Contents

4

Weighted sum of completion times

Eugene L. Lawler
University of California, Berkeley

Maurice Queyranne
University of British Columbia

Andreas S. Schulz
Massachusetts Institute of Technology

David B. Shmoys
Cornell University

Minimization of the mean completion time has always been an intuitively appealing objective. Although the origins of the *Shortest Processing Time* (*SPT*) *rule* are unknown, we do know that W. E. Smith, in one of the first publications in scheduling theory, showed that the *ratio rule*, a generalization of SPT, solves $1||\sum w_j C_j$. We also know that Smith pointed out in 1956 that the ratio rule begs for abstraction in the form of a "preference order."

During the 1960's and 70's, Smith's results on $1||\sum w_j C_j$ were extended to apply to precedence constraints of various kinds, first to parallel chains, then to rooted trees, and then to series-parallel networks, with rooted trees as a special case. During the same period, it was found that the preference order concept applies to a variety of sequencing problems, including the least cost fault detection problem, the two-machine permutation flow shop problem, and problems with maximum cumulative cost and total weighted discounted cost as their objectives. Moreover, it was observed

that the same $O(n \log n)$ algorithm solves any of these problems with series-parallel precedence constraints, provided the preference order has the property of applying to "sequences" and not simply to "jobs." This led to an elegant theory for dealing with precedence constraints in sequencing problems with a variety of scheduling objectives, of which total weighted completion time is but a prototypical example.

Later, the influence of polyhedral theory on combinatorial optimization made its impact on our understanding of a number of variations of the problem $1||\sum w_j C_j$. This understanding led not only to new perspectives on these older results, but also to the development of an extensive literature of approximation algorithms for these problems. In this chapter, we will highlight both of these threads of understanding.

4.1. Smith's ratio rule

By applying Smith's ratio rule, the problem $1||\sum w_j C_j$ can be solved with nothing more than a simple $O(n \log n)$ sort of the jobs by their ratios w_j/p_j (throughout this chapter, we assume all job weights are nonnegative and all processing times positive, i.e., $w_j \geq 0$, and $p_j > 0$, $j = 1, \ldots, n$).

Theorem 4.1. *A sequence is optimal for* $1||\sum w_j C_j$ *if and only if it places the jobs in order of nonincreasing ratios* w_j/p_j.

Proof. We first prove that having nonincreasing ratios is a necessary condition for a sequence to be optimal. Let π be a sequence in which the jobs are not in ratio order. Then, in π, there is a job i that immediately precedes a job j and yet $w_i/p_i < w_j/p_j$. If job j completes at time C_j, then job i completes at time $C_j - p_j$. If we interchange these two jobs, this affects only *their* completion times, not those of other jobs. The result is a strict decrease in total cost, by

$$\begin{aligned} [w_i(C_j - p_j) + w_j C_j] - [w_j(C_j - p_i) + w_i C_j] &= w_j p_i - w_i p_j \\ &= p_i p_j \left(\frac{w_j}{p_j} - \frac{w_i}{p_i} \right) > 0, \end{aligned}$$

from which it follows that π is not optimal.

Conversely, we now prove that having nonincreasing ratios is a sufficient condition for a sequence to be optimal. Let π be a sequence in which the jobs are in ratio order and let π^* be an optimal sequence. If $\pi \neq \pi^*$, then in π^* there is a job i immediately preceding a job j, where j precedes i in π. But then $w_j/p_j \geq w_i/p_i$ (because in π jobs are in order of nonincreasing ratios) and $w_i/p_i \geq w_j/p_j$ (by the first part of this proof because π^* is optimal) and, therefore, $w_i/p_i = w_j/p_j$. Interchanging the jobs in π^* creates a new sequence of equal cost. A finite number of such interchanges converts π^* to π, demonstrating that π is optimal. □

The ratio rule immediately specializes to the celebrated *Shortest Processing Time* or *SPT* rule.

Corollary 4.2. *A sequence is optimal for* $1||\sum C_j$ *if and only if it places the jobs in order of nondecreasing processing times* p_j.

The SPT rule is often applied to more complicated problems than $1||\sum C_j$, sometimes without much theoretical support for its performance. Although no additional proof of the SPT rule is needed, variations of the following proof turn out to be quite useful for parallel machine problems, as we shall see in Chapter **??**. Suppose the jobs are executed in the order $1,2,\ldots,n$. Then we have

$$\begin{aligned} C_1 &= p_1, \\ C_2 &= p_1 + p_2, \\ C_3 &= p_1 + p_2 + p_3, \\ &\cdots \\ C_n &= p_1 + p_2 + p_3 + \cdots + p_n, \end{aligned}$$

giving us

$$\sum_{j=1}^{n} C_j = np_1 + (n-1)p_2 + (n-2)p_3 + \cdots + 2p_{n-1} + p_n. \tag{4.1}$$

This means that the problem of minimizing $\sum_{j=1}^{n} C_j$ is equivalent to the problem of assigning the coefficients $1,2,\ldots,n$ to the processing times p_j in such a way that the weighted sum (4.1) is minimized. This is accomplished by assigning the coefficient 1 to the largest p_j, the coefficient 2 to the next-largest p_j, etc., and the coefficient n to the smallest p_j, as can be verified by an interchange argument similar to that used in the proof of Theorem 4.1.

Smith's ratio rule produces an optimal schedule for the *nonpreemptive* problem $1||\sum w_jC_j$. The reader may wonder if the total cost could be further reduced by allowing preemption, or inserting idle time before all jobs are complete. The answer is "No," even for the most favorable model of preemption, whereby an interrupted job may be resumed at any date without any cost or time penalty. In fact, the same negative answer, "There is no advantage to preemption," applies to a broad class of single-machine scheduling problems. If we let $C = (C_1, C_2, \ldots, C_n)$ denote the completion time vector of a schedule, we say that an objective function $f(C)$ is *monotone* if $C \leq C'$ (i.e., $C_j \leq C'_j$ for each $j = 1,\ldots,n$) implies $f(C) \leq f(C')$. We leave the proof of the following theorem as an exercise, since it can be proved easily with the machinery of the previous chapter.

Theorem 4.3. *There is no advantage to preemption or idle time for the single-machine problem* $1|prec, \overline{d}_j, pmtn|f(C)$ *whenever* f *is monotone.*

Of course, this theorem can be applied to the problem $1|pmtn|\sum w_jC_j$, since we assume throughout this book that each $w_j \geq 0$, $j = 1,\ldots,n$. Hence, Theorem 4.3 implies that Smith's ratio rule is also optimal when preemption and/or idle time are allowed.

Many results in single-machine scheduling, starting with Smith's ratio rule, have a simple and intuitive geometric justification using *two-dimensional Gantt charts*. As in ordinary Gantt charts, the horizontal axis in a 2D Gantt chart represents time. The vertical axis represents total weight: at date $t \geq 0$, we plot the total weight $W(t)$ of all the jobs that have not yet been completed by date t. In a nonpreemptive schedule, if job j is in process at date t, we may also plot $W(t) - w_j$, so the two horizontal lines $W(t)$ and $W(t) - w_j$ delimit a rectangle of length p_j and height w_j, somewhat analogue to the rectangles that represent jobs in ordinary Gantt charts; see Figure **??** (a).

For any nonpreemptive schedule, the area under the curve $W(t)$ is equal to $\sum_{j=1}^{n} w_j C_j$. This can be seen by identifying each term $w_j C_j$ with the area of the horizontal slabs in Figure **??** (b). From this, it is clear that there is no advantage to inserting idle time.

We can draw a descending diagonal in each rectangle representing a job. The *slope* of job j is the ratio $\rho(j) = w_j/p_j$; it is just the negative of the slope of its diagonal. Smith's ratio rule states that the area under the $W(t)$ curve is minimized when the jobs are sequenced with largest slopes first, that is, when the piecewise linear continuous curve $\overline{W}(t)$ defined by the slopes of the rectangles, is made convex. The adjacent pairwise argument used in the proof of Theorem 4.1 is visualized in Figure **??**.

One interpretation of the $\sum w_j C_j$ objective is to consider w_j to be the holding cost (euros per time unit) of a resource needed for the processing of each job j. We assume that we have an initial inventory of the resource that is exactly equal to the amount needed to process all the given jobs; so $W(0) = \sum_{j=1}^{n} w_j$ is the initial total holding cost. It is convenient to measure the resource inventory level at any time t in monetary units (euros), and to identify it with the corresponding total holding cost $W(t)$. If all units of the resource used by job j are consumed instantly at the completion of the job, then the inventory level $W(t)$ follows the piecewise constant curve plotted in the 2D Gantt chart.

In many situations, however, the resource is consumed at a constant rate $\rho(j) = w_j/p_j$ units per time unit during the processing of job j. In that case, the inventory level follows the curve $\overline{W}(t)$, as in Figure **??**(a). The total inventory cost, $\int_0^{+\infty} \overline{W}(t)\,dt$ is the sum of the areas of the resulting horizontal trapezoids. Each trapezoid has the same area as a rectangle with length $(C_j - p_j + C_j)/2 = C_j - p_j/2$ and height w_j; see Figure **??**(b). Alternatively, one can instead define the *mean busy time* M_j of each job j as the midpoint of its nonpreemptive processing: $[(C_j - p_j) + C_j]/2 = C_j - p_j/2$. Thus, we are also interested in finding schedules to minimize the total weighted mean busy time, $\sum w_j M_j$. In the nonpreemptive setting, since the difference between this new objective and $\sum w_j C_j$ is equal to $\sum w_j p_j/2$ (a constant) for any feasible schedule, the two objective functions lead to equivalent optimization problems. Throughout this chapter, we shall see that this alternative perspective significantly improves our understanding.

In the presence of release dates, idle time may be necessary, and we will see in Section 4.10 that there *is* an advantage to preemption, for both the $\sum w_j C_j$ and

$\sum w_j M_j$ objectives. It will also be shown there that the preemptive weighted mean busy time problem $1|r_j, pmtn|\sum w_j M_j$ is solvable to optimality by a very simple algorithm, whereas the preemptive weighted completion time problem $1|r_j, pmtn|\sum w_j C_j$ is NP-hard.

Finally, whereas the nonpreemptive *optimization* problems with $\sum w_j C_j$ and $\sum w_j M_j$ objectives are equivalent (because of the constant difference $\frac{1}{2}\sum w_j p_j$), the corresponding *approximation* problems are *not* equivalent. In fact, an α-approximation algorithm for the nonpreemptive $\sum w_j M_j$ objective is also an α-approximation algorithm for the nonpreemptive $\sum w_j C_j$ objective, but the converse is not necessarily true.

Exercises

4.1. Prove Theorem 4.3.

4.2. Consider $1|\bar{d}_j|\sum C_j$. Assume that there exists a sequence in which all jobs meet their deadlines. Show that the following algorithm produces an optimal feasible sequence:

> From among all jobs j that are eligible to be sequenced last, i.e., are such that $\bar{d}_j \geq p_1 + \cdots + p_n$, put the job last which has the longest processing time. Repeat this procedure with the remaining $n-1$ jobs.

4.3. Use 2D Gantt charts to show that an instance of $1|prec|\sum w_j C_j$ with processing times p_j, weights w_j, and precedence constraints $\rightarrow$ is equivalent to an instance with the same set of jobs, $p'_j = w_j$, $w'_j = p_j$ for each job j, and $j \rightarrow' k$ if and only if $k \rightarrow j$. In particular, a sequence is optimal for the original instance if and only if the reverse sequence is optimal for the new instance, and both have the same objective function value.

4.2. Preference orders on jobs

A very general formulation of optimal sequencing problems is as follows: Given a real-valued function f that assigns a cost $f(\pi)$ to each permutation π of a set N of n jobs, find a permutation π^* of N such that

$$f(\pi^*) = \min\{f(\pi) : \pi \text{ is a permutation of } N\}.$$

If we know nothing at all about the structure of the function f, then we have no alternative but to evaluate f for each of the $n!$ permutations of N. This occurs if f is given to us by a "black box" subroutine, to which we can submit a permutation π, and from which we only receive the value $f(\pi)$ in return. However, we usually know quite a bit about the structure of the function f. We use knowledge of f to solve problems like $1||L_{\max}$ and $1||\sum w_j C_j$ by Jackson's EDD rule and Smith's ratio rule, respectively. Perhaps other problems lend themselves to solution by similar rules. But, if so, what do we mean by "similar?" What is it we need to know about f in order to infer such rules?

Throughout this section and Section 4.3, we will use slightly different notation and will represent a permutation or sequence as the concatenation of disjoint subsequences, e.g., a permutation π of n jobs may be represented as $\pi = (u,s,t,v)$, where each of the n jobs appears in exactly one of the subsequences u,s,t,v. A single job corresponds to a sequence of length one. In the case of $1|\ |L_{\max}$, we observed that $d_i \le d_j$ implies that $f(u,i,j,v) \le f(u,j,i,v)$ [Jackson's rule]. In the proof of Theorem 4.1 we showed that $w_i/p_i \ge w_j/p_j$ implies $f(u,i,j,v) \le f(u,j,i,v)$ [Smith's rule].

Definition 4.4. *A transitive and complete relation $\le_f$ on N is said to be a* preference order on jobs, *relative to objective function f, if it satisfies the* adjacent pairwise interchange property on jobs, *i.e.,*

$$i \le_f j \text{ implies that } f(u,i,j,v) \le f(u,j,i,v), \tag{4.2}$$

for all jobs i,j and all subsequences u,v.

Recall that a relation $\le_f$ is *transitive* if, for each triple $i,j,k \in N$, $i \le_f j$ and $j \le_f k$ imply that $i \le_f k$. A relation $\le_f$ is *complete* if, for each pair $i,j \in N$, either $i \le_f j$ or $j \le_f i$. A relation that is both transitive and complete is sometimes called a *quasi total order*. Such a relation induces a linear ordering of equivalence classes, where i and j are in the same equivalence class if and only if both $i \le_f j$ and $j \le_f i$, in which case we may choose to write $i \equiv_f j$. If $i \le_f j$, but it is not the case that $j \le_f i$, we may write $i <_f j$.

