programming relaxation.

Consider an instance of $1|r_j, prec, pmtn|\sum w_jC_j$. Notice that if $j \prec k$ and $r_j + p_j > r_k$, we can increase the value of r_k to $r_j + p_j$ without causing any feasible schedules to become infeasible. We begin by preprocessing the data in the instance in this manner so that, for all j, k with $j \prec k$, $r_k \geq r_j + p_j$. Next we consider the linear programming formulation given by constraints (2), (3), and (13).

Now, suppose we obtain an optimal linear programming solution to the system given by (1), (2), (3), and (13); call it $\overline{C}_1, \ldots, \overline{C}_n$. We construct a preemptive schedule from the LP solution as follows. We consider the jobs one at a time, in order of their $\overline{C}_j$ values; notice that the ordering is consistent with the precedence constraints, because of (3), and thus, by the time we consider job j, all of its predecessors have already been scheduled. To schedule job j, we find the first point in time, in the partially constructed schedule, at which all of the predecessors of j have completed processing, or time r_j, whichever is larger. Subsequent to that point in time we schedule parts of j in any idle time in the partial schedule, until j gets completely scheduled. We call this algorithm Preemptively-Schedule-by-$\overline{C}_j$.

Lemma 2.8 *Let $\overline{C}_1 \leq \cdots \leq \overline{C}_n$ be an optimal solution to the linear program defined by* (1), (2), (3), *and* (13), *and let $\widetilde{C}_1, \ldots, \widetilde{C}_n$ denote the completion times in the schedule found by* Preemptively-Schedule-by-$\overline{C}_j$. *Then, for $j = 1, \ldots, n$, $\widetilde{C}_j \leq 2\overline{C}_j$.*

Proof: Consider a job j, whose completion time in the constructed schedule is $\widetilde{C}_j$, and consider the partial schedule constructed by the algorithm for jobs $1, \ldots, j$. Let t be defined as the latest point in time prior to $\widetilde{C}_j$ at which there is idle time in this partial schedule (or, if no idle time exists before $\widetilde{C}_j$, set $t = 0$). Let S denote the set of jobs that are partially processed in the interval $[t, \widetilde{C}_j]$, in the partial schedule. First, we observe that no job of S gets released before time t; for if there were such a job, then by the preprocessing of the data there would also be a job k minimal in S with respect to $\prec$ for which this were true. But then job k would have been scheduled during part of the idle time prior to t, which it was not. Therefore, $t = r_{\min}(S)$, and since there is no idle time between t and $\widetilde{C}_j$,

$$\widetilde{C}_j \leq r_{\min}(S) + p(S).$$

Now we return to the strengthened inequalities (13). Recall that the set S was defined relative to the partial schedule obtained just after job j was scheduled; thus, for all $k \in S$, $\overline{C}_k \leq \overline{C}_j$. This fact, combined with (13), implies that

$$\overline{C}_j p(S) \geq \sum_{k \in S} p_k \overline{C}_k \geq r_{\min}(S) p(S) + \frac{1}{2} p(S)^2,$$

or $\overline{C}_j \geq r_{\min}(S) + p(S)/2$. Thus $\widetilde{C}_j \leq 2\overline{C}_j$, as we wished to show. ■

Because of our previous remarks concerning the polynomial solvability of the linear program, we have the following corollary.

Theorem 2.9
Preemptively-Schedule-by-$\overline{C}_j$ *is a 2-approximation algorithm for* $1|r_j, prec, pmtn|\sum w_jC_j$.

Once again, the proof of this theorem has the corollary that the linear programming optimum is within a factor of two of the optimal preemptive schedule value. When considering a preemptive model, it is also interesting to consider the ratio between the nonpreemptive and preemptive optima; that is, to bound the power of preemption. Phillips, Stein, & Wein (1995) and Lai (1995) showed that the optimum for $1|r_j|\sum w_jC_j$ is at most a factor of 2 more than its preemptive relaxation, and Lai (1995) showed that there exist instances for which the ratio is at least 18/13. Since the inequalities (2), (3), and (5) are all valid for preemptive schedules, the proof of Theorem 2.6 implies that the optimum for $1|r_j, prec|\sum w_jC_j$ is always within a factor of 3 of its preemptive relaxation; however, the technique of Phillips, Stein, & Wein (1995) easily extends to yield a bound of 2 for this case as well. Conversely, for any LP relaxation of the preemptive version, the ratio of the nonpreemptive optimum to the preemptive optimum is also a lower bound on the ratio of the nonpreemptive optimum to the LP optimum. Applying this to our strongest LP relaxation, we can conclude that there are instances of $1|r_j|\sum C_j$ for which the optimum value is at least 18/13 times the optimal value for the linear relaxation given by (1), (2) and (13).

3 Identical parallel machines

In this section we show that our approach can be extended to the more general setting in which we have m identical parallel machines; each job can be processed by any of the machines. In a nonpreemptive schedule a job must be processed, in an uninterrupted fashion, by exactly one machine, whereas in a preemptive schedule a job may be interrupted on one machine and continued on another at a later point in time; at any point in time a job may be processed by at most one machine.

The problem of minimizing the total weighted completion time on two identical parallel machines, either preemptively or nonpreemptively, was established to be $\mathcal{NP}$-hard by Bruno, Coffman & Sethi (1974) and Lenstra, Rinnooy Kan & Brucker (1977). We will again use variables C_j to denote the completion time of job j (irrespective of the machine on which it is processed). The convex hull of feasible completion time vectors has not been previously studied in this general setting. We can derive a class of valid inequalities for this model by generalizing the inequalities (5).

Lemma 3.1 *Let $C_1, \ldots, C_n$ denote the job completion times in a feasible schedule for $P||\sum w_jC_j$. Then the C_j satisfy the inequalities*

$$\sum_{j\in S} p_jC_j \geq \frac{1}{2m}\left(p(S)^2 + p^2(S)\right) \quad \textit{for each } S \subseteq N. \tag{14}$$

Proof: Without loss of generality, assume that there is no unforced idle time in the schedule, and that the jobs are indexed so that $C_1 \leq \cdots \leq C_n$. Consider the schedule induced for the subset of jobs $J = \{1, \ldots, j\}$. Job j is the last job to finish among jobs of J. If job j is scheduled on machine i, then i is the most heavily loaded machine (with respect to jobs in J). So the load on machine i is at least $p(J)/m$, and hence $C_j \geq p(J)/m = \sum_{k=1}^{j} p_k/m$. But then

$$\sum_{j=1}^{n} p_jC_j \geq (1/m)\sum_{j=1}^{n} p_j \sum_{k=1}^{j} p_k,$$

and then the usual arithmetic simplifies the right-hand side to yield (14) in the case where $S = \{1, \ldots, n\}$. The general case follows from the fact that a schedule for the entire set of jobs can be interpreted as a schedule for any subset. ■

In fact, Schulz (1995) has also shown that the following slightly stronger class of inequalities are valid:

$$\sum_{j\in S} p_jC_j \geq \frac{1}{2m}p(S)^2 + \frac{1}{2}p^2(S) \text{ for each } S \subseteq N.$$

However, our analyses of approximation algorithms will not require this strengthened class of inequalities. We show next that the inequalities (14) imply a kind of load constraint; this result is an immediate generalization of Lemma 2.1 in the single-machine setting.

Lemma 3.2 *Let $C_1, \ldots, C_n$ satisfy* (14) *and assume without loss of generality that $C_1 \leq \cdots \leq C_n$. Then for each $j = 1, \ldots, n$, if $S = \{1, \ldots, j\}$,*

$$C_j \geq \frac{1}{2m}p(S).$$

Proof: Let $S = \{1, \ldots, j\}$; from (14) and the fact that $C_k \leq C_j$ for each $k = 1, \ldots, j$, we have

$$C_j \cdot p(S) = C_j \sum_{k=1}^{j} p_k \geq \sum_{k=1}^{j} p_kC_k \geq \frac{1}{2m}(p(S)^2 + p^2(S)) \geq \frac{1}{2m}p(S)^2,$$

from which we obtain $C_j \geq \frac{1}{2m}p(S)$. ■

Note that inequalities (14) and Lemma 3.2 apply to both preemptive and nonpreemptive schedules.

As in the single-machine setting, our approximation algorithms are based on solving a linear programming relaxation in the C_j variables and then scheduling the jobs in a natural order dictated by the solution to the linear program. For several models, simple variants of a list-scheduling rule that are based on the LP solution yield excellent performance guarantees; we present these algorithms and their analysis in Sections 3.1 and 3.2. For the most general version of this problem a somewhat more complex approach will be necessary; we present this result in Section 3.3. We note in advance that with every approximation algorithm we obtain a bound on the quality of the associated linear programming relaxation; to avoid excess verbiage we omit explicit statements of these corollaries.

3.1 Independent jobs

We begin by considering the problem $P|r_j|\sum w_jC_j$; as our linear program, we minimize $\sum_j w_jC_j$ subject to constraints (14) and release-date constraints (2), which of course remain valid in the parallel machine setting.

Our algorithm Start-Jobs-by-$\overline{C}_j$ works as follows. We first compute an optimal solution $\overline{C}_1, \ldots, \overline{C}_n$ to this linear program; we again assume without loss of generality that $\overline{C}_1 \leq \cdots \leq \overline{C}_n$. We schedule the jobs iteratively in the order of this list, and for each job j we consider the current schedule after time r_j, and identify the earliest block of p_j consecutive idle time units on some machine in which to schedule this job.