Theorem 4.5. *Given a preference order $\le_f$ on N, an optimal sequence can be found by sorting jobs according to $\le_f$, with $O(n \log n)$ comparisons of jobs with respect to $\le_f$.*

Proof. Let π be any sequence consistent with the preference order, and let π^* be an optimal permutation. If π^* differs from π, then π^* is of the form (u,j,i,v), for some pair i,j of jobs, where i precedes j in π and hence $i \le_f j$. From (4.2) it follows that $f(u,i,j,v) \le f(\pi^*)$, and hence (u,i,j,v) is also optimal. A finite number of such interchanges transforms π^* into π and shows that π is optimal. □

As already observed, Smith's ratio rule for $1|\ |\sum w_j C_j$ and Jackson's EDD rule for $1|\ |L_{\max}$ give rise to special cases of preference orders. Below we consider some other examples of sequencing problems for which there are preference orders on jobs.

Total Weighted Discounted Completion Time.

Sometimes jobs must be scheduled over a time period so long that inflation and interest charges must be taken into account. Suppose that we will be paid w_j dollars upon the completion of job j. The present value of one dollar at time t in the future is $\exp(-rt)$, where $r > 0$ is a fixed discount rate. Hence the present value of completing

job j at time t is $w_j \exp(-rt)$. It follows that to maximize the present value of the payments we will receive we should minimize $\sum_j f_j(C_j)$ with respect to cost functions of the form

$$f_j(t) = -w_j \exp(-rt).$$

We assert (Exercise 4.6) that an optimal sequence is obtained by sequencing jobs in nondecreasing order of the ratios $w_j/[1-\exp(rp_j)]$.

Least Expected Cost Fault Detection.

A system consisting of n components is to be inspected by testing the components one at a time until either one fails (the system is found to be defective) or until all the components pass their tests (the system passes inspection). The cost of testing component j is c_j and the probability that it will pass its test is q_j. Tests are assumed to be statistically independent. Hence if the components are tested in the order $1,2,\ldots,n$, the probability that it will be necessary to test component j is

$$Q_j = q_1 \cdot q_2 \cdot \ldots \cdot q_{j-1},$$

where by convention $Q_1 = 1$. The expected cost of testing is then $\sum_j c_j Q_j$. We assert (Exercise 4.7) that it is optimal to test the components in nondecreasing order of the ratios $c_j/(1-q_j)$.

Weighted Monotone Cost Density.

Suppose for each job j we have

$$f_j(t) = w_j \int_{t-p_j}^{t} g(u)\,du,$$

where g is a nondecreasing "cost density" function, and we want to minimize $\sum_j f_j(C_j)$. We assert (Exercise 4.9) that an optimal permutation is obtained by placing the jobs in order of nonincreasing w_j.

Exercises

4.4. Suppose someone gives you a function $f(\pi)$ in the form of a "black box" subroutine. But she also assures you that $f(\pi)$ is actually the weighted sum of completion times. You have total ignorance of the values of the parameters (i.e., the p_j's and w_j's) of the n jobs in your problem instance. But you can still find an optimal sequence with $O(n \log n)$ calls on the subroutine. How?

4.5. If $\leq_f$ is a preference order on jobs, does it follow that $i \leq_f j$ implies $f(u,i,v,j,w) \leq f(u,j,v,i,w)$, for all i,j,u,v,w? Prove or disprove.

4.6. Prove the validity of the ratio rule asserted for the total weighted discounted completion time problem.

4.7. Prove the validity of the ratio rule asserted for the least expected cost fault detection problem.

4.8. Any instance of the least expected cost fault detection problem can be transformed into an equivalent instance of the total weighted discounted completion time problem as follows. Let $r = 1$. For component j with parameters c_j and q_j, create a job j with parameters $p_j = -\ln q_j$ and $w_j = -c_j/q_j$. Provide a similar transformation in the reverse direction, i.e., from the discounted completion time problem to the fault detection problem, showing that the two problems are equivalent.
4.9. In the case of a weighted monotone cost density function, prove that an optimal sequence is obtained by placing jobs in nonincreasing order of w_j.
4.10. Show that total weighted completion time is a special case of weighted monotone cost density.
4.11. Show that total weighted *discounted* completion time is a special case of weighted monotone cost density.

4.3. Preference orders on sequences and series-parallel precedence constraints

Let Π be the set of all *feasible* permutations of a set $N = \{1, \ldots, n\}$ of n jobs, and $f : \Pi \to \mathbb{R}$ be a cost function. The *constrained optimal sequencing problem* is to find a permutation $\pi^* \in \Pi$ such that $f(\pi^*) = \min\{f(\pi) : \pi \in \Pi\}$.

If the structure of the function f is unknown, then there is no alternative but to evaluate the cost of each feasible permutation in Π. And although the number of feasible permutations may be much smaller than $n!$, this number may still be hopelessly large. In practice, we are likely to know quite a bit about both the function f and the set Π. However, in order to successfully apply preference orders to constrained problems, our preference orders must satisfy stronger properties than before.

Let $\mathcal{N}$ denote the set of all *subpermutations* of the jobs N, i.e., the set of all sequences that can be formed from subsets of N. We shall call elements of $\mathcal{N}$ *sequences*, or *compound jobs*.

Definition 4.6. *A transitive and complete relation $\leq_f$ on $\mathcal{N}$ is said to be a* preference order on sequences, *relative to objective function f, if it satisfies the following* adjacent pairwise interchange property on sequences*:*

$$s \leq_f t \quad \textit{implies} \quad f(u,s,t,v) \leq f(u,t,s,v) \textit{ for all disjoint sequences } u,s,t \textit{ and } v.$$

Thus, by assumption, $\leq_f$ is a complete preorder or, in simpler terms, a total order with possible ties between sequences. We denote by $<_f$ the corresponding strict preorder, that is, $s <_f t$ when $s \leq_f t$ and $t \not\leq_f s$.

Some sequencing problems, like $1|\,|L_{\max}$, admit a preference order on jobs, but not on sequences. (See Exercise 4.13.) However, all of the other problems cited in Section 4.2 do admit preference orders on sequences, with the exception of problems with weighted monotone cost density functions. (See Exercise 4.14.) In particular, in the case of $1|\,|\sum w_j C_j$, the appropriate extension is $s \leq_f t$ if and only if $w(s)/p(s) \geq$

$w(t)/s(t)$, where we extend our usual notation (slightly) to let $p(s)$ and $w(s)$ denote, respectively, the sum of the processing times and weights of the jobs in a sequence s. (See Exercise 4.12.)

In this section, we are primarily interested in the case when the set Π of feasible permutations is specified by *precedence constraints*. We write $i \to j$ to denote the precedence constraint that job i must appear before job j in any feasible permutation π. One strategy for dealing with precedence constraints is to ignore them and simply sort the jobs by preference order. If we are lucky and the resulting sequence turns out to be feasible, then we are done. This follows from the fact that, by making all jobs independent, we have solved a relaxation of the original problem. If an optimal schedule for this relaxation happens to be feasible with respect to the precedence constraints, then it must also be optimal with respect to these constraints.

If the permutation π obtained by sorting jobs by preference order is not feasible, it is due to the *collision* of one or more pairs of jobs i, j where $i \to j$ and $j \leq_f i$. If it happens that j and i are consecutive in π and $i \leq_f j$, the collision can be resolved immediately by interchanging j and i in π. Even if this is not the case, it may be possible to do something about the collision.

Lemma 4.7. *Let i and j be a pair of jobs such that $i \to j$ and $j \leq_f i$. Suppose that, for each $k \in N$ distinct from i and j, either (i) $k \to i$; or (ii) $j \to k$; or (iii) k is unrelated by the precedence constraints to i and also unrelated to j. Then there exists an optimal feasible permutation in which i immediately precedes j.*

Proof. Let $\pi^* = (t, i, u, j, v)$ be an optimal feasible permutation. If u is empty, then we are done. Hence, assume that u is not empty. Since each job k in u is unrelated to both i and j, it follows that the precedence constraints are not violated by interchanging i and u, or by interchanging u and j. It must be the case that either $u \leq_f i$ or $j \leq_f u$, else we would have $i <_f u <_f j$, contradicting the hypothesis that $j \leq_f i$. Hence at least one of the two interchanges results in an optimal feasible permutation in which i immediately precedes j. □

When the hypotheses of the lemma are found to apply to a colliding pair of jobs i and j, the pair can be replaced by the sequence (i, j), with (i, j) inheriting all of the precedence constraints of i and j, e.g., if $j \to k$ then $(i, j) \to k$. The sequence (i, j) can then be treated as a single job, or compound job, for the purpose of reapplying the lemma.

Precedence constraints that consist of *parallel chains* have the nice property that application of Lemma 4.7 is guaranteed to yield a set of sequences that is free of collisions. To see this, one need only note that if any pair of jobs is in collision, there is a colliding pair of jobs i, j that are adjacent in a chain. But then i and j satisfy the hypotheses of the lemma and can be replaced by a single sequence. With no more than $n-1$ repeated applications of the lemma, a collision-free family of sets of sequences can be obtained, in the sense of the following definition. For this definition, let an ordering $(s_1, \ldots, s_\ell)$ of sequences be *consistent with* $\leq_f$ if $s_u \leq_f s_v$ for all $1 \leq u < v \leq \ell$; and *feasible with respect to the precedence constraints* if for

any precedence relation $i \to j$, we have that $i \in s_u$, and $j \in s_v$, for some $u \leq v$.

Definition 4.8. *Let X be a finite index set. A family $\mathcal{S} = \{S(x) : x \in X\}$ of sets $S(x)$ of sequences is said to be* collision-free, *with respect to given precedence constraints and preference order $\leq_f$ on sequences, if*

1. *each job in N is contained in exactly one of the sequences in $\cup_{x \in X} S(x)$; and*
2. *there exists an optimal feasible permutation of the jobs in N in which the jobs in each sequence $s \in \cup_{x \in X} S(x)$ appear consecutively; and*
3. *any ordering of the sequences in each set $S(x)$ which is consistent with $\leq_f$ is also feasible with respect to the precedence constraints.*

In effect, Lemma 4.7 enables us to transform a problem instance with parallel-chains precedence constraints into an instance, consisting of sequences (or compound jobs) $s \in N'$, for which the family $\mathcal{S} = \{ N'\}$ is collision-free. Hence the resulting instance can be dealt with as if it was unconstrained and an optimal feasible permutation is obtained by simply sorting all these sequences in preference order $\leq_f$.

Theorem 4.9. *Let the precedence constraints form parallel chains. Given a preference order $\leq_f$ on $\mathcal{N}$, an optimal feasible permutation of N can be found with $O(n \log n)$ comparisons of sequences with respect to $\leq_f$.*

Series-parallel partial orders are defined recursively as follows:

any partial order $(N, \to)$, where N is a singleton, is series-parallel.

Let $(N_1, \to)$ and $(N_2, \to)$ be disjoint partial orders (i.e., $N_1 \cap N_2 = \emptyset$) that are series-parallel. A partial order $(N_1 \cup N_2, \to)$ is also series-parallel, when relations between jobs in N_1 and jobs in N_2 are determined by either

series composition, in which each job i in N_1 precedes each job j in N_2, i.e.,
$i \to j$ for all $i \in N_1$ and $j \in N_2$,

or

parallel composition, in which the jobs in N_1 and N_2 are unrelated, i.e.,
$i \not\to j$ and $j \not\to i$ for all $i \in N_1$ and $j \in N_2$.

(Relations between pairs of jobs, both of which are in N_1 or in N_2, are unaffected.)

The structure of series-parallel precedence constraints is represented by a *composition* (or *decomposition*) tree in which each leaf of the tree is identified with a job and each internal node of the tree corresponds to a series or a parallel composition operation, and is accordingly labeled either "S" or "P." The left and right children of an S-node are respectively identified with the subsets N_1, N_2 of the series composition operation. The same is true of the children of a P-node, except that the left-right ordering of its children is immaterial.

Some examples of series-parallel precedence constraints and their composition trees are indicated in Figure **??**. Note that parallel chains, in-trees, out-trees, and forests of in-trees and out-trees, are all special cases of series-parallel constraints. The smallest non-series-parallel partial order is the "Z" digraph shown in Figure **??**. In fact, a partial order fails to be series-parallel if and only if it contains four elements in a "Z" relation.

There are efficient algorithms for testing if precedence constraints are series-parallel. In particular, a digraph G with m arcs and n nodes can be tested in $O(m+n)$ time to determine whether the partial order it induces is series-parallel. If the partial order is series-parallel, the algorithm "parses" it and returns a decomposition tree. If the partial order is not series-parallel, the algorithm returns a "Z," proving that it is not.

Given a decomposition tree for series-parallel precedence constraints and a preference order $\leq_f$ on sequences, a plausible strategy for finding an optimal feasible permutation is to work from the leaves of the tree toward the root, dealing with the subproblem at an internal node only after the subproblems at its children have been dealt with. At each node x of the tree we propose to obtain a collision-free family $\mathcal{S}$ of sets of sequences, in which a set $S(x) \in \mathcal{S}$ contains sequences which contain every job corresponding to node x, and no other job. The algorithm will maintain the following two invariants after each node has been dealt with:

(1) the current family $\mathcal{S}$ of sets of sequences is collison-free; and

(2) the permutation obtained by sorting all sequences in $\mathcal{S}$ in preference order $\leq_f$ is optimal for the relaxed problem restricted only by the precedence constraints defined by the nodes dealt with so far.

Thus, at the root node x_0 the set $\mathcal{S}$ will consist of a single set $S(x_0)$, and an optimal feasible permutation will be obtained by simply sorting all these sequences in preference order $\leq_f$.

We initialize $\mathcal{S} = \{S(j) : j \in N\}$ where each $S(j)$ consists of the single sequence j. Thus all leaves have been dealt with, and invariants (1) and, by Theorem 4.5, (2) are verified. At a P-node x with children y and z, all that is necessary is to replace in $\mathcal{S}$ the sets $S(y)$ and $S(z)$ with the set $S(x) = S(y) \cup S(z)$, since none of the sequences in $S(x)$ collide. If $\mathcal{S}$ is collision-free before this replacement, then it also is collision-free afterwards. Moreover, invariant (2) continues to hold because we have not added any new precedence constraint. At an S-node, things are a bit more complicated. However, collisions can be resolved by repeated application of the following generalization of Lemma 4.7.

Lemma 4.10. *Let $\mathcal{S}$ be a collision-free family of sets of sequences. Let $S(a)$ and $S(b)$ be two sets in $\mathcal{S}$ such that, for every $s \in S(a)$ and $t \in S(b)$ there exist jobs $i \in s$ and $j \in t$ such that $i \to j$. Let α be a $\leq_f$-maximal sequence in $S(a)$ and β be a $\leq_f$-minimal sequence in $S(b)$, such that $\beta \leq_f \alpha$. Assume that, for each $k \in N$ which is not in any sequence in $S(a) \cup S(b)$, either*

(i) $k \to i$ *for some job* i *in every sequence in* $S(a)$*; or*

(ii) $j \to k$ *for some job* j *in every sequence in* $S(b)$*; or*

(iii) k *is unrelated by the precedence constraints to every job in every sequence in* $S(a) \cup S(b)$.

Then there exists an optimal feasible permutation in which every sequence in $\mathcal{S}$ *is consecutive and sequence* α *immediately precedes sequence* β.

Proof. Since $\mathcal{S}$ is collision-free, let $\pi^* = (t, \alpha, u, \beta, v)$ be an optimal feasible permutation in which each sequence appearing in $\mathcal{S}$ is consecutive, and for which the number $|u|$ of jobs between α and β is as small as possible. If $|u| = 0$ we are done; hence assume that $|u| \geq 1$. If any sequence in $S(a)$ appears in u, let s be the first such sequence and let $u = (u', s, u'')$ with no sequence in u' contained in $S(a)$. If u' is empty, then by the $\leq_f$-maximality of α in $S(a)$ we may interchange s and α, obtaining an optimal permutation with fewer than $|u|$ jobs between α and β, a contradiction to u being of minimum size. If u' is nonempty, u' precedes $s \in S(a)$ in the feasible permutation π^*, and no sequence in u' can be contained in $S(b)$ either. Therefore, by condition (iii) of the lemma, we can feasibly interchange α with u', and u' with s. The minimality of $|u|$ and the optimality of π^* imply that $\alpha <_f u'$ and $u' \leq_f s$; hence $\alpha <_f s$, a contradiction to the $\leq_f$-maximality of α in $S(a)$. Therefore, no sequence in $S(a)$ can appear in u. A symmetric argument implies that no sequence in $S(b)$ can appear in u. But now we may feasibly interchange α with u, and u with β, so we must have $\alpha <_f u <_f \beta$, a contradiction with the assumption $\beta \leq_f \alpha$. □

Let x be an S-node and y and z be its left and right children. Let ℓ be a $\leq_f$-maximal sequence in $S(y)$ and r be a $\leq_f$-minimal sequence in $S(z)$. If $\ell <_f r$ then we can simply merge $S(y)$ and $S(z)$ into a single set $S(x) = S(y) \cup S(z)$ and the resulting family is collision-free. Moreover, invariant (2) continues to hold. Otherwise, $r \leq_f \ell$ and we can apply Lemma 4.10: there exists an optimal feasible permutation in which every sequence in $\mathcal{S}$ is consecutive and sequence ℓ immediately precedes sequence r. Thus we concatenate the two sequences ℓ and r into a single sequence $\lambda = (\ell, r)$. If $s <_f \lambda$ for all $s \in S(y) \setminus \{\ell\}$ and $\lambda <_f s$ for all $s \in S(z) \setminus \{r\}$, then we can replace $S(y)$ and $S(z)$ with $S(x) = (S(y) \setminus \{\ell\}) \cup (S(z) \setminus \{r\}) \cup \{\lambda\}$, and the resulting family is again collision-free. Otherwise, we form a collision-free family $\mathcal{S}'$ by replacing $S(y)$ and $S(z)$ with the three sets $S'(y) = S(y) \setminus \{\ell\}$, $\{\lambda\}$, and $S'(z) = S(z) \setminus \{r\}$. However, we aim to replace in turn these three sets with a single set $S(x)$ that comprises all of the corresponding jobs: for the resulting family to be collision-free, $S(x)$ must satisfy Property 3 of Definition 4.8.

If $\lambda \leq_f s$ for some $s \in S'(y)$ then we may again apply Lemma 4.10 with $S'(y)$ as $S(a)$ and $\{\lambda\}$ as $S(b)$: we conclude that there exists an optimal feasible permutation in which every sequence in $\mathcal{S}'$ is consecutive and a $\leq_f$-maximal sequence α in $S'(y)$ immediately precedes sequence λ. Thus replacing λ with (α, λ) and $S'(y)$ with $S'(y) \setminus \{\alpha\}$ maintains the invariant that the resulting family is collision-free. Similarly, if $s \leq_f \lambda$ for some $s \in S'(z)$ then we may remove from $S'(z)$ a $\leq_f$-minimal sequence β

and merge it with λ (so (λ,β) now replaces λ). We may repeat these two operations until we obtain a collision-free family $\mathcal{S}'$ such that all sequences $s \in S'(y)$ satisfy $s <_f \lambda$ and all sequences $t \in S'(z)$ satisfy $\lambda <_f t$. We may now replace these three sets by their union $S(x) = S'(y) \cup \{\lambda\} \cup S'(z)$ and obtain the desired collision-free family. At this point, we have dealt with the S-node x and may proceed to another tree node.

To summarize the preceding discussion, we now give a pseudo-code for the recursive computation of a set of sequences $S(x)$ at a node x of a series-parallel decomposition tree, so as to maintain a collision-free family of sets of sequences. We suppose that the sets of sequences at each node are recorded in a priority queue supporting the operations of `findmin`, `findmax`, `deletemin`, `deletemax`, and `merge`. The `findmin` operation returns a sequence which is $\leq_f$-minimal in the set, while `deletemin` returns such a sequence and also deletes it from the set; similarly for `findmax` and `deletemax`. When L is empty the `findmax` and `deletemax` operations return a dummy job such that $\max L <_f j$, for all jobs j; similarly, when R is empty, $\min R >_f j$, for all jobs j. Finally, `merge` forms the union of two sets of sequences.

$S(x)$:

Case (x is a leaf): **return** $S := \{j\}$, where j is the job at x;
Case (x is a P node): **return** $S :=$ `merge`$(S(\text{left}(x)), S(\text{right}(x)))$;
Case (x is an S node):
  $L := S(\text{left}(x))$; $R := S(\text{right}(x))$;
  if `findmax`$(L) <_f$ `findmin`(R) **then return** $S :=$ `merge`(L,R);
    else
    $s :=$ (`deletemax`(L), `deletemin`(R)); * concatenation *
    while (`findmax`$(L) \geq_f s$) **or** (`findmin`$(R) \leq_f s$)
      if `findmax`$(L) \geq_f s$ **then** $s :=$ (`deletemax`(L), s);
                else $s :=$ (s, `deletemin`(R));
      endwhile
    $S :=$ `merge`(L, R);
    return $S :=$ `merge`$(S, \{s\})$;
    endif.

Each of the priority queue operations can be implemented to run in $O(\log n)$ time, and each operation is performed no more than $O(n)$ times. The final sort of the strings obtained at the root of the tree requires no more than $O(n \log n)$ comparisons, and invariant (2) implies that this is an optimal sequence for the entire instance. Hence we have the following result.

Theorem 4.11. *Let series-parallel precedence constraints be specified by a decomposition tree. Given a preference order $\leq_f$ on sequences, an optimal feasible permutation of N can be found with $O(n \log n)$ comparisons of sequences with respect to $\leq_f$, and at most $O(n \log n)$ time for other operations.*

In particular, a variety of single-machine scheduling with precedence constraints can be solved in $O(n \log n)$ time; see the exercises below for examples.

Exercises

4.12. Prove that the preference order on sequences defined for the $\sum w_j C_j$ criterion is correct.

4.13. Show that $1|\ |L_{\max}$ does not admit a preference order on sequences.

4.14. Show that there is no preference order on sequences when f is determined by weighted monotone cost density functions.

4.15. Find a preference order on sequences for each of the following problems:

(a) Total weighted discounted completion time problem.

(b) Least cost fault detection problem.

4.16. Provide pseudocode for obtaining a collision-free set in the case of parallel-chains precedence constraints. You should be able to achieve $O(n)$ running time.

4.4. NP-hardness of further constrained min-sum problems

Unfortunately, it is relatively easy to move from the world of polynomial-time solvable problems to the world of NP-hard ones. In this section, we will present three NP-hardness results: $1|r_j|\sum C_j$, $1|r_j, pmtn|\sum w_j C_j$, and $1|prec|\sum w_j C_j$.

Theorem 4.12. *The problem $1|r_j|\sum C_j$ is NP-hard in the strong sense.*

Proof. We will show that the 3-PARTITION problem (see Chapter **??**) reduces to the decision version of $1|r_j|\sum C_j$. Consider an instance of 3-PARTITION, consisting of positive integers $a_1, \ldots, a_{3t}, b$, with $b/4 < a_j < b/2$ for all j and $\sum_j a_j = tb$. Recall that this is a yes-instance if and only if the index set $T = \{1, \ldots, 3t\}$ can be partitioned into t mutually disjoint 3-element subsets $S_1, \ldots, S_t$ with $\sum_{j \in S_i} a_j = b$ for $i = 1, \ldots, t$. We will define an instance of $1|r_j|\sum C_j$ and an integer Z such that there exists a schedule of value $\sum C_j \leq Z$ if and only if the instance of 3-PARTITION is a yes-instance.

The scheduling instance has three types of jobs. First, for each $j \in T$, there is a job J_j with release date 0 and processing time a_j. Second, for each $i \in \{1, \ldots, t-1\}$, there are v jobs $K_k^{(i)}$ with release date ib and processing time 0 $(k = 1, \ldots, v)$. Finally, there are w jobs L_ℓ with release date tb and processing time 1 $(\ell = 1, \ldots, w)$. The values of Z, v and w will be defined later.

Suppose we have a yes-instance of 3-PARTITION, and consider the following schedule (see Figure **??**). The three jobs J_j with $j \in S_i$ are processed in the interval $[(i-1)b, ib]$, for $i = 1, \ldots, t$; the sum of their completion times is bounded from above by

$$Z_J = \sum_{i=1}^{t} 3ib = \frac{3}{2} t(t+1)b.$$

The jobs $K_k^{(i)}$ are scheduled to start at their release dates; their total completion time is equal to

$$Z_K = \sum_{i=1}^{t-1} vib = \frac{1}{2}vt(t-1)b.$$

The jobs L_ℓ are processed consecutively in the interval $[tb, tb+w]$; their total completion time is given by

$$Z_L = \sum_{\ell=1}^{w} (tb+\ell) = wtb + \frac{1}{2}w(w+1).$$

We now define $v = Z_J$, $w = Z_J + Z_K$, and $Z = Z_J + Z_K + Z_L$. The schedule corresponding to the yes-instance of 3-PARTITION has a value $\sum C_j \leq Z$.

Conversely, consider a schedule satisfying $\sum C_j \leq Z$. We claim that, in any such schedule, all J_j are completed by time tb. If this is not the case, then at least one J_j as well as all L_ℓ finish after tb, so that

$$\sum C_j > \sum_{\ell=1}^{w+1} (tb+\ell) = Z_L + tb + w + 1 > Z.$$

It may be assumed that, for any i, all $K_k^{(i)}$ $(k = 1, \ldots, v)$ are processed consecutively; this follows from a straighforward interchange argument. We now also claim that, if $\sum C_j \leq Z$, then all $K_k^{(i)}$ start at their release dates. Otherwise, at least v of these jobs are delayed by one time unit, so that

$$\sum C_j > Z_K + v + Z_L = Z.$$

We conclude that, in any schedule of value $\sum C_j \leq Z$, all jobs J_j are processed in the interval $[0, tb]$, interrupted by zero-time jobs at each point in time ib $(i = 1, \ldots, t-1)$. Hence, in each interval $[(i-1)b, ib]$, three jobs J_j are processed for a total duration of b time units. This implies that we have a yes-instance of 3-PARTITION.

Note that the number of jobs is proportional to b, so that the correctness of the reduction essentially depends upon the *strong* NP-completeness of 3-PARTITION. Finally, let us mention that one can modify the reduction such that all jobs have a positive length (Exercise 4.17). □

We did mention earlier that, in the presence of release dates, there can be advantage to preemption. In fact, for $1|r_j|\sum C_j$ there is; in addition, $1|r_j, pmtn|\sum C_j$ can be solved efficiently by the *Shortest Remaining Processing Time* (*SRPT*) rule; see Theorem 4.25 below. However, the preemptive problem is NP-hard if jobs can have different weights.

Theorem 4.13. *The problem* $1|r_j, pmtn|\sum w_j C_j$ *is NP-hard in the strong sense.*

Proof. We will show that this problem is NP-hard in the ordinary sense, by a reduction from the PARTITION problem. It is not hard to extend this to a reduction from the 3-PARTITION problem, which implies NP-hardness in the strong sense; see Exercise 4.18.

An instance of PARTITION consists of positive integers $a_1,\ldots,a_t,b$ with $\sum_j a_j = 2b$. It is a yes-instance if and only if the index set $T=\{1,\ldots,t\}$ includes a subset S with $\sum_{j\in S} a_j = b$.