Lemma 3.3 *Let $\overline{C}_1 \leq \cdots \leq \overline{C}_n$ be an optimal solution to the linear program defined by* (1), (2), *and* (14), *and let $\widetilde{C}_1, \ldots, \widetilde{C}_n$ denote the completion times in the schedule found by* Start-Jobs-by-$\overline{C}_j$. *For each $j = 1, \ldots, n$,*

$$\widetilde{C}_j \leq (4 - \frac{1}{m})\overline{C}_j.$$

Proof: Consider the schedule induced by the jobs $1, \dots, j$, and let $S = \{1, \dots, j\}$. Any idle period on a machine in this partial schedule must end at the release date of some job in S. Consequently, all machines are busy between time $r_{\max}(S)$ and the start of job j. Thus

$$\begin{aligned} \tilde{C}_j &\leq r_{\max}(S) + \frac{1}{m} p(S \setminus \{j\}) + p_j \\ &= r_{\max}(S) + \frac{1}{m} p(S) + (1 - \frac{1}{m}) p_j. \end{aligned} \tag{15}$$

To bound this expression, we note that the constraints of the LP formulation ensure that $\overline{C}_j \geq p_j$, and, since $\overline{C}_j \geq \overline{C}_k$ for $k = 1, \dots, j$, that $\overline{C}_j \geq r_{\max}(S)$. By Lemma 3.2 we have

$$2\overline{C}_j \geq \frac{1}{m} p(S),$$

which yields an overall upper bound of $(4 - \frac{1}{m})\overline{C}_j$ on $\tilde{C}_j$. ■

To solve the linear program in polynomial time we again use the ellipsoid algorithm; the separability of the constraints follows from the fact that the inequalities (14) are merely a rescaled version of (5), which Queyranne (1993) proved are separable. Thus we have the following theorem.

Theorem 3.4 Start-Jobs-by-$\overline{C}_j$ *is a* $(4 - \frac{1}{m})$*-approximation algorithm for* $P|r_j|\sum w_j C_j$.

The ideas used in this result are similar to those introduced by Phillips, Stein & Wein (1995) to convert preemptive parallel machine schedules to nonpreemptive schedules.

For the case in which there are no non-trivial release dates, Kawaguchi & Kyan (1986) have shown that the following is a $\frac{(\sqrt{2}+1)}{2}$-approximation algorithm: order the jobs by non-decreasing ratio p_j/w_j and apply the list-scheduling algorithm of Graham. In this special case, our algorithm Start-Jobs-by-$\overline{C}_j$ is closely related to this algorithm; assume that the jobs are indexed so that $p_1/w_1 \leq \cdots \leq p_n/w_n$. Suppose that we started by solving a somewhat weaker linear program instead: minimize $\sum_j w_j C_j$ subject to (14). In other words, we relax the constraint that $C_j \geq p_j$, $j = 1, \dots, n$. However, this is the same linear program as we would solve for a 1-machine input in which job j requires p_j/m units of processing. By the theorem of Queyranne (1993), the optimal solution to this linear program is $\overline{C}_j = p(\{1, \dots, j\})/m$. In other words, our modified algorithm is exactly the algorithm of Kawaguchi & Kyan (1986). Furthermore, equation (15) implies that

$$\tilde{C}_j \leq p(S)/m + (1 - \frac{1}{m}) p_j \leq \overline{C}_j + (1 - \frac{1}{m}) C_j^*,$$

and hence we obtain a simple proof that the algorithm of Kawaguchi & Kyan is a $(2 - \frac{1}{m})$-approximation algorithm. More importantly, this analysis implies the following bound on the strength of the linear relaxation used by Start-Jobs-by-$\overline{C}_j$.

Corollary 3.5 *The linear program* (1), (2), *and* (14) *is a* $(2 - \frac{1}{m})$*-relaxation of* $P||\sum w_j C_j$.

3.2 Preemptive scheduling and unit-time jobs

We consider next the preemptive variant, $P|r_j, prec, pmtn|\sum w_j C_j$, in which we are allowed to interrupt the processing of a job and continue it later, possibly on another machine. We will give a simple 3-approximation algorithm for this problem. Furthermore, if all of the jobs are unit-length and the release dates are integral, then the algorithm does not introduce any preemptions; hence, this also yields a 3-approximation algorithm for $P|r_j, prec, p_j = 1|\sum w_j C_j$.

In his ground-breaking paper, Graham (1966) showed that a simple list-scheduling rule is a $(2-\frac{1}{m})$-approximation algorithm for $P|prec|C_{\max}$. In this algorithm, the jobs are ordered in some list, and whenever one of the m machines becomes idle, the next available job on the list is started on that machine, where a job is available if all of its predecessors have completed processing. Graham actually showed that when this algorithm is used to schedule a set N of jobs, the length $C_{\max}$ of the resulting schedule is at most

$$\frac{1}{m}p(N \setminus \mathcal{C}) + p(\mathcal{C}),$$

where $\mathcal{C}$ denotes the set of jobs that form the longest chain (with respect to processing times) of precedence-constrained jobs ending with the job that completes last in the schedule.

We shall analyze a preemptive variant of Graham's list-scheduling rule. The jobs are listed in order of non-decreasing $\overline{C}_j$ value, where $\overline{C}_j$, $j = 1, \ldots, n$, denotes an optimal solution to the linear program (1), (2), (3), and (14); once again, we shall assume that the jobs are indexed so that $\overline{C}_1 \le \overline{C}_2 \le \cdots \le \overline{C}_n$. A job j is *available* for processing in a schedule at time t, if $r_j \le t$ and all predecessors of job j have completed by time t. A machine is *available* at time t if it not assigned to be processing a job at that time. The algorithm Preemptively-List-Schedule-by-$\overline{C}_j$ constructs the schedule "in time". If machine i becomes available at time t, then, among all currently available jobs, this machine is assigned to process the one that occurs earliest in the list. If a job j becomes available at time t, and there is a job currently being processed that occurs later on the list than j, then, among all jobs currently being processed, job j preempts the one that occurs latest in the list.

Observe that each preemption in the schedule can be associated with the release of a job. Hence, there are at most $n-1$ preemptions in the schedule found by Preemptively-List-Schedule-by-$\overline{C}_j$. Furthermore, if the release dates are integral, then all preemptions occur at integer points. This implies that if all jobs are of unit length, then no preemptions occur, and the schedule found is actually a nonpreemptive one; in this case, Preemptively-List-Schedule-by-$\overline{C}_j$ is precisely the list-scheduling algorithm of Graham.

Theorem 3.6 *For $P|r_j, prec, pmtn|\sum w_jC_j$,* Preemptively-List-Schedule-by-$\overline{C}_j$ *is a 3-approximation algorithm.*

Proof: The proof of this result is very similar in spirit to Graham's original analysis for $P|prec|C_{\max}$. Let $\widetilde{C}_1, \ldots, \widetilde{C}_n$ be the completion times of the scheduled jobs. Let us focus on a particular job j. We claim that the time interval from 0 to $\widetilde{C}_j$ can be partitioned into two sets of intervals; the total length of one of these sets can be bounded above by $\overline{C}_j$, while the length of the other can be bounded above by $2\overline{C}_j$.

We construct the partition as follows. Let $t_0 = \widetilde{C}_j$ and $j_1 = j$. We first derive a chain of jobs $j_s \prec j_{s-1} \prec \cdots \prec j_1$ from the schedule in the following way. Inductively, for $k = 1, \ldots, s$, define t_k as the time at which job j_k becomes available; if $r_{j_k} = t_k$, then set $s = k$, and the construction is complete. Otherwise, let j_{k+1} denote a predecessor of j_k that completes at time t_k. Clearly, we have that $j_s \prec j_{s-1} \prec \cdots \prec j_1$; let $\mathcal{C}$ denote the set of jobs in this chain. A simple inductive argument shows that the constraints (2) and (3) imply that $\overline{C}_j \ge r_{j_s} + p(\mathcal{C})$. We can think of this lower bound as the total length of the union of the (disjoint) time intervals in which some job in this chain is being processed, together with the interval $(0, r_{j_s}]$. So to compute an upper bound on $\widetilde{C}_j$, we need only consider the complementary set of time intervals within $(0, \widetilde{C}_j]$: let $\mathcal{T}$ denote the set of times t between t_s and t_0 during which no job in $\mathcal{C}$ is being processed.

We wish to show that $\mathcal{T}$ consists of (disjoint) intervals of time of total length at most $2\overline{C}_j$. Consider any point in time $t \in \mathcal{T}$ in the interval $(t_k, t_{k-1}]$, $k = 1, \ldots, s$: at this point in time, the job j_k is available. Since it is not being processed, this implies that no machine is idle; furthermore, each job being processed must occur earlier in the list than j_k, and hence earlier in the list than j. In other words, for every $t \in \mathcal{T}$, each machine is processing some job in $S = \{1, \ldots, j\} \setminus \mathcal{C}$. Hence the total length of $\mathcal{T}$ is at most $p(S \setminus \mathcal{C})/m$; by Lemma 3.2, $p(S)/m \leq 2\overline{C}_j$.