Given an instance of PARTITION, we define t jobs j, $(j=1,\ldots,t)$, with $r_j=0$ and $p_j=w_j=a_j$. For these jobs, any nonpreemptive schedule without idle time is optimal: all $r_j=0$, so that there is no advantage to preemption, and all $w_j/p_j=1$, so that the ratio rule may choose any job order. The value of such a schedule is given by

$$Z_J = \sum_{1\le j\le k\le t} a_j a_k.$$

(This can easily seen by considering 2-dimensional Gantt charts; see Section 4.5 below.) We define one more job, K, with release date b, processing time 1, and weight 2.

Suppose we have a yes-instance of PARTITION, and consider the following schedule. All jobs j with $j\in S$ are processed in the interval $[0,b]$, job K is scheduled in $[b,b+1]$, and the remaining jobs are processed in $[b+1,2b+1]$ (see Figure **??**(a)). Since these latter jobs are all delayed by one time unit, their contribution to $\sum w_jC_j$ increases by the sum of their weights, which is equal to the sum of their processing times. The value of this schedule is therefore equal to

$$Z = (Z_J+b)+2(b+1) = Z_J+3b+2.$$

Now consider any feasible schedule, and suppose that job K finishes at time $b+1+c$, for some $c\ge 0$. It may be assumed that there is no idle time in the interval $[0,2b+1]$ and that job K starts at time $b+c$. For the jobs that finish after job K, let d be the total amount of processing done on them prior to K, for some $d\ge 0$ (see Figure **??**(b)). Again, the contribution of these jobs to $\sum w_jC_j$ increases by the sum of their weights, which is now equal to $d+b-c$. Hence, the value of this schedule is given by

$$(Z_J+d+b-c)+2(b+c+1) = Z_J+3b+c+d+2 = Z+c+d.$$

It follows that a schedule has value $\sum w_jC_j \le Z$ if and only if $c=d=0$; that is, job K is processed in $[b,b+1]$ and each other job is entirely processed either before K or after K. Such a schedule exists if and only if we have a yes-instance of PARTITION. □

Note that, alternatively, we could have given a very high weight to job K, thereby immediately fixing its starting time to b. The above technique, however, can be extended to yield similar and simple reductions to a number of related problems, including $1|d_j|\sum w_jC_j$, $1||\sum w_jT_j$, and $P||\sum w_jC_j$.

We show the strong NP-hardness of the problem $1|\,prec|\sum w_jC_j$ by a two-step reduction from the LINEAR ARRANGEMENT problem, which is defined as follows: assign the vertices of an undirected graph $G=(V,E)$ to integer points on the real line so that the sum of edge lengths is minimized. More formally, given $G=(V,E)$ and a positive integer Z, is there a one-to-one mapping f of V to $\{1,\ldots,n\}$ such that $\sum_{\{u,v\}\in E}|f(u)-f(v)|\leq Z$? This problem is NP-complete, and we first reduce it to a version of $1|\,prec|\sum w_jC_j$ where jobs can have zero processing times and negative weights.

Lemma 4.14. *The optimal linear arrangement problem is polynomially reducible to the precedence-constrained single-machine scheduling problem with nonnegative processing times and arbitrary weights.*

Proof. Given an instance $G=(V,E)$ and Z of the LINEAR ARRANGEMENT problem, we introduce a job for each vertex and one for each edge. Let d_v be the degree of vertex v in G. For each vertex $v\in V$, the corresponding job v has weight $w_v=-d_v$ and processing time $p_v=1$. For each edge $\{u,v\}\in E$, the corresponding job $\{u,v\}$ has weight $w_{\{u,v\}}=2$ and processing time $p_{\{u,v\}}=0$. Moreover, each edge job $\{u,v\}$ has exactly two predecessors, namely u and v.

Suppose that we have a mapping f of V to $\{1,\ldots,n\}$ such that $\sum_{\{u,v\}\in E}|f(u)-f(v)|\leq Z$. Schedule the vertex jobs in the same order; i.e., v is scheduled in position $f(v)$ among all vertex jobs. An edge job $\{u,v\}$ is scheduled immediately after its predecessor that completes later. The total weighted completion time of the resulting schedule is given by

$$\begin{aligned}
&\sum_{v\in V} w_vC_v+\sum_{\{u,v\}\in E} w_{\{u,v\}}C_{\{u,v\}}\\
&=-\sum_{v\in V} d_vf(v)+\sum_{\{u,v\}\in E} 2\max\{f(u),f(v)\}\\
&=\sum_{\{u,v\}\in E}\big(2\max\{f(u),f(v)\}-f(u)-f(v)\big)\\
&=\sum_{\{u,v\}\in E}|f(u)-f(v)|\ .
\end{aligned}$$

It is therefore at most Z. On the other hand, if there is a schedule of total weighted completion time at most Z, we may assume, without loss of generality, that each edge job $\{u,v\}$ is processed as soon as both jobs u and v are completed. Then, the same calculation as before implies that the order of the vertex jobs defines a solution of the given instance of the LINEAR ARRANGEMENT problem with value at most Z. □

Given an instance of $1|\,prec|\sum w_jC_j$, adding a constant to the weight of every job not only changes the value of each schedule, but also can change the relative order of schedules with respect to their objective function values. However, if the constant is only added to the weights of jobs with processing time 1, and all other jobs have

zero processing time, then the relative order of schedules is maintained. Let $d_{\max}$ be the maximal degree of a vertex in G. We can then add $d_{\max}$ to the weight of each vertex job in the proof of Lemma 4.14 to see that the scheduling problem remains NP-hard for instances with nonnegative weights.

One can also easily modify the reduction so that all jobs have positive processing times (Exercise 4.19). We have completed the proof of the following theorem.

Theorem 4.15. *The problem* $1|prec|\sum w_jC_j$ *is NP-hard in the strong sense.*

Exercises

4.17. Modify the reduction in the proof of Theorem 4.12 so that all jobs in the resulting instance of $1|r_j|\sum C_j$ have strictly positive processing times.

4.18. Give a reduction from the 3-PARTITION problem to show that the problem $1|r_j, pmtn|\sum w_jC_j$ is indeed NP-hard in the *strong* sense.

4.19. Modify the reduction in the proof of Theorem 4.15 so that all jobs in the resulting instance of $1|prec|\sum w_jC_j$ have *unit* processing time.

4.20. Modify the reduction in the proof of Theorem 4.15 so that all jobs in the resulting instance of $1|prec|\sum C_j$ have unit weight and processing times 0 or 1.

4.5. The ratio rule via linear programming

Linear programming methods have played a significant role in the development of combinatorial optimization; scheduling is no exception to this rule. We next present another proof of correctness for Smith's ratio rule using the tools of linear programming.

Let us draw a 2-dimensional Gantt chart where the "resource consumed" during the execution of every job j is in fact the amount of processing, or *work* done on the job. The vertical axis may thus be interpreted as the *remaining work*; see Figure **??** for an illustration. Note that all jobs now have the same slope $\rho(j) = 1$. Recall that the mean busy time of a job j in a nonpreemptive schedule is $M_j = C_j - p_j/2$.

Consider any subset $A \subseteq N$, where $N = \{1,\ldots,n\}$ is the set of all jobs to be processed. From the 2D Gantt chart, or from Smith's ratio rule, it follows that a schedule minimizes $\sum_{j\in A} p_jM_j$ if and only if all jobs in A start at time zero and no idle time is incurred until the processing of all jobs in A is complete. Jobs not in A have a zero weight in the objective $\sum_{j\in A} p_jM_j$, and can be processed at any time after all jobs in A are complete. The resulting minimum objective value is the area of the triangle below job set A, that is, $\frac{1}{2}\,p(A)^2 = \frac{1}{2}\left(\sum_{j\in A} p_j\right)^2$. This shows that for any subset $A \subseteq N$, the following so-called *parallel inequality*

$$\sum_{j\in A} p_jM_j \geq \frac{1}{2}\,p(A)^2 \tag{4.3}$$

holds for any feasible schedule in which no job can start before time 0 and the ma-

chine can process at most one job at a time. These inequalities are valid for *any* scheduling problem, whenever N is a set of jobs or operations to be processed on a machine with unit speed and unit capacity.

The parallel inequalities may be written, using completion times instead of mean busy times, in the equivalent form

$$\sum_{j\in A} p_j C_j \geq \frac{1}{2}\, p(A)^2 + \frac{1}{2} \sum_{j\in A} p_j^2 \tag{4.4}$$

and may be interpreted as enforcing the requirement that no set of jobs can be completed too early on a machine with limited processing capacity.

The parallel inequalities (4.3) or (4.4), and variations or strengthenings thereof to take into account additional constraints or characteristics (such as precedence constraints, release dates, different processing speeds on different machines, etc.) play an important role in defining relaxations and approximation algorithms for more complicated scheduling problems.

In this light, we start by considering the problem that we know how to solve: $1|\,|\sum w_j M_j$. Suppose, without loss of generality, that we have reindexed the jobs so that $\rho(1) \geq \rho(2) \geq \cdots \geq \rho(n)$. Consider the linear program

$$\min \left\{ \sum_{j\in N} w_j M_j : \sum_{j\in A} p_j M_j \geq \frac{1}{2}\, p(A)^2 \text{ for all } A \subseteq N \right\}. \tag{4.5}$$

We can show that the ratio rule is optimal by proving that if we schedule the jobs in the order $1,2,\ldots,n$, this feasible solution is an optimal solution to this linear program. Consequently, this schedule must also be optimal for $1|\,|\sum w_j M_j$ (and also $1|\,|\sum w_j C_j$).

Let $\overline{M}_j$, $j = 1,\ldots,n$, denote the mean busy times of the jobs scheduled in the order $1,2,\ldots,n$. We use linear programming duality to prove the optimality of this schedule. That is, we exhibit a feasible dual solution that satisfies the complementary slackness conditions with $\overline{M}_j$, $j = 1,\ldots,n$. The dual linear program has nonnegative variables y_A, for each $A \subseteq N$, and has a constraint for each $j = 1,\ldots,n$, $\sum_{A:j\in A} p_j y_A = w_j$, or equivalently, that $\sum_{A:j\in A} y_A = \rho(j)$.

The complementary slackness conditions for this pair of linear programs amount to $y_A > 0$ only if the corresponding parallel inequality is satisfied with equality. If we keep this in mind, then we can deduce a feasible solution for the dual as follows: the parallel inequalities corresponding to the sets $A_j = \{1,\ldots,j\}$ hold with equality. Since n is only in A_n, we set $y_{A_n} = \rho(n)$; working backwards, we see then that $y_{A_{n-1}} = \rho(n-1) - \rho(n)$, and in general, $y_{A_j} = \rho(j) - \rho(j+1)$, $j = 1,\ldots,n-1$. The sorting of the jobs ensures that these values are all nonnegative. Moreover, $y_A = 0$ for all other $A \subseteq N$. Hence, we have feasible primal and dual solutions satisfying the complementary slackness conditions, and so they are optimal for their respective linear programs.

This proof of the correctness of the ratio rule actually says something much

stronger, and truly remarkable. What it says, at its core, is that the parallel inequalities completely describe the feasible set of mean busy time vectors; more precisely, what we have shown is, in essence, that the feasible region of the linear program in M_j variables is exactly the convex hull of vectors that correspond to mean busy times of feasible schedules. Of course, the same holds true for completion time vectors.

Exercises

4.21. If one would want to solve the linear program (4.5) by standard linear programming methods, one would have to deal with the exponential number of constraints. Because of the equivalence of optimization and separation (see Chapter **??**), it suffices to solve the separation problem associated with the parallel inequalities $\sum_{j\in A} p_j M_j \geq \frac{1}{2}\, p(A)^2$, $A \subseteq N$. Given a vector $M^* \in \mathbb{Q}^N$, the separation problem is to decide whether M^* satisfies all parallel inequalities and, if not, to produce a parallel inequality that is violated by M^*.

(a) Defining the violation $v(A) = \frac{1}{2}\, p(A)^2 - \sum_{j\in A} p_j M_j^*$, $A \subseteq N$, compute $v(A \cup \{k\}) - v(A)$ for $k \notin A$, and $v(A) - v(A \setminus \{j\})$ for $j \in A$.

(b) Assume, without loss of generality, that $M_1^* \leq M_2^* \leq \cdots \leq M_n^*$. Using (a), show that a parallel inequality most violated by M^*, if any, can be found among one of the consecutive sets $\{1\}, \{1,2\}, \ldots, \{1,2,\ldots,n\}$.

It follows that (4.5) can be solved in polynomial time. This should be hardly surprising, since it is solved by Smith's ratio rule. The real interest of the separation algorithm is in solving linear programming relaxations of more complicated scheduling problems, as will be seen later in this and other chapters.

4.22. Show that the mean busy times M_j, $j = 1, \ldots, n$, of any feasible schedule for $1|r_j|\sum w_j C_j$ satisfy the inequalities $\sum_{j\in A} p_j M_j \geq (\min_{j\in A} r_j + \frac{1}{2}\, p(A))p(A)$.

4.23. Solve the separation problem associated with the inequalities in Exercise 4.22.

4.6. Approximation algorithms for $1|prec|\sum w_j C_j$

In light of the NP-hardness of $1|prec|\sum w_j C_j$, it is natural to consider approximation algorithms. In Section 4.5, we used a linear program to give an alternative proof of optimality for Smith's ratio rule. Looking at this proof from a different angle, we see that sequencing jobs in nondecreasing order of their completion times in the solution to the linear program results in an optimal schedule for $1|\,|\sum w_j C_j$. We now use the same idea to create a schedule whose cost is within a factor 2 of that of an optimum for the strongly NP-hard problem $1|prec|\sum w_j C_j$.

While the constraints of the linear program for $1|\,|\sum w_j C_j$ consisted of the parallel inequalities (4.4) only, we add the following inequalities to make sure that the resulting schedule is consistent with the precedence constraints:

$$C_j - C_i \geq p_j \text{ for all pairs } i \to j. \tag{4.6}$$

Let C_j, $j = 1,\ldots,n$, be a solution of the linear program to minimize $\sum_{j\in N} w_jC_j$ subject to the parallel inequalities (4.4) for all $A \subseteq N$ and the precedence constraints (4.6). Without loss of generality, we can reindex the jobs so that $C_1 \leq C_2 \leq \cdots \leq C_n$. Because of (4.6), $i < j$ whenever $i \to j$. In contrast to the case $1|\ |\sum w_jC_j$, the values C_j do in general not correspond to job completion times in an actual schedule, even when they correspond to a basic feasible solution; see Exercise 4.25. However, we can easily construct a feasible schedule by sequencing the jobs in the order $1,2,\ldots,n$. The completion times $\overline{C}_j$ of this schedule are $\overline{C}_1 = p_1, \overline{C}_2 = p_1 + p_2, \ldots, \overline{C}_n = p_1 + p_2 + \cdots + p_n$. Note that this schedule does not violate any precedence constraints. We will now show that the total weighted completion time of this schedule is at most twice that of an optimal schedule.

Consider an arbitrary, but fixed job k. Recall that $(C_1, C_2, \ldots, C_n)$ satisfies the parallel inequalities (4.4); in particular, for $A = \{1,2,\ldots,k\}$,

$$\sum_{j=1}^{k} p_jC_j \geq \frac{1}{2}\Big(\sum_{j=1}^{k} p_j\Big)^2 \ ,$$

where we have even dropped a nonnegative term from the right-hand side of inequality (4.4). Because of the ordering of jobs, the left-hand side of this inequality is bounded from above by $C_k \sum_{j=1}^{k} p_j$. We therefore obtain

$$\overline{C}_k = \sum_{j=1}^{k} p_j \leq 2C_k \ .$$

Thus, $\sum_{j=1}^{n} w_j\overline{C}_j \leq 2\sum_{j=1}^{n} w_jC_j$; given *any* feasible solution of the linear program to minimize $\sum_{j\in N} w_jC_j$ subject to all parallel inequalities and precedence constraints, we can construct a schedule with total weighted completion time at most twice its value. If we start with an optimal solution C_j, $j = 1,2,\ldots,n$, of the linear program, then $\sum_{j\in N} w_jC_j$ is a lower bound on the cost of an optimal schedule; we have proved the following theorem.

Theorem 4.16. *Scheduling jobs in nondecreasing order of completion times in an optimal solution to the linear program* $\min\{\sum_{j=1}^{n} w_jC_j : C$ *satisfies* (4.4) *and* (4.6)$\}$ *is a* 2*-approximation algorithm for the problem* $1|prec|\sum w_jC_j$.

Actually, we still have to argue that the linear program in C_j variables can be solved in polynomial time, because there are exponentially many parallel inequalities. While it follows from Exercise 4.21 that this is indeed the case, we here take a different route. Suppose there is another linear programming relaxation that only has polynomially many variables and constraints, so it can obviously be solved in polynomial time. Furthermore, suppose that every solution of the new linear program can be mapped to a feasible solution of the original linear program with no change of its objective function value. Then we could compute an optimal solution to the new linear program, map it to a solution of the old linear program, and apply the previous algorithm. We now describe such a linear program.

One way to derive another linear programming relaxation of the scheduling problem $1|prec|\sum w_jC_j$ is to formulate it as an integer program, and then relax the integrality constraints. Solving $1|prec|\sum w_jC_j$ is equivalent to determining, for each pair i and j of jobs, whether i precedes j in the solution, or not. For any pair $i \neq j$ of jobs, we introduce the variable $\delta_{ij} \in \{0,1\}$: $\delta_{ij} = 1$ indicates that job i precedes job j, and $\delta_{ij} = 0$ indicates otherwise. Therefore,

$$\delta_{ij} + \delta_{ji} = 1 \text{ for all pairs } i, j \in N, i \neq j. \tag{4.7}$$

We can obviously represent every feasible schedule by a $0/1$-vector $\delta = (\delta_{ij})_{i \neq j}$ of this type; moreover, the resulting $0/1$-vectors satisfy further inequalities. The precedence constraints imply that

$$\delta_{ij} = 1 \text{ for all } i \rightarrow j. \tag{4.8}$$

In addition, when job j precedes job k, and k precedes i, then j also precedes job i. This translates into the following inequalities:

$$\delta_{jk} + \delta_{ki} - \delta_{ji} \leq 1 \text{ for all triples } i, j, k \in N, i \neq j \neq k \neq i. \tag{4.9}$$

These inequalities are called *transitivity constraints*. On the other hand, every $0/1$-vector δ that satisfies the transitivity constraints (4.9), the precedence constraints (4.8), and (4.7) represents a feasible job sequence. Given such δ, the completion time C_j of job j in the corresponding schedule is

$$C_j = \sum_{i \neq j} p_i \delta_{ij} + p_j. \tag{4.10}$$

So $1|prec|\sum w_jC_j$ is equivalent to minimizing $\sum_{j=1}^{n} \sum_{i \neq j} w_j p_i \delta_{ij} + \sum_{j=1}^{n} w_j p_j$ subject to the constraints (4.7) – (4.9), and $\delta_{ij} \in \{0,1\}$ for all $i \neq j$. If we replace $\delta_{ij} \in \{0,1\}$ by

$$\delta_{ij} \geq 0 \text{ for all } i \neq j, \tag{4.11}$$

we obtain a linear program of polynomial size whose optimal value is a lower bound on the value of an optimal schedule. It remains to show that, if δ is a solution to this linear program, then the vector defined by (4.10), for $j = 1, 2, \ldots, n$, is a solution to the linear program in C_j variables.

Let us consider the precedence constraints (4.6) first. Assume $i \rightarrow j$; then $\delta_{ij} = 1$, and (4.10) gives

$$C_j = \sum_{k \neq i,j} p_k \delta_{kj} + p_i + p_j \ .$$

Because of (4.7), $\delta_{ji} = 0$, and thus (4.10) yields

$$C_i = \sum_{k \neq i,j} p_k \delta_{ki} + p_i \ .$$

Let k be a job with $k \neq i, j$. Since $\delta_{ji} = 0$, the transitivity constraint (4.9) for this triple implies that $\delta_{kj} = 1 - \delta_{jk} \geq \delta_{ki}$. Hence, $C_j \geq C_i + p_j$.

We now turn our attention to the parallel inequalities (4.4). Fix $A \subseteq N$. Then,

$$\begin{aligned}
\sum_{j\in A} p_j C_j &= \sum_{j\in A} p_j \Big(\sum_{k\neq j} p_k \delta_{kj} + p_j \Big) = \sum_{\substack{j\in A, k\in N \\ j\neq k}} p_j p_k \delta_{kj} + \sum_{j\in A} p_j^2 \\
&\geq \sum_{\substack{j,k\in A \\ j\neq k}} p_j p_k \delta_{kj} + \sum_{j\in A} p_j^2 = \sum_{\substack{j,k\in A \\ j<k}} p_j p_k (\delta_{jk} + \delta_{kj}) + \sum_{j\in A} p_j^2 \\
&= \frac{1}{2} p(A)^2 + \frac{1}{2} \sum_{j\in A} p_j^2 \ .
\end{aligned}$$

The last equality follows from (4.7).

Corollary 4.17. *Let δ be an optimal solution to the linear program* $\min\{\sum_{k\neq j} w_k p_j \delta_{jk} : \delta \geq 0$ *satisfies* (4.7) – (4.9)$\}$. *Define C_j according to* (4.10), *for $j = 1, 2, \ldots, n$, and sequence the jobs in nondecreasing order of C_j values. Then, the total weighted completion time of the resulting schedule is at most twice that of an optimal schedule.*

The analysis of this 2-approximation algorithm is tight; i.e., there exist instances for which the resulting schedule has cost close to twice that of an optimal schedule. There are also instances for which the value of an optimal schedule is essentially twice that of the linear program; see Exercise 4.27.

The technique of scheduling jobs in order of their "completion times" in a linear programming relaxation of the problem is of use in more complicated problems than $1|prec|\sum w_jC_j$, including parallel machine and open shop problems. We shall see some of the resulting approximation algorithms in Chapters **??**.

Interestingly, no approximation algorithm with a performance guarantee strictly less than 2 is known for the problem $1|prec|\sum w_jC_j$. However, there is a generic way of designing a 2-approximation algorithm that does not require solving a linear program, which we describe next. For that, we need the concept of Sidney decomposition.

Exercises

4.24. For an instance of $1||\sum w_jC_j$, consider the linear program $\min\{\sum_{i,j,i\neq j} w_j p_i \delta_{ij} : \delta_{ij} + \delta_{ji} = 1, \delta_{ij} \geq 0$ for all $i \neq j\}$. Show that the optimal value of this linear program plus $\sum_j w_j p_j$ is equal to the value of an optimal schedule. How can one use this fact to give another proof of the optimality of Smith's ratio rule?

4.25. Consider the following instance of $1|prec|\sum w_jC_j$: there are three jobs of unit length each, job 1 precedes jobs 2 and 3, and $w_1 = 0$, $w_2 = 1$, and $w_3 = 1$. Show that $C_1 = 4/3$, $C_2 = 7/3$, and $C_3 = 7/3$ is an optimal basic feasible solution of the linear program $\min\{\sum_j w_jC_j : C$ satisfies (4.4) and (4.6)$\}$.

4.26. Show that the analysis that led to Theorem 4.16 is tight:

(a) Describe a family of instances for which the ratio of the objective function value returned by the algorithm to that of an optimal schedule converges to 2

with increasing values of n.

(b) Give a family of instances for which the ratio of the cost of an optimal schedule to that of an optimal solution to the linear program in completion time variables converges to 2 with increasing values of n.

4.27. Show that the analysis that led to Corollary 4.17 is tight.

4.7. Sidney Decompositions

Recall that any optimal sequence for $1||\sum w_jC_j$ orders jobs according to nonincreasing ratios w_j/p_j. In the presence of precedence constraints, we already saw that this ordering can lead to collisions that cannot always be resolved. The Sidney decomposition offers one way to extend Smith's ratio rule to $1|prec|\sum w_jC_j$ by considering ratios of subsets. A subset $I \subseteq N$ that contains all its predecessors under the precedence constraints is said to be an *initial set* of N. That is, I is an initial set if $j \in I$ and $i \rightarrow j$ imply $i \in I$. Note that a subset $I \subseteq N$ is an initial set of N if and only if there is a feasible sequence that schedules all jobs in I before all remaining jobs. A *ratio-maximal initial set* I is an initial set of N with maximum ratio $\rho(I) = w(I)/p(I)$. Here, as before, $w(I) = \sum_{j\in I} w_j$ and $p(I) = \sum_{j\in I} p_j$.

Theorem 4.18. *Let I^* be a ratio-maximal initial set of N. There exists an optimal sequence of N that schedules the jobs in I^* before all remaining jobs.*

Before we prove this theorem, we discuss its implications and related aspects. First, note that an initial set is a closure in the directed graph defined by reversing all precedence constraints. A ratio-maximal initial set can therefore be computed in polynomial time; see Chapter **??**. Second, we can apply Theorem 4.18 to the set $N \setminus I^*$ of remaining jobs. If we repeat this, then we eventually obtain a decomposition $(I_1, I_2, \ldots, I_k)$ of N such that $I_i \cap I_j = \emptyset$, for all $1 \leq i < j \leq k$, $I_1 \cup I_2 \cup \ldots \cup I_k = N$, and I_j is a ratio-maximal initial set of $N \setminus (I_1 \cup \ldots \cup I_{j-1})$. A decomposition of this kind is known as a *Sidney decomposition*. Note that each set I_j, for $j = 1, 2, \ldots, k$, is ratio-maximal for itself; i.e., $\rho(I) \leq \rho(I_j)$ for all initial sets $I \subseteq I_j$. A sequence is *consistent with* a given Sidney decomposition if it first schedules the jobs in I_1, then the jobs in I_2, and so forth, until it eventually schedules all jobs in I_k. We can now reformulate Theorem 4.18 so that it encompasses Smith's ratio rule (Theorem 4.1).

Corollary 4.19. *A sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_k)$ is optimal for $1|prec|\sum w_jC_j$ if π is consistent with a Sidney decomposition $(I_1, I_2, \ldots, I_k)$ of N, and each subsequence π_j is optimal for I_j, $j = 1, \ldots, k$.*

Corollary 4.19 implies that it suffices to consider approximation algorithms that are consistent with a Sidney decomposition. In particular, if we had a 2-approximation algorithm for instances whose ground sets are ratio-maximal, then we could apply this algorithm to all sets I_j in a Sidney decomposition, concatenate

the resulting sequences in order, and thus obtain a 2-approximation algorithm for the entire instance. It turns out that we do not have to work hard to find such a 2-approximation algorithm.

An important structural property of instances with a ratio-maximal ground set N is that sequencing the jobs in *any* feasible order constitutes a 2-approximation algorithm. Assume that N is a ratio-maximal initial set of itself. That is, $\rho(I) \leq \rho(N)$ for all initial sets $I \subseteq N$. Let us interpret this situation with the help of 2D Gantt charts. Consider an arbitrary feasible schedule, such as the one in Figure **??**, and the line segment connecting the point $(0, w(N))$ on the vertical axis with the lower right corner of any rectangle representing a job j. The absolute value of the slope of this line segment is equal to the ratio $\rho(I)$ of the initial set I of jobs defined by $I = \{k \in N : C_k \leq C_j\}$. For any job j, the lower endpoint of this line segment is on or above the diagonal defined by the two points $(0, w(N))$ and $(p(N), 0)$, because N is ratio-maximal and the negative value of the slope of this diagonal is $\rho(N)$. Recall that the area under the curve $W(t)$ is equal to $\sum_{j=1}^{n} w_j C_j$. So we have just argued that the total weighted completion time of any feasible schedule, especially that of an optimal one, is at least the area under the diagonal, which is $w(N)p(N)/2$. On the other hand, the cost of any schedule is at most $w(N)p(N)$. The proof of the following theorem is complete.