Thus $\widetilde{C}_j \leq 3\overline{C}_j$, for each $j = 1, \ldots, n$. Noting once again that the linear program can be solved in polynomial time, we have established our theorem. ■

As we noted above, this also implies the following result.

Corollary 3.7 *For $P|r_j, prec, p_j = 1|\sum w_j C_j$, Preemptively-List-Schedule-by-$\overline{C}_j$ is a 3-approximation algorithm.*

If $r_j = 0$, $j = 1, \ldots, n$, then we can slightly refine the analysis of Theorem 3.6. In this case, we can partition the schedule into $\mathcal{T}$ and the periods of time in which some job in $\mathcal{C}$ is being processed. Hence,

$$\widetilde{C}_j \leq p(S \setminus \mathcal{C})/m + p(\mathcal{C}) = p(S)/m + (1 - \frac{1}{m})p(\mathcal{C}).$$

This implies that Preemptively-List-Schedule-by-$\overline{C}_j$ is a $(3 - \frac{1}{m})$-approximation algorithm for both $P|prec, pmtn|\sum w_j C_j$ and $P|prec, p_j = 1|\sum w_j C_j$.

3.3 The general problem

We next consider $P|r_j, prec|\sum w_j C_j$ in its full generality. Unfortunately, we do not know how to prove a good performance guarantee for this model by using a simple list-scheduling variant. However, we are able to give a 7-approximation algorithm for $P|r_j, prec|\sum w_j C_j$, by considering a somewhat more sophisticated algorithm. Observe that if we use only *one* of our m machines, and schedule the jobs in order of their LP optimal values, then Lemma 3.2 implies that this schedule has objective function value within a factor of $2m + 1$ of the m-machine optimum (and within a factor of $2m$ if all release dates are 0). Hence, for $m \leq 3$, this dominates the more sophisticated approach.

Our algorithm for $P|r_j, prec|\sum w_j C_j$, which we call Interval-Schedule-by-$\overline{C}_j$, begins as before by finding the optimal solution $\overline{C}_1, \ldots, \overline{C}_n$ to the linear program to minimize $\sum_j w_j C_j$ subject to (2), (3), and (14); as before, we assume that $\overline{C}_1 \leq \cdots \leq \overline{C}_n$. Next, we divide the time line into intervals $[1, 1], (1, 2], (2, 4], \ldots, (2^{L-2}, 2^{L-1}]$, where L is the smallest integer such that 2^{L-1} is at least $r_{\max}(N) + p(N)$ (i.e., an upper bound on the length of any feasible schedule with no unforced idle time). For conciseness, let $\tau_0 = 1$ and $\tau_\ell = 2^{\ell-1}$, $\ell = 1, \ldots, L$. We use $\ell(j)$ to denote the index of the upper endpoint of the interval in which $\overline{C}_j$ lies, i.e., the smallest value of $\ell \geq 1$ such that $\tau_\ell \geq \overline{C}_j$. Furthermore, let S_ℓ denote the set of jobs j with $\ell(j) = \ell$, $\ell = 1, \ldots, L$. We define $t_\ell = (1/m)p(S_\ell)$; t_ℓ can be thought of as the average load on a machine for the set S_ℓ. For each $\ell = 0, 1, \ldots, L$, we set

$$\overline{\tau}_\ell = 1 + \sum_{k=1}^{\ell} (\tau_k + t_k).$$

We schedule the jobs in S_ℓ, using the list-scheduling algorithm of Graham, in the interval $\overline{\tau}_{\ell-1}$ to $\overline{\tau}_\ell$.

Theorem 3.8 Interval-Schedule-by-$\overline{C}_j$ *is a 7-approximation algorithm for $P|r_j, prec|\sum w_j C_j$.*

Proof: We first show that this is a feasible schedule. The constraints (3) ensure that the precedence constraints are enforced, since for each job $j \in S_\ell$, each of its predecessors is assigned to S_k for some $k \in \{1, \dots, \ell\}$. We also need to show that the schedule respects the release-date constraints. If $j \in S_\ell$, $\ell = 1, \dots, L$, then $r_j \leq \overline{C}_j \leq \tau_\ell$. However,

$$\overline{\tau}_{\ell-1} \geq 1 + \sum_{k=1}^{\ell-1} \tau_k = \tau_\ell,$$

and hence $r_j \leq \overline{\tau}_{\ell-1}$. Hence, the analysis of the list-scheduling rule for each interval reduces to the case without release dates. Graham's analysis implies that the length of the schedule constructed for S_ℓ can be bounded by the maximum length of any precedence chain, plus the average load on a machine. The constraints (3) ensure that the maximum length of a chain in S_ℓ is at most τ_ℓ, and the average load is t_ℓ. Hence, we have allocated sufficient time to complete this fragment of the schedule.

Next we show that each job j completes by time at most $7\overline{C}_j$. Consider the completion time of job $j \in S_\ell$. By the Graham-like analysis discussed in the proof of Theorem 3.6, $\widetilde{C}_j$ is bounded above by $\overline{\tau}_{\ell-1} + t_\ell + \beta_j$, where β_j is the length of some chain that ends with job j. Combining terms, we can rewrite this bound as $1 + \sum_{k=1}^{\ell-1} \tau_k + \sum_{k=1}^{\ell} t_k + \beta_j$, which is at most $\tau_\ell + \sum_{k=1}^{\ell} t_k + \overline{C}_j$ (recall that $\beta_j \leq \overline{C}_j$). Consider the job $j(\ell) \in S_1, \dots, S_\ell$ whose $\overline{C}_j$-value is largest. Lemma 3.2 implies that

$$\sum_{k=1}^{\ell} t_k = (1/m) \sum_{k=1}^{\ell} p(S_k) \leq 2\overline{C}_{j(\ell)} \leq 2\tau_\ell. \tag{16}$$

Thus

$$\widetilde{C}_j \leq \tau_\ell + 2\tau_\ell + \overline{C}_j \leq 7\overline{C}_j,$$

since $\tau_\ell \leq 2\overline{C}_j$. This completes the proof. ■

We note that subsequent to this work, Chakrabarti, Phillips, Schulz, Shmoys, Stein, & Wein (1996) proposed a 5.33-approximation algorithm based on the same linear programming formulation. Finally, since the inequalities (2), (3), and (14) are all valid for the preemptive relaxation, we have also shown that the ratio between the nonpreemptive optimum and the preemptive optimum is at most 7. In fact, if we replace $\overline{C}_j$ by the completion time of job j in an optimal preemptive schedule, then the proof of Theorem 3.8 implies that this ratio is at most 5: instead of inequality (16), we know that the total processing requirement of jobs finishing by τ_ℓ in the optimal preemptive schedule is at most $m\tau_\ell$. The result of Chakrabarti et al. (1996) also implies an even tighter upper bound on this ratio.

4 Interval-indexed formulations and unrelated machines

In this section we consider the problem of scheduling on unrelated parallel machines, and give a (16/3)-approximation algorithm for $R|r_j|\sum_j w_j C_j$. In contrast to the results of the previous sections, we do not use linear programming formulations in C_j variables, but rather a formulation inspired by time-indexed linear programming formulations. We shall introduce the notion of an *interval-indexed* formulation, in which the decision variables merely indicate in which time-interval a given job completes. The intervals are constructed by partitioning the time horizon at geometrically increasing points; consequently, unlike the time-indexed formulation, this new formulation is of polynomial size. Furthermore, since the ratio between the endpoints of each interval is bounded by a constant, we can assign a job to complete within this interval without too much concern about when within the interval it actually completes.

We will, in fact, consider a slightly more general problem, in which the release date of a job may depend on the machine, and is thus denoted r_{ij}: job j may not be processed on machine i until time r_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$. This model will also be relevant to our discussion of network scheduling models.

We will first give an 8-approximation algorithm that is somewhat simpler to explain. We can divide the time horizon of potential completion times into the following intervals: $[1, 1]$, $(1, 2]$, $(2, 4], \ldots, (2^{L-2}, 2^{L-1}]$, where L is chosen to be the smallest integer such that $2^{L-1} \geq \max_j r_{ij} + \sum_j \max_i p_{ij}$; that is, 2^{L-1} is a sufficiently large time horizon. For conciseness, let $\tau_0 = 1$, and $\tau_\ell = 2^{\ell-1}$, $\ell = 1, \ldots, L$, and so the ℓth interval runs from time $\tau_{\ell-1}$ to τ_ℓ, $\ell = 1, \ldots, L$.

Consider the following linear programming relaxation, in which the interpretation of each $0-1$ decision variable $x_{ij\ell}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, and $\ell = 1, \ldots, L$, is to indicate if job j is scheduled to complete on machine i within the interval $(\tau_{\ell-1}, \tau_\ell]$:

$$\text{minimize} \quad \sum_{j=1}^{n} w_j \sum_{i=1}^{m} \sum_{\ell=1}^{L} \tau_{\ell-1} x_{ij\ell} \tag{17}$$

subject to

$$\sum_{i=1}^{m} \sum_{\ell=1}^{L} x_{ij\ell} = 1, \qquad j = 1, \ldots, n; \tag{18}$$

$$\sum_{j=1}^{n} p_{ij} x_{ij\ell} \leq \tau_\ell, \qquad i = 1, \ldots, m, \ \ell = 1, \ldots, L; \tag{19}$$

$$x_{ij\ell} = 0, \qquad \text{if } \tau_\ell < r_{ij} + p_{ij}; \tag{20}$$

$$x_{ij\ell} \geq 0, \qquad i = 1, \ldots, m, \ j = 1, \ldots, n, \ \ell = 1, \ldots, L. \tag{21}$$

Lemma 4.1 *For $R|r_{ij}|\sum w_j C_j$, the optimal value of the linear program* (17) – (21) *is a lower bound on the optimal total weighted completion time, $\sum w_j C_j^*$.*

Proof: Consider an optimal schedule and set $x_{ij\ell} = 1$ if job j is assigned to machine i and completes within the ℓth interval. This solution is clearly feasible: constraints (19) are satisfied since the total processing requirement on machine i of jobs that complete within $(\tau_{\ell-1}, \tau_\ell]$ is at most τ_ℓ; constraints (20) are satisfied since any job j that completes by τ_ℓ on machine i must have $r_{ij} + p_{ij} \leq \tau_\ell$. Finally, if job j completes within the ℓth interval, $\ell = 1, \ldots, L$, then its completion time is at least $\tau_{\ell-1}$; hence, the objective function value of the feasible solution constructed is no more than $\sum w_j C_j^*$. ■

One unusual aspect of the formulation (17) – (21) is that it is the linear relaxation of an integer program that is, itself, a relaxation of the original problem. We believe that this idea might prove useful in other settings as well.

Our rounding technique is based on the observation that this linear program is essentially the same as the one considered by Shmoys & Tardos (1993) for the generalized assignment problem. We will apply their rounding technique both in this section and the next; we therefore present a brief discussion of the generalized assignment problem and the main result of Shmoys & Tardos (1993).

Shmoys & Tardos consider the following linear program in the setting with m unrelated machines and n jobs, where processing job j on machine i requires p_{ij} time units and incurs a cost c_{ij}, for each $i = 1, \ldots, m$, $j = 1, \ldots, n$ (and the costs need not be non-negative):

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \leq C \tag{22}$$

$$\begin{aligned}\sum_{i=1}^{m} x_{ij} &= 1, && \text{for } j = 1,\dots,n, && (23)\\ \sum_{j=1}^{n} p_{ij}x_{ij} &\le T_i, && \text{for } i = 1,\dots,m, && (24)\\ x_{ij} &\ge 0, && i = 1,\dots,m,\ j = 1,\dots,n. && (25)\end{aligned}$$

Theorem 4.2 (Shmoys & Tardos (1993)) *There is a polynomial-time algorithm that, given a feasible solution x to the linear program* (22) – (25), *rounds this solution to an integer solution $\bar{x}$ such that*

$$x_{ij} = 0 \Rightarrow \bar{x}_{ij} = 0 \qquad (26)$$

and $\bar{x}$ satisfies constraints (22), (23), *and*

$$\sum_{j=1}^{n} p_{ij}\bar{x}_{ij} \le t_i + \max_{j:x_{ij}>0} p_{ij}, \qquad \text{for each } i = 1,\dots,m, \qquad (27)$$

where $t_i = \sum_{j=1}^{n} p_{ij}x_{ij}$. Furthermore, if we let $J_i = \{j : \bar{x}_{ij} = 1\}$, $i = 1,\dots,m$, then each $J_i = S_i \cup B_i$, where $\sum_{j\in S_i} p_{ij} \le t_i$, and $|B_i| \le 1$, $i = 1,\dots,m$. Finally, the analogous theorem holds if we replace constraints (23) *with $\sum_{i=1}^{m} x_{ij} \le 1$, $i = 1,\dots,m$.*

Observe that the properties of S_i and B_i imply that equation (27) holds: the total processing requirement of S_i is bounded by t_i, and, by (26), the job j in B_i must have $x_{ij} > 0$.

Consider again the linear relaxation (17) – (21). If we view a machine-interval pair as a virtual machine, then this linear program is a further constrained variant of (22) – (25), where (20) are the only additional constraints. Nonetheless, any feasible solution to (17) – (21) can be viewed as a feasible solution to (22) – (25), and hence Theorem 4.2 can be applied to round this solution.

We can use this rounding theorem to devise an approximation algorithm for $R|r_{ij}|\sum w_jC_j$. We first solve the linear program (17) – (21), apply the rounding technique of Shmoys & Tardos (1993) to this feasible solution x, and interpret the rounded integer solution $\bar{x}$ as a schedule in the following way. Consider the set of jobs $J_{i\ell} = \{j : \bar{x}_{ij\ell} = 1,\ \ i = 1,\dots,m,\ \ell = 1,\dots,L\}$, and let $\bar{\tau}_\ell = \sum_{k=0}^{\ell} 2\tau_k = 2^{\ell+1}$, $\ell = 0,\dots,L$. We shall process each job $j \in J_{i\ell}$ on machine i entirely within the interval from $\bar{\tau}_{\ell-1}$ to $\bar{\tau}_\ell$ (where the ordering of jobs within this interval is arbitrary). First, we shall show that this is feasible. Clearly, $\tau_\ell \le \bar{\tau}_{\ell-1}$, $\ell = 1,\dots,L$. If $r_{ij} + p_{ij} > \tau_\ell$, then $x_{ij} = 0$, and hence $\bar{x}_{ij} = 0$; in other words, for each job $j \in J_{i\ell}$, $r_{ij} \le r_{ij} + p_{ij} \le \tau_\ell$, $i = 1,\dots,m$, $\ell = 1,\dots,L$. Putting these two facts together, we see that job j is released by time $\bar{\tau}_{\ell-1}$. Furthermore, the total processing requirement of the jobs in $J_{i\ell}$ satisfies

$$\sum_{j\in J_{i\ell}} p_{ij} \le \tau_\ell + \max_{j:x_{ij\ell}>0} p_{ij} \le 2\tau_\ell = \bar{\tau}_\ell - \bar{\tau}_{\ell-1},$$

and so these jobs can all be processed by machine i entirely within the interval from $\bar{\tau}_{\ell-1}$ to $\bar{\tau}_\ell$.

To analyze the quality of this schedule, first note that the objective function value (17) of the rounded solution $\bar{x}$ is at most the objective function value of the optimal LP solution x (by Theorem 4.2). Observe that each $j \in J_{i\ell}$ contributes $w_j\tau_{\ell-1}$ to the objective function value (17) of $\bar{x}$, whereas it completes by time $\bar{\tau}_\ell$ in our schedule and hence contributes at most $w_j\bar{\tau}_\ell$ to the objective function value of the schedule found. Since $\bar{\tau}_\ell/\tau_{\ell-1} \le 8$, $\ell = 1\dots,L$, the total weighted completion time of the schedule found is no more than 8 times the objective value of the rounded solution $\bar{x}$. Hence, we have found a schedule with total weighted completion time within a factor of 8 of optimal.

To give an improved performance guarantee, we note that the linear programming relaxation (17) – (21) is quite weak in the following sense. The load constraints (19) limit the load on machine i for the ℓth interval to τ_ℓ. If, for example, this constraint is satisfied with equality, then for each of the previous intervals, machine i can have no load whatsoever. We can capture this by adding the following constraints:

$$\sum_{k=1}^{\ell}\sum_{j=1}^{n} p_{ij}x_{ijk} \quad \leq \quad \tau_\ell, \qquad\qquad i=1,\ldots,m,\ \ell=1,\ldots,L. \tag{28}$$

Lemma 4.3 *For $R|r_{ij}|\sum w_jC_j$, the optimal value of the linear program* (17) – (21) *and* (28), *is a lower bound on the optimal total weighted completion time, $\sum w_jC_j^*$.*

We show next that Theorem 4.2 implies that an optimal solution to the strengthened linear relaxation can be rounded to yield a schedule better than the one found above. As above, any feasible solution to the linear program (17) – (21) and (28) can be viewed as a feasible solution to (22) – (25). Let x denote the optimal solution to the strengthened linear relaxation, and let $t_{i\ell} = \sum_{j=1}^{n} p_{ij}x_{ij\ell}$, $i=1,\ldots,m$, $\ell=1,\ldots,L$; thus, $\sum_{k=1}^{\ell} t_{ik} \leq \tau_\ell$, $\ell=1,\ldots,L$. The rounding theorem produces an integer solution $\bar{x}$ which can be interpreted as the job partition $J_{i\ell}$, $i=1,\ldots,m$, $\ell=1,\ldots,L$, where each set $J_{i\ell} = B_{i\ell}\cup S_{i\ell}$ such that (i) $\sum_{j\in S_{i\ell}} p_{ij} \leq t_{i\ell}$, (ii) $|B_{i\ell}|\leq 1$, and (iii) for each job $j\in B_{i\ell}\cup S_{i\ell}$, $r_{ij}+p_{ij}\leq \tau_\ell$.

Consider some machine $i=1,\ldots,m$. We construct the following schedule: let

$$\bar{\tau}_{i\ell} = 1 + \sum_{k=1}^{\ell}(\tau_k + t_{ik}), \quad \ell = 0,\ldots,L; \tag{29}$$

the jobs in $B_{i\ell}\cup S_{i\ell}$ are scheduled in the interval from $\bar{\tau}_{i,\ell-1}$ to $\bar{\tau}_{i,\ell}$ sorted in the order of nondecreasing p_j/w_j ratio. We shall call this the Greedy LP-interval algorithm. (Observe that if we changed (29) by replacing t_{ik} with its upper bound τ_k, implied by (19), then $\bar{\tau}_{i\ell}$ is simply $\bar{\tau}_\ell$.)