Theorem 4.20. *The total weighted completion time of any feasible sequence for* $1|\,prec|\sum w_j C_j$ *that is consistent with a Sidney decomposition is at most twice that of an optimal schedule.*

So even if the only ratio-maximal initial set is N itself, i.e., we cannot divide the original problem into smaller pieces, we are at least assured that no feasible sequence is too far from optimal.

It is time to prove Theorem 4.18. For notational convenience, we use the equivalent integer programming formulation of $1|\,prec|\sum w_j C_j$ to do so. We have to prove that, if I^* is a ratio-maximal initial set of N, then there exists an optimal solution δ^* to the integer program defined by the constraints (4.7) – (4.9) and the objective $\sum_{i \neq j} w_j p_i \delta_{ij}$ such that $\delta^*_{ij} = 1$ for all $i \in I^*$, $j \in N \setminus I^*$.

Proof. [Theorem 4.18] Let δ be an optimal solution of the integer program. Suppose $\delta_{ij} < 1$ for some $i \in I^*, j \in N \setminus I^*$. For each $k \in I^*$, define $I_k = \{j \in N \setminus I^* : \delta_{jk} > 0\}$. If a job $i \in N \setminus I^*$ is a predecessor of $j \in I_k$, i.e., $i \to j$, then the transitivity constraint (4.9) applied to the triple i, j, k, and $\delta_{ji} = 1 - \delta_{ij} = 0$ imply that $\delta_{ik} = 1 - \delta_{ki} \geq \delta_{jk} > 0$. Hence, $i \in I_k$, and I_k is an initial set of $N \setminus I^*$. Similarly, for each $k \in N \setminus I^*$, $F_k = \{i \in I^* : \delta_{ki} > 0\}$ is a final set of I^*. (That is, $I^* \setminus F_k$ is an initial set of I^*.)

Let $\varepsilon = \min\{\delta_{ij} : i \in N \setminus I^*, j \in I^*, \delta_{ij} > 0\}$, and consider the vector δ^* defined as:

$$\delta^*_{ij} = \begin{cases} \delta_{ij} + \varepsilon & \text{if } i \in I^*, j \in N \setminus I^*, \text{ and } \delta_{ij} < 1; \\ \delta_{ij} - \varepsilon & \text{if } i \in N \setminus I^*, j \in I^*, \text{ and } \delta_{ij} > 0; \\ \delta_{ij} & \text{otherwise;} \end{cases} \qquad \text{for } i, j \in N, i \neq j.$$

Clearly, $\delta^*_{ij} + \delta^*_{ji} = 1$ for all $i \neq j$, and $\delta^*_{ij} = 1$ for all $i \to j$. Consider now a triple $i, j, k \in N$, $i \neq j \neq k \neq i$. If $\{i, j, k\} \in I^*$ or $\{i, j, k\} \in N \setminus I^*$, then the associated transitivity constraint is satisfied because δ^* and δ coincide in the relevant components. Otherwise, it is convenient to rewrite the transitivity constraint $\delta_{jk} + \delta_{ki} - \delta_{ji} \leq 1$ as $\delta_{jk} + \delta_{ki} + \delta_{ij} \leq 2$. We may assume, because of symmetry, that $i \in I^*$ and $j \in N \setminus I^*$. Suppose that $\delta^*_{jk} + \delta^*_{ki} + \delta^*_{ij} > 2$. There are two cases: either $k \in I^*$ or $k \in N \setminus I^*$. In the first case, $\delta^*_{ki} = \delta_{ki}$, so we must have $\delta^*_{ij} = \delta_{ij} + \varepsilon$ and $\delta^*_{jk} > 0$. But then $\delta_{jk} = \delta^*_{jk} + \varepsilon$, and, therefore, $\delta^*_{jk} + \delta^*_{ki} + \delta^*_{ij} = \delta_{jk} + \delta_{ki} + \delta_{ij}$, a contradiction. The second case is handled analogously. Thus, δ^* also satisfies the transitivity constraints.

The difference in the objective function values of δ and δ^* can be calculated as follows:

$$\begin{aligned} \sum_{\substack{i,j \in N \\ i \neq j}} p_i w_j \delta_{ij} - \sum_{\substack{i,j \in N \\ i \neq j}} p_i w_j \delta^*_{ij} &= \varepsilon \sum_{k \in N \setminus I^*} p_k w(F_k) - \varepsilon \sum_{k \in I^*} p_k w(I_k) \\ &= \varepsilon \sum_{k \in N \setminus I^*} p_k p(F_k) \rho(F_k) - \varepsilon \sum_{k \in I^*} p_k p(I_k) \rho(I_k) \ . \end{aligned}$$

Because I^* is ratio-maximal, $\rho(I_k) \leq \rho(I^*) \leq \rho(F_k)$, for all k; see Exercise 4.28. Hence, the difference in objective function values can be bounded from below by

$$\varepsilon \rho(I^*) \Big(\sum_{k \in N \setminus I^*} p_k p(F_k) - \sum_{k' \in I^*} p_{k'} p(I_{k'}) \Big) \ .$$

Because $k' \in F_k$ if and only if $k \in I_{k'}$, this expression evaluates to zero. Therefore, the objective function value of δ^* is not worse than that of δ. So δ^* is an optimal solution as well, and $\delta^*_{ij} = 1$ for all $i \in I^*$ and $j \in N \setminus I^*$. □

Of course, in the proof $\varepsilon = 1$, and the reader might have wondered why we avoided using the integrality of δ and δ^*. The reason is that the result holds true for the linear programming relaxation in δ variables as well. That is, if I^* is a ratio-maximal initial set, then there exists an optimal solution δ^* to the linear program to minimize $\sum_{j \in N} w_j C_j$ subject to (4.7) – (4.11) such that $\delta^*_{ij} = 1$ whenever $i \in I^*$ and $j \in N \setminus I^*$. We can use the same proof; while ε need not be equal to 1 anymore, the variable that determined the value of ε will be reduced to 0 in δ^*. If δ^* does not have the desired property yet, we can repeat the same procedure with $\delta = \delta^*$ until it does.

Corollary 4.21. *Let $(I_1, I_2, \ldots, I_k)$ be a Sidney decomposition of $1|prec|\sum w_j C_j$. There exists an optimal solution δ^* of the linear program to minimize $\sum_{j \in N} w_j C_j$ subject to* (4.7) – (4.11) *such that $\delta^*_{ij} = 1$ for all $i \in I_h$, $j \in I_\ell$ whenever $1 \leq h < \ell \leq k$.*

Exercises

4.28. Let S be a ratio-maximal initial set of N, I an initial set of $N \setminus S$, and F a final set of S. Show that $\rho(I) \leq \rho(S) \leq \rho(F)$.

4.29. Let I and J be two ratio-maximal initial sets of N. Are $I \cap J$ and $I \cup J$ ratio-maximal initial sets as well? Prove or disprove.

4.30. Show that there exists a unique Sidney decomposition $(I_1, I_2, \ldots, I_k)$ such that $\rho(I_1) > \rho(I_2) > \cdots > \rho(I_k)$. Prove also that for any other Sidney decomposition $(J_1, J_2, \ldots, J_\ell)$ of the same instance and any index $i \in \{1, \ldots, \ell\}$, $J_i \subseteq I_j$ for some $j \in \{1, \ldots, k\}$.
4.31. Let $(N, \rightarrow)$ be a series-parallel partial order that is the parallel composition of $(N_1, \rightarrow)$ and $(N_2, \rightarrow)$. Show that there exists a ratio-maximal initial set I of N that is completely contained in either N_1 or N_2.

4.8. An integrality theorem for series-parallel precedence constraints

With Corollary 4.21 in place, the following result comes essentially for free.

Theorem 4.22. *When the precedence constraints are series-parallel, then the linear program to minimize* $\sum_{j \in N} w_j C_j$ *subject to* (4.7) – (4.11) *has an integer optimal solution.*

Proof. The proof is by induction on the number of jobs. The result is obviously true if there are just one or two jobs. Suppose it is true for all instances with series-parallel precedence constraints on n jobs, and consider the case with $|N| = n+1$. The set N is either a series or a parallel composition of two proper subsets N_1 and N_2. In the first case, all variables δ_{ij} with $i \in N_1$ and $j \in N_2$ have to be equal to 1, because of (4.8). Hence, the linear program for $N_1 \cup N_2$ decomposes into two separate linear programs, one for N_1 and one for N_2, each of which has an integer optimal solution, by induction. In the second case, N is the parallel composition of N_1 and N_2. Let I^* be an inclusion-minimal ratio-maximal initial set of N; it follows from Exercise 4.31 that I^* is entirely contained in N_1 or N_2. By Corollary 4.21, there exists an optimal solution δ^* such that $\delta^*_{ij} = 1$ for $i \in I^*$ and $j \in N \setminus I^*$. The linear program for N therefore decomposes again into two separate, smaller linear programs for I^* and $N \setminus I^*$, respectively, which have integer optimal solutions. □

Theorem 4.22 implies that the optimal value of the linear program is identical to the value of the optimal schedule. It does not necessarily imply that an optimal solution to the linear program is integer (and, therefore, a schedule) because there can be noninteger optimal basic feasible solutions, although this can be overcome by data perturbation (see Exercise 4.32 (a) and (b)). There is a related, seemingly weaker linear program in δ variables that has the same properties as the one that we have presented so far. It has the additional property that *all* its basic feasible solutions are integer if the precedence constraints are series-parallel. To prove this result, it will be convenient to work with the following characteristic of series-parallel precedence constraints.

Lemma 4.23. *If the precedence constraints are series-parallel, then there exists a total ordering* π *of all jobs, which is consistent with the precedence constraints, such that, for all triples* $i, j, k \in N$ *with* $i \rightarrow j$ *and* k *unrelated to* i *and* j*, either* k *precedes* i *in* π *or* j *precedes* k*.*

Proof. Consider the decomposition tree of the precedence constraints, and the total ordering of jobs obtained by parsing the leafs from left to right. Note that this ordering is consistent with the precedence constraints; if $i \to j$, then the leaf corresponding to i is to the left of that of j. Now consider a job k that is unrelated to i and j. The tree has an internal node representing a parallel composition where i and j are part of one of the two subtrees, while k is in the other subtree. Hence, k's leaf is to the left of the leafs of both i and j, or to their right. □

We use the total ordering π from Lemma 4.23 to eliminate half of the δ variables from the previous linear program. Because of (4.7), i.e., $\delta_{ij} + \delta_{ji} = 1$, we know the value of δ_{ij} when we know that of δ_{ji}, and vice versa. We only keep the variables δ_{ij} for which i precedes j in π. We also eliminate all variables whose values are fixed by the precedence constraints; i.e., we do not keep δ_{ij} with $i \to j$. Apart from the nonnegativity constraints, we are left with the transitivity constraints, some of which we shall drop as well. More precisely, we only keep the transitivity constraints

$$\delta_{jk} + \delta_{ki} - \delta_{ji} \leq 1$$

for triples $i, j, k \in N$, $i \neq j \neq k \neq i$, for which $i \to j$, and k is unrelated to both i and j. Depending on whether j precedes k or k precedes i in π, we derive one of the following two inequalities:

$$\delta_{jk} \leq \delta_{ik} \text{ or } \delta_{ki} \leq \delta_{kj} \quad .$$

For convenience, let us assume that we (re)index the jobs so that they appear in the order $1, 2, \ldots, n$ in π. The resulting linear program,

$$\begin{aligned}
\min \quad & \sum_{\substack{i<j \\ i \not\to j}} w_j p_i \delta_{ij} && && (4.12a)\\
\text{s.t.} \quad & \delta_{jk} \leq \delta_{ik} && \text{for } i \to j, j < k, && (4.12b)\\
& \delta_{ki} \leq \delta_{kj} && \text{for } i \to j, k < i, && (4.12c)\\
& \delta_{ij} \geq 0 && \text{for } i < j, && (4.12d)
\end{aligned}$$

is a min-weight closure problem (see Chapter **??**); in particular, its constraint matrix is totally unimodular, and thus all basic feasible solutions are integral.

Interestingly, the linear program (4.12) has the same optimal value as the previous one, although it has fewer constraints (Exercise 4.32 (c)). Because one can find an optimal sequence when one can determine the optimal value (Exercise 4.33), this yields another way of efficiently computing an optimal sequence for series-parallel precedence constraints. As the derivation of the linear program (4.12) only hinges on Lemma 4.23, we have actually proved that $1|prec|\sum w_j C_j$ can be solved in polynomial time for a much larger class of precedence constraints. The class of partial orders possessing a *nonseparating linear extension*, i.e., a total order π with the property described in Lemma 4.23 coincides with that of *two-dimensional* partial orders. The *dimension* of a partial order $\to$ is the minimum number of linear orders whose

intersection is $\rightarrow$. There are efficient algorithms for recognizing two-dimensional partial orders. If the partial order is two-dimensional, they return a nonseparating linear extension. We can conclude this section with the following theorem.

Theorem 4.24. *If the precedence constraints are two-dimensional, an optimal sequence for $1|prec|\sum w_jC_j$ can be found in polynomial time.*

Exercises

4.32. Consider the linear program to minimize $\sum_{j\in N} w_jC_j$ subject to (4.7) – (4.11).

(a) Show that it can have nonintegral optimal basic feasible solutions, even if there are no precedence constraints.

(b) Let $0 < \varepsilon < 1/(2\sum_{j\in N} p_j)$, and assume that the precedence constraints are series-parallel. Prove that the instance with redefined weights $\widetilde{w}_j = w_j + \varepsilon^{2j}$ has a unique Sidney decomposition. Use this fact in the proof of Theorem 4.22 to show that the linear program has a unique optimal solution, which is a schedule.