Lemma 4.4 *The schedule produced by the algorithm* Greedy LP-interval *is feasible.*

Proof: Consider some machine $i=1,\ldots,m$: we must show that the release dates are respected and that each set of jobs $B_{i\ell}\cup S_{i\ell}$, $\ell=1,\ldots,L$, fits into its assigned interval. Since $\tau_\ell = 2^{\ell-1}$, $\ell=1,\ldots,L$, we see that

$$\bar{\tau}_{i,\ell-1} \geq 1 + \sum_{k=1}^{\ell-1}\tau_k = \tau_\ell.$$

For each job $j\in B_{i\ell}\cup S_{i\ell}$, we have that $r_{ij}\leq r_{ij}+p_{ij}\leq\tau_\ell$ and hence job j has been released by time $\bar{\tau}_{i,\ell-1}$. Furthermore, we have that

$$\sum_{j\in B_{i\ell}\cup S_{i\ell}} p_{ij} \leq \tau_\ell + t_{i\ell} = \bar{\tau}_{i\ell} - \bar{\tau}_{i,\ell-1},$$

and so these jobs can all be processed entirely within this interval. ■

Lemma 4.5 *Consider the class of all schedules $\mathcal{S}$ in which every job $j\in B_{i\ell}\cup S_{i\ell}$ is scheduled entirely within the interval from $\bar{\tau}_{i,\ell-1}$ to $\bar{\tau}_{i\ell}$, for each $i=1,\ldots,m$, $\ell=1,\ldots,L$. Among all schedules in $\mathcal{S}$, the* Greedy LP-interval *algorithm produces a schedule of minimum total weighted completion time.*

Proof: The proof of Lemma 4.4 implies that any schedule in $\mathcal{S}$ is feasible. For any schedule in $\mathcal{S}$, view the completion time of each job $j \in B_{i\ell} \cup S_{i\ell}$ as $\bar{\tau}_{i,\ell-1} + \widehat{C}_j$. When optimizing among all schedules in $\mathcal{S}$, the problem of minimizing $\sum_j w_j \widehat{C}_j$ is equivalent to the problem of minimizing $\sum_j w_j C_j$. However, in former case, it is clear that we have mL independent sequencing problems, and each is equivalent to an instance of $1||\sum w_j C_j$. By the classical result of Smith (1956), each can be solved by ordering the jobs in $B_{i\ell} \cup S_{i\ell}$ in order of non-decreasing ratio p_j/w_j. However, this is exactly our algorithm Greedy LP-interval. ■

Given the rounded solution $\bar{x}$, let $\overline{C}_j = \tau_{\ell-1}$ whenever $\bar{x}_{ij\ell} = 1$; in other words, $\overline{C}_j$ is the completion time that the rounded solution $\bar{x}$ is charged for job j in its objective function (17).

Theorem 4.6 *For $R|r_{ij}|\sum w_j C_j$,* Greedy LP-interval *is a $\frac{16}{3}$-approximation algorithm.*

Proof: We will show that the schedule produced by the Greedy LP-interval algorithm is good by analyzing two *other* ways to sequence the jobs within each interval, and then showing that one of the two resulting schedules has total weighted completion time within a factor of 16/3 of optimal. By Lemma 4.5, this implies the theorem.

The two schedules that we consider are as follows: for each interval, either always assign the job in $B_{i\ell}$ before the jobs in $S_{i\ell}$, $\ell = 1, \ldots, L$, or vice versa. Observe that for any sequence of the jobs in $B_{i\ell} \cup S_{i\ell}$ in its interval, each such job j completes by time

$$\bar{\tau}_{i\ell} = 1 + \sum_{k=1}^{\ell} t_{ik} + \sum_{k=1}^{\ell} \tau_k \leq 1 + \tau_\ell + \sum_{k=1}^{\ell} \tau_k = \tau_\ell + \tau_{\ell+1} \leq 6\tau_{\ell-1} = 6\overline{C}_j.$$

This implies that the algorithm is a 6-approximation algorithm, but we will show something a bit stronger. We first consider the schedule in which each "B" job is scheduled first in its interval. In that case, the job $j \in B_{i\ell}$ completes by time

$$1 + \sum_{k=1}^{\ell-1} t_{ik} + \sum_{k=1}^{\ell} \tau_k \leq 1 + \tau_{\ell-1} + \sum_{k=1}^{\ell} \tau_k = \tau_{\ell-1} + \tau_{\ell+1} \leq 5\tau_{\ell-1} = 5\overline{C}_j.$$

On the other hand, in the schedule in which each "B" job is scheduled last in its interval, each job $j \in S_{i\ell}$ completes by time

$$1 + \sum_{k=1}^{\ell} t_{ik} + \sum_{k=1}^{\ell-1} \tau_k \leq 1 + \tau_\ell + \sum_{k=1}^{\ell-1} \tau_k = \tau_\ell + \tau_\ell \leq 4\tau_{\ell-1} = 4\overline{C}_j.$$

Let $\omega_B = \sum_{i=1}^m \sum_{\ell=1}^L \sum_{j \in B_{i\ell}} w_j \overline{C}_j$ and $\omega_S = \sum_{i=1}^m \sum_{\ell=1}^L \sum_{j \in S_{i\ell}} w_j \overline{C}_j$. By Theorem 4.2, $\omega_B + \omega_S$ is a lower bound on the optimal value, $\sum_j w_j C_j^*$. The first schedule has total weighted completion time at most $6\omega_S + 5\omega_B$, and the second one has total weighted completion time at most $4\omega_S + 6\omega_B$. Suppose that $\omega_S = \alpha(\omega_B + \omega_S)$. If $\alpha \geq 1/3$, then

$$4\alpha(\omega_B + \omega_S) + 6(1-\alpha)(\omega_B + \omega_S) = (6 - 2\alpha)(\omega_B + \omega_S) \leq (16/3)(\omega_B + \omega_S).$$

On the other hand, if $\alpha \leq 1/3$,

$$6\alpha(\omega_B + \omega_S) + 5(1-\alpha)(\omega_B + \omega_S) = (5 + \alpha)(\omega_B + \omega_S) \leq (16/3)(\omega_B + \omega_S).$$

Hence, we have shown that one of these schedules has objective function value within a factor of 16/3 of the optimum, and the schedule found by the algorithm is at least as good. ■

Corollary 4.7 *The linear program* (17) – (21), (28) *is a* $\frac{16}{3}$*-relaxation of* $R|r_{ij}|\sum w_jC_j$.

Deng, Liu, Long, & Xiao (1990) and Awerbuch, Kutten, & Peleg (1992) independently introduced the notion of *network scheduling*, in which parallel machines are connected by a network, each job is located at one given machine at time 0, and cannot be started on another machine until sufficient time elapses to allow the job to be transmitted to its new machine; it is assumed that an unlimited number of jobs can be transmitted over any network link at the same time. This model can be reduced to the problem of scheduling with machine-dependent release dates: if job j originates on machine k, we set r_{ij} to be the time that it takes to transmit a job on machine k to machine i.

We thereby obtain the following corollary.

Corollary 4.8 *There is a* 16/3*-approximation algorithm to minimize the average weighted completion time of a set of jobs scheduled on a network of unrelated machines.*

The best previously known algorithms, due to Awerbuch, Kutten, & Peleg (1992) and Phillips, Stein, & Wein (1994), provided only polylogarithmic performance guarantees .

5 A general on-line framework

In this section, we describe a technique that yields an on-line 4ρ-approximation algorithm to minimize the weighted sum of completion time objective, where ρ depends on the scheduling environment; the setting is on-line in the sense that we are constructing the schedule as time proceeds, and do not know of the existence of job j until time r_j. If one views the role of the LP in Section 4 as assigning the jobs to intervals, this on-line result shows that if one does this assignment in a greedy fashion, then one can still obtain a good performance guarantee. The technique is quite general, and depends only on the existence of an off-line algorithm for the following problem.

THE MAXIMUM SCHEDULED WEIGHT PROBLEM: Given a certain scheduling environment, a deadline D, a set of jobs available at time 0, and a weight for each job, construct a feasible schedule that maximizes the total weight of jobs completed by time D.

We require a *dual ρ-approximation algorithm* for the maximum scheduled weight problem, which produces a schedule of length at most ρD and whose total weight is at least the optimal weight for the deadline D. Dual approximation algorithms were first shown to be useful in the design of traditional approximation algorithms by Hochbaum & Shmoys (1987).

Our technique, which is similar to one used by Blum, Chalasani, Coppersmith, Pulleyblank, Raghavan, and Sudan (1994), is useful in the design of on-line algorithms with performance guarantees that nearly match those obtained by the best off-line approximation algorithms. The required subroutine is a generalization of a subroutine used in the design of approximation algorithms to minimize the length of the schedule. For several of the models considered in this paper, the design of this more general subroutine is a straightforward extension of techniques devised for minimizing the length of the schedule (although we do not yet see how to construct this subroutine for precedence-constrained models). In addition to the simplicity of the approach, the performance guarantees can be quite good. In fact, for $1|r_j|\sum w_jC_j$ and $P|r_j|\sum w_jC_j$, respectively, it leads to $(3+\epsilon)$- and $(4+\epsilon)$-approximation algorithms, which asymptotically match the guarantees proved for these models in Section 4.

This result provides a means to convert off-line scheduling algorithms into on-line algorithms. A result of similar flavor was given for minimizing the length of a schedule by Shmoys, Wein & Williamson (1995), but that result has the advantage that the subroutine required is simply the off-line version of the same problem. In that case, an off-line ρ-approximation algorithm yields an on-line 2ρ-approximation algorithm. We first describe our framework Greedy-Interval and establish its performance guarantee. We then briefly discuss several applications.

The framework Greedy-Interval is also based on dividing the time horizon of possible completion times at geometrically increasing points. Let $\tau_0 = 1$ and $\tau_\ell = 2^{\ell-1}$. The algorithm constructs the schedule iteratively: in iteration $\ell = 1, 2, \ldots$, we wait until time τ_ℓ, and then focus on the set of jobs that have been released by this time, but not yet scheduled, which we denote J_ℓ. We invoke the dual ρ-approximation algorithm for the set of jobs J_ℓ and the deadline $D = \tau_\ell$; notice that, in applying the off-line dual approximation algorithm, we assume that the jobs are available at time 0. The schedule produced by the subroutine is then assigned to run from time $\rho\tau_\ell$ to time $\rho\tau_{\ell+1}$. Let $\widetilde{S}_\ell$ denote the set of jobs scheduled during iteration ℓ. Since $\rho\tau_{\ell+1} - \rho\tau_\ell \geq \rho\tau_\ell$, it is clear that the schedule produced by this algorithm is feasible.

To analyze the performance guarantee of this algorithm, consider a fixed optimal schedule: let L be defined so that each job completes in this schedule by time τ_L, and let S_ℓ^* denote the set of jobs that complete in the ℓth interval, $(\tau_{\ell-1}, \tau_\ell]$, $\ell = 1, \ldots, L$. We will argue that the total weight scheduled by Greedy-Interval dominates the total weight scheduled in the optimal schedule, in the following sense: for each $\ell = 1, \ldots, L$,

$$\sum_{k=1}^{\ell} w(\widetilde{S}_k) \geq \sum_{k=1}^{\ell} w(S_k^*), \tag{30}$$

where $w(S) = \sum_{j \in S} w_j$ for each subset $S \subseteq \{1, \ldots, n\}$. Focus on a particular interval $\ell = 1 \ldots, L$, and consider the set of jobs $S = \cup_{k=1}^{\ell} S_k^* - (\cup_{k=1}^{\ell-1} \widetilde{S}_k)$. Since each job $j \in S$ is completed by τ_ℓ in the optimal schedule, it is clearly released by τ_ℓ, and by definition, it has not been scheduled by Greedy-Interval before τ_ℓ. Hence, $j \in J_\ell$, and so $S \subseteq J_\ell$. Furthermore, S can be scheduled to complete by τ_ℓ (since all jobs in S are completed by τ_ℓ in the optimal schedule). Hence, in iteration ℓ, the dual approximation algorithm must return a set $\widetilde{S}_\ell$ of total weight at least $w(S)$. This implies the dominance property (30). A further consequence of this property is that Greedy-Interval has scheduled all of the jobs by iteration L.

Since the sets S_ℓ^*, $\ell = 1, \ldots, L$, specify an optimal schedule, $\sum_j w_j C_j^* \geq \sum_{\ell=1}^{L} \tau_{\ell-1} w(S_\ell^*)$. (Note that we are using our assumption about the data that no job can complete before time 1.) The schedule produced by Greedy-Interval has total weighted completion time at most

$$\sum_{\ell=1}^{L} \rho\tau_{\ell+1} w(\widetilde{S}_\ell) \leq 4\rho \sum_{\ell=1}^{L} \tau_{\ell-1} w(\widetilde{S}_\ell). \tag{31}$$

However, the dominance property (30), combined with the fact that $\sum_{\ell=1}^{L} w(S_\ell^*) = \sum_{\ell=1}^{L} w(\widetilde{S}_\ell)$, implies this upper bound is at most $4\rho \sum_{\ell=1}^{L} \tau_{\ell-1} w(S_\ell^*)$.

Theorem 5.1 *Given a dual ρ-approximation algorithm for the maximum scheduled weight problem, the framework* Greedy-Interval *yields an on-line 4ρ-approximation algorithm to minimize the total weighted completion time.*

Next we consider how to apply this framework to specific scheduling environments by describing the necessary dual ρ-approximation algorithms. For a single machine and identical parallel machines we provide dual polynomial approximation schemes for the maximum scheduled weight problem; for the unrelated parallel machine and network scheduling environments, we generalize the algorithms of Shmoys and Tardos (1993) and Phillips, Stein & Wein (1994) to provide dual 2-approximation algorithms. These lead to, respectively, $(4+\epsilon)$- and 8-competitive on-line algorithms for these four scheduling problems, where $\epsilon > 0$ is fixed but arbitrarily small.

We first consider applying Greedy-Interval to the problem $1|r_j|\sum w_jC_j$. In this context, the maximum scheduled weight problem is as follows: given a deadline D and a set of jobs J, find a subset of jobs S with $p(S) \leq D$ so as to maximize $w(S)$. This problem is simply the knapsack problem, where the size of the knapsack is D, the size of an "object" j (i.e., job j) is p_j, and the value of that object is w_j, the weight of the associated job. We shall argue that it is straightforward to adapt the fully polynomial approximation scheme of Ibarra & Kim (1975) to yield a dual approximation scheme for this problem. Given $\epsilon > 0$ and a set of n jobs, we round down the processing time of each job to the nearest multiple of $\delta = \epsilon D/n$. More precisely, we set $\bar{D} = \lfloor D/\delta \rfloor$, $\bar{p}_j = \lfloor p_j/\delta \rfloor$, and $\bar{w}_j = w_j$, $j = 1, \ldots, n$. Next we apply a standard dynamic programming algorithm to this rescaled and rounded instance of the knapsack problem; this algorithm runs in $O(n\bar{D}) = O(n^2/\epsilon)$ time. Let $\bar{S}$ denote the optimal solution for the modified instance that is found by this algorithm, and let S^* denote an optimal solution for the unrounded instance. We have that $\delta\bar{p}(S^*) \leq p(S^*) \leq D$; since each $\bar{p}_j$ is integer, this implies that $\bar{p}(S^*) \leq \bar{D}$; that is, S^* is a feasible solution for the modified data. Since $\bar{S}$ is an optimal solution for the modified data, $w(\bar{S}) \geq w(S^*)$. On the other hand,

$$p(\bar{S}) \leq \sum_{j \in \bar{S}} \delta(\bar{p}_j + 1) \leq p(\bar{S}) + n\delta \leq D + \epsilon D = (1+\epsilon)D.$$

This shows that the proposed algorithm is a dual $(1+\epsilon)$-approximation algorithm.

Theorem 5.2 *There is a dual $(1+\epsilon)$-approximation algorithm for the maximum scheduled weight problem in the single-machine scheduling environment.*

Corollary 5.3 *For $1|r_j|\sum w_jC_j$,* Greedy-Interval *yields an on-line $(4+\epsilon)$-approximation algorithm.*

In fact, we can improve on this result by slightly modifying the framework in this setting. Greedy-Interval merely requires that the jobs in $\widetilde{S}_\ell$ be scheduled in the interval $(\rho\tau_\ell, \rho\tau_{\ell+1}]$, without specifying the order in which they should be scheduled. Furthermore, any ordering of the jobs within this interval produces a feasible schedule. Hence, it is most natural to sequence the job in order of non-increasing w_j/p_j ratio. We shall show that this heuristic allows us to prove a stronger performance guarantee.

Consider the completion time of some job in $\widetilde{S}_\ell$. In (31), we use the fact that the completion time of this job is at most $\rho\tau_{\ell+1}$, the upper endpoint of the interval in which these jobs are scheduled. Instead, let this completion time C_j be viewed as $\rho\tau_\ell + \delta_j$. Thus, we can show that $\sum_j w_jC_j$ is at most the sum of $\sum_{\ell=1}^{L} \rho\tau_\ell w(\widetilde{S}_\ell)$ and $\sum_{j=1}^{n} w_j\delta_j$. The first term is exactly half of the upper bound used in (31), and hence is at most $2\rho\sum_{j=1}^{n} w_jC_j^*$. However, since the algorithm is now sequencing the jobs in $\widetilde{S}_\ell$ optimally, we know that

$$\sum_{j \in \widetilde{S}_\ell} w_j\delta_j \leq \sum_{j \in \widetilde{S}_\ell} w_jC_j^*,$$

where C_j^* denote the completion time of job j in some optimal schedule of the entire instance of the problem, $1|r_j|\sum w_jC_j$. Hence, $\sum_{j=1}^{n} w_j\delta_j \leq \sum_{j=1}^{n} w_jC_j^*$, and so the performance guarantee is $2\rho + 1$. From Theorem 5.2 we obtain the following corollary.