(c) Show that any optimal solution to this linear program is also an optimal solution to the linear program (4.12).

4.33. Suppose someone gives you a "black box" subroutine that, given an instance of $1|prec|\sum w_jC_j$, returns its optimal value. Show that you can find an optimal sequence with a polynomial number of calls on the subroutine.

4.34. Show that a partial order is two-dimensional if and only if it has a nonseparating linear extension.

4.35. Present a two-dimensional partial order that is not series-parallel.

4.9. Approximation algorithms for $1|r_j|\sum C_j$

Once we add release dates to the problem of minimizing the average completion time, we again reach the realm of *NP*-complete problems. Since we are unlikely to have a polynomial-time algorithm to solve this problem, we turn our attention to approximation algorithms. Our approach will again be based on solving a relaxation to the problem; in this case, we relax the condition that the schedule must be nonpreemptive. We then show how to convert the optimal preemptive schedule to a nonpreemptive one, without increasing the objective function value of the schedule too much.

The first component of this approach is that the preemptive variant can be solved efficiently. The natural rule to consider is the *Shortest Remaining Processing Time* (*SRPT*) rule: at each moment in time, one should always be processing a job that could be completed earliest. This rule only preempts jobs when a new job is released. A straightforward interchange argument can be used to prove the following theorem; this will be left as an exercise.

Theorem 4.25. *Any SRPT schedule is optimal for the problem* $1|r_j, pmtn|\sum C_j$.

An important property of the SRPT rule is that it is an online algorithm, in the sense that the algorithm need only know of the existence of a job and its parameters at the moment in time that it is released.

A seemingly naive approach to converting a preemptive schedule into a nonpreemptive one is to schedule the jobs in the order in which they completed in the preemptive one. Yet, recall that a similar approach has worked well for $1|prec|\sum w_j C_j$. Let C_j, $j = 1\ldots,n$, denote the completion times of jobs in a preemptive schedule. We can assume, without loss of generality, that we have reindexed the jobs so that $C_1 \leq C_2 \leq \cdots \leq C_n$. We construct the minimal nonpreemptive schedule in which the jobs are processed in this order; job 1 is processed from r_1 to $r_1 + p_1$, and each job $j+1$ is either scheduled to start at the time $\overline{C}_j$ that j completes, or at its release date, r_{j+1}, whichever is later.

We can analyze the completion time $\overline{C}_j$ of job j in the following way. Trace backwards from $\overline{C}_j$ in this schedule to find the latest idle time prior to the completion of job j; we see that

$$\overline{C}_j = r_k + \sum_{\ell=k}^{j} p_\ell,$$

where job k is the first job processed after this idle time. Since the completion of job k also precedes the completion of job j in the preemptive schedule, we have that $r_k \leq C_k \leq C_j$. Furthermore, all of the jobs ℓ, $\ell = k,\ldots,j$, have been completely processed in the preemptive schedule by C_j, and hence $\sum_{\ell=k}^{j} p_\ell \leq C_j$. Therefore $\overline{C}_j \leq 2C_j$.

Thus, we see that $\sum_{j=1}^{n} w_j\overline{C}_j \leq 2\sum_{j=1}^{n} w_j C_j$; given *any* preemptive schedule, we can find a nonpreemptive schedule with total weighted completion time no more than twice that of the preemptive one. Of course, if we start with the optimal preemptive schedule, then we have found a nonpreemptive schedule with objective function value at most twice that of the preemptive optimum. The preeemptive optimum is always at most the nonpreemptive optimal value, and so we have just proved that the schedule found has total weighted completion time at most twice the optimal value.

Theorem 4.26. *Sequencing jobs in order of nondecreasing completion times in the SRPT schedule is a 2-approximation algorithm for the problem* $1|r_j|\sum C_j$.

In fact, we can strengthen this result to derive an algorithm that is an online 2-approximation algorithm. The idea behind this is quite simple. We maintain a queue of available jobs (which handles the jobs in a "first-in first-out" manner, and hence fixes the order in which the jobs are processed in the schedule). Instead of making a job available at its release time, we will use the online SRPT rule to create a *shadow* schedule. This schedule is not used for the actual processing of the jobs, but only as a kind of side computation. Nonetheless, this side computation can be done in an online manner.

We make a job j available only at date C_j^{SRPT}, the time that it is *completed* in the SRPT schedule. Clearly, this queueing mechanism ensures that the jobs are processed in the order of their completion in the preemptive schedule, but it most likely introduces additional idle time into the schedule. However, the analysis does not change much. Now, the latest idle time prior to $\overline{C}_j$ ends with the point in time that some job k is made available (instead of being released). Hence,

$$\overline{C}_j = C_k^{\text{SRPT}} + \sum_{\ell=k}^{j} p_\ell \leq 2C_j^{\text{SRPT}},$$

and we still have that $\sum w_j\overline{C}_j \leq 2\sum w_j C_j^{\text{SRPT}}$; we have proved the following theorem.

Theorem 4.27. *Scheduling jobs in the order of nondecreasing SRPT completion times with delayed starts is an online* 2*-approximation algorithm for the problem* $1|r_j|\sum C_j$.

Surprisingly, this result is best possible, in the sense that, for any $\alpha < 2$, there does not exist a deterministic online α-approximation algorithm for $1|r_j|\sum C_j$ (Exercise 4.37). However, as we shall see next, if the algorithm is allowed to "toss coins," one can prove a better result, in expectation. For each input, the objective function value of the schedule found by such a randomized algorithm is a random variable; it is natural to analyze the performance of such a randomized algorithm by considering its expected value, and to show, for each possible input, that this expectation is no more than a factor of ρ times the optimum value. We call an algorithm with this performance guarantee a *randomized* ρ*-approximation algorithm*.

We will now give an online randomized $e/(e-1)$-approximation algorithm called the *randomized* α*-point algorithm*. The algorithm is based on a more general procedure to convert a preemptive schedule σ into a nonpreemptive one. The algorithm first picks an $\alpha \in (0,1]$ according to some probability density function f. We define the α*-point* $C_j(\alpha)$ of job j to be the first moment in time where a total of αp_j units of processing of job j have been completed in σ. We now modify the online algorithm discussed above, so that each job is made available, and placed into the queue, only at its α-point. This means that the algorithm now schedules the jobs in the order that the α-points occur in σ, with the additional condition that j is not allowed to be started prior to $C_j(\alpha)$. Subject to these two constraints, the schedule $\overline{\sigma}$ produced is minimal in the sense that each jobs starts as early as possible. Let $\overline{C}_j$ denote the completion time of job j in $\overline{\sigma}$; that is, if π is the order in which the jobs are processed, then $\pi(1)$ completes at time $\overline{C}_{\pi(1)} = C_{\pi(1)}(\alpha) + p_{\pi(1)}$, and in general, each job $\pi(j)$ starts at the maximum of its α-point in σ and $\overline{C}_{\pi(j-1)}$, and completes $p_{\pi(j)}$ time units later, $j = 2,\ldots,n$. We will analyze the expected objective function value $\sum_j \overline{C}_j$ as a function of the choice of α.

We next introduce notation that will be convenient for the analysis of $\overline{\sigma}$. We will give, for each $j = 1,\ldots,n$, an upper bound on $\overline{C}_j$ that depends on our choice of α. Let us focus on the completion time of some job j. Let $\beta_k(\alpha)$ be the fraction of job k

that has been completed in the preemptive schedule σ by the α-point of job j, that is, by $C_j(\alpha)$. For simplicity, let $\beta_k = \beta_k(1)$. Similarly, let $R(\alpha)$ denote the total idle time in the preemptive schedule prior to the α-point of j. Since the machine, at any point in time, is either processing some job, or idle, we get the following lemma.

Lemma 4.28. *In the preemptive schedule σ, for each $\alpha \in (0,1]$, the α-point of job j is equal to $C_j(\alpha) = R(\alpha) + \sum_{k=1}^{n} \beta_k(\alpha) p_k$.*

On the other hand, we show next that we can also express the completion time of j in $\overline{\sigma}$ in terms of similar components.

Lemma 4.29. *In the nonpreemptive schedule $\overline{\sigma}$, job j completes by time*

$$\overline{C}_j \leq C_j(\alpha) + \sum_{k:\beta_k(\alpha)\geq\alpha} (1+\alpha-\beta_k(\alpha))p_k.$$

Proof. We shall analyze a worse nonpreemptive schedule $\hat{\sigma}$, in the sense that for each $k = 1,\ldots,n$, we have that the completion time $\hat{C}_k \geq \overline{C}_k$.

Imagine starting out with the optimal schedule σ for the preemptive relaxation, and write the Gantt chart for this schedule on a strip of ticker tape. We will modify this schedule by a number of "cut and paste" operations to convert it into a nonpreemptive schedule $\hat{\sigma}$.

Consider the schedule σ "over time," starting at time 0 and advancing in time, until an α-point of some job k is reached. Cut the schedule at this point, and paste into this schedule a block of time of length p_k that will be used for the nonpreemptive processing of job k. For the intervals of time that are spent processing job k earlier in this schedule, replace the processing of job k with new idle time, which we will call *omission time*. For each interval of time that is spent processing k after this nonpreemptive block, cut that time interval out of the schedule, and then rejoin the schedule without any new idle time introduced at this point. Continue processing the schedule in this way until the α-point of each job has been reached. Call this schedule $\hat{\sigma}$. See Figure **??** for an illustration.

We begin by making a number of observations. The effect of inserting the nonpreemptive block of length p_k shifts the entire subsequent schedule exactly p_k time units later. However, the deletion of the processing of job k beyond this new block of length p_k shifts earlier some portion of the schedule beyond this block. A total of length $(1-\alpha)p_k$ is deleted, and so the end of the schedule is shifted $(1-\alpha)p_k$ time units earlier by these deletions. However, the net effect of the changes prompted by reaching the α-point of job k does not shift any portion of the schedule earlier. This allows us to conclude, for example, that the schedule is feasible, since each job k starts no earlier than r_k before the modifications, and hence must start no earlier than r_k after them. Furthermore, we also see that as we modify the schedule, when we find an α-point of job k in the schedule, it must currently correspond to a time that is at least $C_k(\alpha)$. Thus, we are scheduling the nonpreemptive block for each job k to start no earlier than $C_k(\alpha)$. Finally, the jobs are processed in this schedule in the

same order as their α-points in σ. We have constructed a schedule $\hat{\sigma}$ in which the jobs are processed in the same order as in σ, and no job k starts before $C_k(\alpha)$. Since $\overline{\sigma}$ is the minimal nonpreemptive schedule that is consistent with these constraints, we see that $\overline{C}_k \leq \hat{C}_k$, $k = 1, \ldots, n$.

We now analyze the schedule $\hat{\sigma}$ up to the completion time of job j. Each time interval on the ticker tape after these operations corresponds to one of the following:

- true idle time (i.e., in the preemptive schedule σ, not omission time): these add up exactly to $R(\alpha)$ since we have not altered true idle time.
- processing time: the jobs k that are processed in $\hat{\sigma}$ by time $\hat{C}_j$ are exactly those that reach their α-point in σ no later than $C_j(\alpha)$; in terms of the notation that we have set up, this is equivalent to writing that $\beta_k(\alpha) \geq \alpha$. Hence this total is equal to $\sum_{k:\beta_k(\alpha)\geq\alpha} p_k$.
- omission time for each job k that has reached its α-point in σ by time $C_j(\alpha)$ (including j itself): for each job k, we replaced the first αp_k times unit of its processing by omission time (here all of this time occurs prior to the processing of j); hence this totals $\sum_{k:\beta_k(\alpha)\geq\alpha} \alpha p_k$.
- omission time for each job k that reaches its α-point in σ after time $C_j(\alpha)$: in this case, considering just the part of schedule $\hat{\sigma}$ up to $\hat{C}_j$, only the fraction $\beta_k(\alpha)$ of the processing of k that precedes $C_j(\alpha)$ is replaced by omission time; hence this contributes a total of $\sum_{k:\beta_k(\alpha)<\alpha} \beta_k(\alpha) p_k$.

This yields

$$\hat{C}_j = R(\alpha) + \sum_{k:\beta_k(\alpha)<\alpha} \beta_k(\alpha) p_k + (1+\alpha) \sum_{k:\beta_k(\alpha)\geq\alpha} p_k.$$

Applying Lemma 4.28, we see that this can be written as

$$\hat{C}_j = C_j(\alpha) + \sum_{k:\beta_k(\alpha)\geq\alpha} (1+\alpha-\beta_k(\alpha)) p_k.$$

Since $\overline{C}_j \leq \hat{C}_j$, for any given choice of α ,

$$\overline{C}_j \leq C_j(\alpha) + \sum_{k:\beta_k(\alpha)\geq\alpha} (1+\alpha-\beta_k(\alpha)) p_k.$$

This completes the proof of the lemma. □

This also has the following immediate consequence.

Corollary 4.30. *In the nonpreemptive schedule $\overline{\sigma}$, job j completes by time*

$$\overline{C}_j \leq C_j + \sum_{k:\beta_k\geq\alpha} (1+\alpha-\beta_k) p_k.$$

Here, C_j is the completion time of job j in the preemptive schedule σ.

Proof. From Lemma 4.29,

$$\begin{aligned}\overline{C}_j &\leq C_j(\alpha) + \sum_{k:\beta_k(\alpha)\geq\alpha} (1+\alpha-\beta_k(\alpha))p_k \\ &= C_j(\alpha) + \sum_{k:\beta_k(\alpha)\geq\alpha} (\beta_k-\beta_k(\alpha))p_k + \sum_{k:\beta_k(\alpha)\geq\alpha} (1+\alpha-\beta_k)p_k.\end{aligned}$$

The second term in this last expression corresponds to work done on jobs k between $C_j(\alpha)$ and C_j, and hence the sum of the first two terms is at most C_j. □

It is now just a straightforward calculation to compute an upper bound on the expected completion time $E[\overline{C}_j]$ of job j in $\overline{\sigma}$: for any probability density function $f(x)$, we can compute the expectation of the upper bound just derived.