Corollary 5.4 *For $1|r_j|\sum w_jC_j$,* Greedy-Interval *yields an on-line $(3+\epsilon)$-approximation algorithm.*

The identical parallel machines environment requires a much more involved algorithm that is basically a modification of the polynomial approximation scheme of Hochbaum and Shmoys (1987) for scheduling identical parallel machines to minimize the makespan. We assume that we are given a set of n jobs with processing times and weights, a deadline D, and $\epsilon > 0$. Our goal is to determine a subset of jobs (and an associated schedule) that can be scheduled on m machines to complete by time $(1+\epsilon)D$, whose total weight is at least as large as that of any subset of jobs that can be scheduled to complete by time D. Without loss of generality we assume that all processing times are at most D, since otherwise they cannot be part of such a schedule. In order to simplify the exposition, we shall merely show that, for any fixed $\epsilon > 0$, there exists such a polynomial-time algorithm; we shall briefly mention techniques for improving the running time at the end.

First we introduce two positive parameters γ and δ, $\gamma < \delta$, whose values will be specified later. We partition the set of jobs into two sets: a job j is *short* if $p_j \leq \delta$ and is *long*, otherwise. For each long job j, we round down its processing time to the nearest multiple of γ. Notice that there are fewer than D/γ distinct processing-time values for long jobs, after rounding; let us assume that there are K distinct values $\tilde{p}_1, \ldots, \tilde{p}_K$.

Next, we introduce the notion of a *machine pattern* that describes a possible assignment of long job sizes to one machine. A machine pattern is specified by the number of long jobs of each processing size; such a pattern can be denoted with a K-tuple, $(n_1, \ldots, n_K)$. We assume that the sum of the rounded processing times in any machine pattern is at most D, and it could be much smaller than D; for example, the empty pattern $(0, \ldots, 0)$ is allowed.

Our strategy will be to focus on choosing a set of m machine patterns, and, given those patterns, generating a schedule with length slightly larger than D whose weight is at least as good as any schedule conforming to that choice of patterns. Then, by trying every possible combination of patterns, we will be guaranteed to find a schedule whose weight is super-optimal and whose length is only slightly degraded. By setting δ and γ judiciously, we will be able to ensure that there is at most a polynomial number of pattern combinations to try, while simultaneously ensuring that, upon inserting the original processing times for the rounded times, we can still achieve a schedule length of $(1+\epsilon)D$. We will call a choice of m patterns *feasible* if there exists a sufficient number of long jobs of each type to actually fill out the patterns.

Lemma 5.5 *Given a feasible choice of patterns, there is a $O(n \log n)$-time algorithm to construct a schedule whose length with respect to the rounded processing times is at most $D + \delta$, such that the sum of the weights of all jobs in the schedule is as large as in any schedule of length at most D that conforms to the choice of patterns.*

Proof: Let us assume that the m chosen patterns together require N_k jobs of type k, for $k = 1, \ldots, K$. For each k, we order the jobs of that type according to non-increasing w_j and we select the first N_k jobs on the list to be in the schedule. Next, let $T = mD - \sum_{k=1}^{K} N_k \tilde{p}_k$, the total amount of machine time left for scheduling the short jobs. Let us assume that the set of short jobs is $\{j_1, \ldots, j_{n'}\}$, ordered so that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_{n'}/p_{n'}$. Consider the index s for which

$$\sum_{j=1}^{n'-1} p_j < T \leq \sum_{j=1}^{n'} p_j$$

or let $s = n'$ if $\sum_{j=1}^{n'} < T$. Consider the partial schedule given by the selected long jobs scheduled according to the machine patterns. We will augment this schedule with the short jobs $j_1, \ldots, j_s$ in such a way that all of these short jobs get scheduled and the overall length of the new schedule (with respect to the rounded processing times) is at most $D + \delta$. This can be done as follows: assign these s short jobs in order, and for each such job j merely identify some machine i which is currently processing jobs of total (rounded) length less than D, and schedule job j on machine i; by an averaging argument, the definition of T and s ensures that there must always exist such a machine i. Consequently, the total processing load assigned to each machine i exceeds D by less than δ, the maximum length of any short job.

We claim that the resulting schedule has total weight at least as large as any schedule for the original problem that conforms to the machine patterns of length at most D. First, among all long jobs we have clearly chosen a set that maximizes the weight of the selected machine patterns. Moreover, T is an upper bound on the total amount of processing time available for scheduling short jobs in any schedule conforming to the chosen machine patterns; since we have chosen the short jobs greedily and have either scheduled all of them or a set of them that has processing time at least T, their total weight must equal or exceed the weight of the short jobs in any schedule with the properties described. Thus, the weight of the constructed schedule is super-optimal relative to the chosen set of machine patterns. The running time of the algorithm is clearly linear once the jobs of every rounded size have been sorted. ■

It remains to show that we can choose δ and γ in a way that any such schedule, when the processing times are unrounded, has length at most $(1+\epsilon)D$, while simultaneously ensuring that the number of possible combinations of m machine patterns is at most polynomial in the size of the input.

The number of possible long jobs that could fit on one machine in a schedule of length D is bounded above by $\lfloor D/\delta \rfloor$, and the total number of job sizes is bounded by $\lfloor D/\gamma \rfloor$; thus an upper bound on the number of distinct machine patterns is $M := (D/\gamma)^{D/\delta}$. To bound the number of ways of selecting a combination of these patterns for m machines, note that each pattern describes at most m machines; therefore the total number pattern combinations is bounded above by m^M. (In fact, complete enumeration is not necessary, and the algorithm can be made much more efficient by employing a simple dynamic programming approach.) In a greedily constructed schedule based on machine patterns, the length of any schedule is bounded by $D+\delta+(D/\delta)\gamma$, where the last term reflects the increase caused by the unrounding of at most D/δ long jobs; we wish to restrict this to be at most $D+\epsilon D$. Values of δ and γ that achieve this bound while making M sufficiently small are $\delta = \epsilon D/2$ and $\gamma = \epsilon^2 D/4$. We observe that these values imply that M is a constant (albeit depending doubly exponentially on $1/\epsilon$), and so there are at most a polynomial number m^M of distinct pattern combinations to try. Since a schedule corresponding to a pattern can be computed in $O(n \log n)$ time, we have obtained the following theorem.

Theorem 5.6 *There is a dual $(1+\epsilon)$-approximation algorithm for the maximum scheduled weight problem in the identical parallel machine scheduling environment.*

Corollary 5.7 *For $P|r_j|\sum w_j C_j$,* Greedy-Interval *yields an on-line $(4+\epsilon)$-approximation algorithm.*

We next turn to the case in which we have a network of unrelated parallel machines: each job j originates on some machine k, and may be transferred to some other machine i through the network; we let r_{ij} denote the earliest time at which job j can begin processing on machine i, which is the sum of its release date and the time required to transfer the job to machine i. The machines themselves are unrelated: each job j requires p_{ij} time units of processing when scheduled on machine i, $i = 1, \ldots, m$.

Phillips, Stein, & Wein (1994) gave an off-line 2-approximation algorithm for minimizing the schedule length in a network of unrelated machines; we will show how to adapt this result to obtain the required subroutine for Greedy-Interval for this scheduling environment.

Consider the maximum scheduled weight problem in this environment: we are given a set of jobs J and a deadline D; for each $j \in J$ and each machine $i = 1, \ldots, m$, we are also given p_{ij}

and r_{ij}, the machine-dependent processing and allow starting times, respectively. Consider the following linear program:

$$\text{maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{ij} \tag{32}$$

subject to

$$\begin{aligned}
\sum_{i=1}^{m} x_{ij} &\leq 1, && j \in J; && (33)\\
\sum_{j\in J} p_{ij}x_{ij} &\leq D, && i = 1,\ldots,m; && (34)\\
x_{ij} &= 0, && \text{if } D < r_{ij} + p_{ij}; && (35)\\
x_{ij} &\geq 0, && i = 1,\ldots,m,\ j \in J. && (36)
\end{aligned}$$

This linear program is a relaxation of the maximum scheduled weight problem: if we consider the optimal schedule for the latter problem, and set $x_{ij} = 1$ whenever job j is scheduled by time D on machine i, then x is a feasible (integer) solution for the linear program (32)-(36). We will derive a dual 2-approximation algorithm by applying Theorem 4.2 to round the optimal solution to this linear program.

Let x denote the optimal solution to the linear program (32)-(36). If we set $c_{ij} = -w_j$, for each $i = 1,\ldots,m$, $j \in J$, and $C = -\sum_{i=1}^{n}\sum_{j\in J} w_j x_{ij}$, then x is a feasible solution to the linear relaxation of the generalized assignment problem, (22)-(25). As a result, we can invoke Theorem 4.2 and round x to obtain an integer solution $\bar{x}$. By this theorem, we know that

$$\sum_{i=1}^{m}\sum_{j\in J} w_j \bar{x}_{ij} \geq \sum_{i=1}^{m}\sum_{j\in J} w_j x_{ij};$$

that is, if we let $\widetilde{S}$ denote the set of jobs j for which some component $\bar{x}_{ij} = 1$, then $w(\widetilde{S})$ is at least the LP optimum, and is consequently at least the optimal value for the maximum scheduled weight problem.

We will show that the set of jobs $\widetilde{S}$ can be scheduled by time $2D$, and hence derive a dual 2-approximation algorithm. By Theorem 4.2, the set $\widetilde{S}$ can be partitioned into $B_i \cup S_i$, $i = 1,\ldots,m$. For each job $j \in B_i \cup S_i$, $x_{ij} = 0$ whenever $r_{ij} + p_{ij} > D$, and so by (26), $\bar{x}_{ij} > 0$ implies that $r_{ij} + p_{ij} \leq D$. This implies that the job j in B_i (if it exists) can be scheduled on machine i from time $D - p_{ij}$ to time D. Furthermore, we know that $\sum_{j\in S_i} p_{ij} \leq D$, and hence all of the jobs in S_i can be scheduled on machine i from time D to $2D$.

Theorem 5.8 *For scheduling on a network of unrelated parallel machines, there is a dual 2-approximation algorithm for the maximum scheduled weight problem.*

Corollary 5.9 *For minimizing $\sum w_j C_j$ in a network of unrelated parallel machines,* Greedy-Interval *yields an on-line 8-approximation algorithm.*

If each job can be transferred between machines without delay, then we have reduced the problem to ordinary unrelated machines, and so we obtain the following corollary.

Corollary 5.10 *For $R|r_j|\sum w_j C_j$,* Greedy-Interval *yields an on-line 8-approximation algorithm.*

Acknowledgments We are grateful to Cor Hurkens, Cindy Phillips, Maurice Queyranne, and Cliff Stein for helpful discussions. Preliminary presentations of this research were given in the conference papers of Hall, Shmoys, & Wein (1996) and Schulz (1996).

References

von Arnim, A., R. Schrader, Y. Wang (1996). The permutahedron of N-sparse posets. Preprint. Dept. of Computer Science, University of Cologne, Cologne, Germany.

von Arnim, A., A. S. Schulz (1994). Facets of the generalized permutahedron of a poset. Preprint 386/1994, Dept. of Mathematics, Technical University of Berlin, Berlin, Germany, to appear in *Discrete Appl. Math.*.

Awerbuch, B., S. Kutten, D. Peleg (1992). Competitive distributed job scheduling. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 571–581.

Balas, E. (1985). On the facial structure of scheduling polyhedra. *Math. Programming Stud.* **24** 179–218.

Blum, A., P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, M. Sudan (1994). The minimum latency problem. *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, 163–171.

Bruno, J. L., E. G. Coffman Jr., R. Sethi (1974). Scheduling independent tasks to reduce mean finishing time. *Comm. ACM* **17** 382–387.

Chakrabarti, S., S. Muthukrishnan (1996). Job scheduling for practical parallel database and scientific applications. *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures,* to appear.

Chakrabarti, S., C. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, J. Wein (1996). Improved scheduling algorithms for minsum criteria. *Proceedings of the 23rd International Colloquium on Automata, Languages and Processing*, to appear.

Chudak, F., D. B. Shmoys (1996). Near-optimal schedules for precedence-constrained scheduling problems on uniformly related parallel machines. In preparation.

Crama, Y., F. C. R. Spieksma (1995). Scheduling jobs of equal length: complexity, facets, and computational results. Balas, E., J. Clausen, eds., *Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science 920*, Springer, Berlin, 277–291.

Deng, X., H. Liu, J. Long, B. Xiao (1990). Deterministic load balancing in computer networks. *Proceedings of 2nd Annual IEEE Symposium on Parallel and Distributed Processing,* 50–57.

Dyer, M. E., L. A. Wolsey (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Appl. Math.* **26** 255–270.

Even, G., J. Naor, S. Rao, B. Schieber (1995). Divide-and-conquer approximation algorithms via spreading metrics. *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, 62–71.

Goemans, M. X. (1996). A supermodular relaxation for scheduling with release dates. *Integer Programming and Combinatorial Optimization*, Proceedings of the 5th International IPCO Conference, *Lecture Notes in Computer Science 1084*, Springer, to appear.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Tech. J.* **45** 1563–1581.

Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** 287–326.

Hall, L. A., D. B. Shmoys, J. Wein (1996). Scheduling to minimize average completion time: off-line and on-line algorithms. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 142–151.

Hochbaum, D. S., D. B. Shmoys (1987). Using dual approximation algorithms for scheduling problems: practical and theoretical results. *J. Assn. Comput. Mach.* **34** 144–162.

Ibarra, O. H., C. E. Kim (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *J. Assn. Comput. Mach.* **22** 463–468.

J. M. Jaffe (1980). Efficient scheduling of tasks without full use of processor resources. *Theoret. Comput. Sci.* **12** 1–17.

Kawaguchi, T., S. Kyan (1986). Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM J. Comput.* **15** 1119–1129.

Lai, T-C. (1995). Earliest completion time and shortest remaining processing time sequencing rules. Preprint, College of Management, National Taiwan University.

Lasserre, J.-B., M. Queyranne (1992). Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. Balas, E., G. Cornuéjols, R. Kannan, eds., *Integer Programming and Combinatorial Optimization*, Proceedings of the 2nd International IPCO Conference, 136–149.

Lenstra, J. K., A. H. G. Rinnooy Kan, P. Brucker (1977). Complexity of machine scheduling problems. *Ann. Discrete Math.* **1** 343–362.

Lenstra, J. K., D. B. Shmoys, É. Tardos (1990). Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming* **46** 259–271.

Lin, J. H., J. S. Vitter (1992). ϵ-approximation with minimum packing constraint violation. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 771–782.

Möhring, R. H., M. W. Schäffter, A. S. Schulz (1996). Scheduling jobs with communication delays: using infeasible solutions for approximation. *Proceedings of the Fourth Annual European Symposium on Algorithms, Lecture Notes in Computer Science,* Springer, to appear.

Munier, A., J. C. König (1993). *A heuristic for a scheduling problem with communication delays.* Internal Report LRI 871, Université de Paris-sud, Orsay, France.

Phillips, C., C. Stein, J. Wein (1994). Task scheduling in networks. *Proceedings of the Fourth Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 824* , Springer, Berlin, 290–301.

Phillips, C., C. Stein, J. Wein (1995). Scheduling jobs that arrive over time. *Proceedings of the Fourth Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 955*, Springer, Berlin, 290–301.

Potts, C. N. (1980). An algorithm for the single machine sequencing problem with precedence constraints. *Math. Programming Stud.* **13** 78–87.

Queyranne, M. (1993). Structure of a simple scheduling polyhedron. *Math. Programming* **58** 263–285.

Queyranne, M., A. S. Schulz (1994). *Polyhedral approaches to machine scheduling.* Preprint No. 408/1994, Department of Mathematics, Technical University of Berlin, Berlin, Germany.

Queyranne, M., A. S. Schulz (1995). Scheduling unit jobs with compatible release dates on parallel machines with nonstationary speeds. Balas, E., J. Clausen, eds., *Integer Programming and Combinatorial Optimization*, Proceedings of the 4th International IPCO Conference, *Lecture Notes in Computer Science 920*, Springer, Berlin, 307–320.

Queyranne, M., Y. Wang (1991a). Single-machine scheduling polyhedra with precedence constraints. *Math. Oper. Res.* **16** 1–20.

Queyranne, M., Y. Wang (1991b). *A cutting plane procedure for precedence-constrained single machine scheduling.* Working paper, Faculty of Commerce, University of British Columbia, Vancouver, British Columbia.

Ravi, R., A. Agrawal, P. Klein (1991). Ordering problems approximated: single-processsor scheduling and interval graph completion. *Proceedings of the 1991 International Colloquium on Automata, Languages and Processing* , *Lecture Notes in Computer Science 510*, Springer, Berlin, 751–762.

Schulz, A. S. (1995). *Scheduling and Polytopes.* PhD thesis, Technical University of Berlin, Berlin, Germany.

Schulz, A. S. (1996). Scheduling to minimize total weighted completion time: performance guarantees of LP–based heuristics and lower bounds. Cunningham, W. H., S. T. McCormick, M. Queyranne, eds., *Integer Programming and Combinatorial Optimization*, Proceedings of the 5th International IPCO Conference, *Lecture Notes in Computer Science 1084*, Springer, to appear.

Shmoys, D. B., É. Tardos (1993). An approximation algorithm for the generalized assignment problem. *Math. Programming* **62** 461–474.

Shmoys, D. B., J. Wein, D. P. Williamson (1995). Scheduling parallel machines on-line. *SIAM J. Comput.* **24** 1313-1331.

Smith, W. (1956). Various optimizers for single-stage production. *Naval Res. Logist. Quart.* **3** 59–66.

Sousa, J. P., L. A. Wolsey (1992). A time-indexed formulation of non-preemptive single-machine scheduling problems. *Math. Programming* **54** 353–367.

Trick, M. (1994). Scheduling multiple variable speed machines. *Operations Research* **42** 234–248.

Van den Akker, J. M. (1994). *LP-based solution methods for single-machine scheduling problems.* PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Van den Akker, J. M., C. P. M. Van Hoesel, M. W. P. Savelsbergh (1993). *Facet-inducing inequalities for single-machine scheduling problems.* Memorandum COSOR 93-27, Eindhoven University of Technology, Eindhoven, The Netherlands.

Van den Akker, J. M., C. A. J. Hurkens, M. W. P. Savelsbergh (1995). *A time-indexed formulation for single-machine scheduling problems: branch and cut.* Preprint.

Wolsey, L. A. (1985). *Mixed integer programming formulations for production planning and scheduling problems.* Invited talk at the 12th International Symposium on Mathematical Programming, MIT, Cambridge.

Wolsey, L. A. (1990). Formulating single machine scheduling problems with precedence constraints. Gabsewicz, J. J., J.-F. Richard, L. A. Wolsey, eds., *Economic Decision Making: Games, Econometrics and Optimisation, Contributions in Honour of Jacques Dreze.* North-Holland, Amsterdam, 473–484.