Theorem 4.31. *If $\alpha \in (0,1]$ is selected according to the probability density function $f(x) = e^x/(e-1)$, then for the nonpreemptive schedule $\overline{\sigma}$, $E[\overline{C}_j] \leq \frac{e}{e-1}C_j$.*

Proof. We can bound $E[\overline{C}_j]$ by integrating the bound in Corollary 4.30. Note that α appears in two ways in the bound of this corollary: in the $(1+\alpha-\beta_k)p_k$ term, and in the fact that we need that $\alpha \leq \beta_k$ for any job k that contributes to the bound. Alternatively, for each job k, we can focus on those α such that $\alpha \leq \beta_k$. Hence, we have that

$$E[\overline{C}_j] \leq C_j + \sum_{k=1}^{n} p_k \int_0^{\beta_k} f(\alpha)(1+\alpha-\beta_k)d\alpha.$$

Straightforward calculus (if one recalls that integrating xe^x yields $e^x(x-1)$) shows that

$$\int_0^{\beta} f(x)(1+x-\beta)dx = \frac{\beta}{e-1}.$$

Hence, since $\sum_{k=1}^{n} p_k\beta_k \leq C_j$ by Lemma 4.28,

$$E[\overline{C}_j] \leq C_j + \frac{1}{e-1}\sum_{k=1}^{n} p_k\beta_k \leq \frac{e}{e-1}C_j.$$

□

The expectation of the total completion time is, by linearity of expectation, within a factor $e/(e-1)$ of the total completion time of the preemptive schedule σ. If we apply this randomized conversion to the optimal solution σ^* for $1|r_j, pmtn|\sum C_j$ found by the SRPT rule, we see that the expected total completion time is at most $e/(e-1)$ times the preemptive optimum. The preemptive optimum is, of course, a lower bound on the nonpreemptive optimum and hence we have obtained the following theorem.

Theorem 4.32. *The randomized α-point algorithm which draws α according to the density function in Theorem 4.31 provides an online randomized $e/(e-1)$-approximation algorithm for the problem $1|r_j|\sum C_j$.*

There are a few immediate implications of this result. Note that we have constructed a nonpreemptive schedule of total completion time within a factor of $e/(e-1) \approx 1.58$ of the preemptive optimum. This places a limit on the power of preemption: by allowing preemption, we can improve the objective function by at most a factor of 1.58. As this argument does not depend on our ability to solve the preemptive problem in polynomial time, the limit on the power of preemption is the same for the total weighted completion time objective.

Many people are troubled at first by the fact that our focus on approximation algorithms is in order to be guaranteed that the result is good, and yet now we only know that it is expected to be good. While that is true, it is important to realize that this theorem says that *for every input* we are assured that the expected result will be good, and the source of randomness is merely the coin tosses used by the algorithm.

However, one can convert this randomized algorithm into a deterministic algorithm (provided we are considering off-line algorithms, where we can see the entire input in advance). We now just compute the minimal schedule in which the jobs are ordered consistently with their α-points. (That is, $\overline{C}_1 = r_1 + p_1$ and $\overline{C}_j = \max\{\overline{C}_{j-1}, r_j\} + p_j$.) Since we have shown that by choosing α in this random manner, the α-point algorithm delivers a schedule with expected objective function value within a factor of 1.58 of the optimum, then there must exist some α with the property that if that α is used, then that schedule has objective function value within a factor of 1.58 of the optimum. So, if we could compute an α for which this α-point algorithm would deliver the best solution, then the schedule found must be within a factor of 1.58 of the optimum.

If we consider the schedule constructed for various values of α, the schedule only changes when the order in which the jobs reach their α-points in the SRPT schedule σ^* changes. This can only happen when there exists some job that is preempted when exactly an α fraction of its processing has been completed. Recall that in σ^* a job is only preempted if another job arrives whose processing time is shorter than the remaining one of the current job. In particular, there are at most $n-1$ preemptions. Hence there are at most n potentially critical values of α, and we can compute these critical values directly from σ^*. If we run the α-point algorithm for all of these values of α and take the best schedule, then we have deterministically computed a schedule within a factor of 1.58 of optimal.

Theorem 4.33. *The best α-point algorithm is a (deterministic) $\frac{e}{e-1}$-approximation algorithm for the problem $1|r_j|\sum C_j$.*

The only reason that the same approach does not lead to an approximation algorithm of the same performance guarantee for $1|r_j|\sum w_j C_j$ is that its preemptive relaxation $1|r_j, pmtn|\sum w_j C_j$ is NP-hard (see Theorem 4.13). We will therefore discuss a similar conversion technique that starts from another, efficiently computable preemptive schedule in the next section.

Exercises

4.36. Prove Theorem 4.25.

4.37. Show that no deterministic approximation algorithm for $1|r_j|\sum C_j$ that works online, can have a performance guarantee smaller than 2.
4.38. Use Yao's minimax principle to show that no randomized approximation algorithm for $1|r_j|\sum C_j$ that works online, can have an expected performance guarantee smaller than $e/(e-1)$.

4.10. Mean busy times revisited & approximation algorithms for $1|r_j|\sum w_j C_j$

For nonpreemptive schedules, the mean busy time of each job j is a mere translation of the completion time by $p_j/2$. But it is also interesting to consider mean busy times for preemptive schedules. Let $I_j(t) = 1$ if job j is in process at time t, and 0 otherwise. The function I_j is called the *indicator function* of job j in the given schedule. Because the schedule is feasible, all p_j units of work for each job j are processed, that is,

$$\int_0^{+\infty} I_j(t)\,dt = p_j.$$

Consider again the 2-dimensional Gantt chart, and the interpretation that $\rho(j)$ specifies the holding cost of job j per unit of unprocessed work. Consider the curve defined by the function $\overline{W}(t)$, introduced in Section 4.1, which is piecewise linear; each piece corresponding to the (partial) processing of job j has slope $-\rho(j) = -w_j/p_j$. As we have already seen, the area under this curve corresponds to the total cost of holding the inventory over the planning horizon. We can decompose this area by breaking it into horizontal trapezoidal pieces, each of which corresponds to a unique job j being (partially) processed for an interval $[s,s']$; see Figure **??**. The area in this trapezoid is

$$\rho(j)(s'-s)(s'+s)/2 = \int_s^{s'} \rho(j)\,t\,dt$$

and so the total area in all of the trapezoids corresponding to j is $\int_0^{+\infty} \rho(j)\,t\,I_j(t)\,dt$. This is the total cost of holding job j over the planning horizon; this is equivalent to w_j times M_j, the *mean busy time* of job j, where

$$M_j = \frac{1}{p_j}\int_0^{+\infty} I_j(t)\,t\,dt\ . \tag{4.13}$$

Note that this is the natural extension of the definition of M_j in the nonpreemptive case. Thus the mean busy time M_j is the average time at which the machine is processing job j. Even in this preemptive setting, if we let S_j denote the start time of

job j in a given schedule, we have

$$S_j + \frac{1}{2}\, p_j \leq M_j \leq C_j - \frac{1}{2}\, p_j\,, \tag{4.14}$$

where each of these inequalities holds with equality if and only if job j is processed without preemption.

The interpretation of $\sum_j w_j M_j$ as the area under the curve $\overline{W}(t)$ leads to an easy extension of the ratio rule to solve the problem $1|r_j, pmtn|\sum w_j M_j$. Intuitively, this area is made small by having the curve descend to 0 as sharply and as early as possible. In other words, consider the *preemptive ratio rule*: at each moment in time, among all available jobs choose a job j for which the ratio w_j/p_j is maximum. Thus, we need preempt a job j only when a job k with a bigger ratio is released.

We can prove the optimality of the preemptive ratio rule by an interchange argument. Suppose that the schedule σ produced by the algorithm is not optimal, and consider an optimal schedule σ^* whose earliest difference from σ is as late as possible. Suppose that the two schedules agree up to time t, at which point σ^* processes job j, whereas σ processes job k. Of course, this means that σ^* has not completed work on job k at time t, and k must still be processed later, next starting at some time t'. Consider these two fragments of processing jobs, job j at t and job k at t' in σ^*, and suppose that the jobs are processed in time periods $[t, t+\delta)$ and $[t', t'+\delta)$, respectively, for some $\delta > 0$. Both jobs have been released by t, and so we could swap the processing of δ time units of the two jobs. By the preemptive ratio rule, $\rho(k) \geq \rho(j)$. However, since the objective function value is the area under the $\overline{W}(t)$ curve, we see that unless $\rho(j) = \rho(k)$, this interchange will improve the objective function value, which contradicts the optimality of σ^*. But if $\rho(j) = \rho(k)$ then this interchange produces another optimal schedule, but one for which the ratio rule agrees even longer than for σ^*. This contradiction yields the following theorem.

Theorem 4.34. *A schedule is optimal for $1|r_j, pmtn|\sum w_j M_j$ if and only if it is a preemptive ratio rule schedule.*

In the previous section, we saw that the fact that $1|r_j, pmtn|\sum C_j$ could be solved in polynomial time led to a good approximation algorithm for the nonpreemptive variant. Next, we will see that Theorem 4.34 provides a similar engine for the weighted extension. In fact, it might seem natural to take the solution provided by this algorithm, and then process the jobs nonpreemptively in the order in which their mean busy times occur. Unfortunately, this algorithm does not perform well (see Exercise 4.40). Instead, we will use the second technique of the previous section: we will consider a randomized approach to converting the preemptive ratio rule schedule to a nonpreemptive one.

In fact, we will consider a somewhat more flexible algorithm, which we call the *randomized α_j-point algorithm*:

(1) compute an optimal schedule for $1|r_j, pmtn|\sum w_j M_j$;
(2) for $j = 1, \ldots, n$, pick α_j independently and uniformly at

random from the interval $(0,1]$;
(3) for each job j, compute its α_j-point $C_j(\alpha_j)$
in the optimal preemptive schedule;
(4) schedule the jobs nonpreemptively in nondecreasing order of their
α_j-points, always scheduling the next job as early as possible
(subject to this ordering constraint).

As we did in the previous section, our analysis of the algorithm is more general than we need for deriving an approximation algorithm; we will show that given any preemptive schedule in which job j has mean busy time M_j, the expected value of $\sum w_j C_j$ for the schedule produced by this algorithm is at most $2\sum_j w_j(p_j/2+M_j)$.

We start with an easy observation, that follows directly from the definition of the mean busy time: choosing α_j uniformly in $(0,1]$ is just a rescaled version of choosing a time uniformly within $(0,p_j]$.

Lemma 4.35. *If α_j is selected uniformly at random in $(0,1]$, then $E[C_j(\alpha_j)] = M_j$.*

Let $\overline{C}_j$ denote the completion time of job j in the schedule $\overline{\sigma}$ produced by the algorithm. Note that $\overline{C}_j$ is a random variable, since the ordering depends on the random choices α_j, $j = 1,\ldots,n$. Nonetheless, we can write that $\overline{C}_j = P_j + R_j$, where the random variables P_j and R_j are, respectively, the total processing and total idle times prior to the completion of job j in the schedule.

Lemma 4.36. *The randomized α_j-point algorithm produces a schedule that, with probability 1, has $R_j \le C_j(\alpha_j)$, $j = 1,\ldots,n$.*

Proof. We will show that for *any* realization of the values α_j, $j = 1,\ldots,n$, the machine is never idle within the time interval $(C_j(\alpha_j),\overline{C}_j]$; clearly, this implies the lemma. Any job k that is processed prior to job j can have its release date no later than $C_j(\alpha_j)$ (since it has reached its α_k-point by then). Suppose that there was idle time in the interval $(C_j(\alpha_j),\overline{C}_j]$; this must be due to the fact that the next job in the ordering is not yet released, but this is impossible, since all jobs processed before j have been released by this point. □

Next we analyze, for each job j, the total time P_j the machine is busy prior to the completion of j.

Lemma 4.37. *The randomized α_j-point algorithm produces a schedule for which $E[P_j] \le p_j + M_j$.*

Proof. Focus on a job j; each job is processed prior to its own completion, and so we see that

$$E[P_j] = p_j + \sum_{k \neq j} Pr[k \text{ precedes } j \text{ in the schedule } \overline{\sigma}]\, p_k.$$

In order to analyze the summand, we first fix a value $\alpha \in (0,1]$, and condition on the event that $\alpha_j = \alpha$. This fixes the α_j-point of job j at some time t. Once we have

done this, it is easy to compute the conditional probability that job k precedes job j; k precedes j if and only if α_k is chosen such that $C_k(\alpha_k)$ is at most t. If we let β_k denote the fraction of job k that is processed in the preemptive schedule prior to t, then the probability that $C_k(\alpha_k)$ is at most t is exactly β_k. Note that this uses the fact that each α_k is chosen independently of α_j. Thus, $\sum_{k\neq j} Pr[k \text{ precedes } j \text{ in } \overline{\sigma} \,|\, \alpha_j = \alpha] = \sum_{k\neq j} \beta_k p_k$. But this is exactly the total processing done prior to t (in the preemptive schedule) on jobs other than j, and hence is at most $t = C_j(\alpha)$. Thus, $E[P_j \,|\, \alpha_j = \alpha] \leq p_j + C_j(\alpha)$.

Taking the expectation over all such choices α and applying Lemma 4.35, we see that $E[P_j] \leq p_j + M_j$. □

It is now a simple matter of putting the pieces together to obtain the following theorem.

Theorem 4.38. *The randomized α_j-point algorithm is a randomized 2-approximation algorithm for $1|r_j|\sum w_jC_j$.*

Proof. Let C_j^*, $j = 1,\ldots,n$, denote the completion times in an optimal schedule for $1|r_j|\sum w_jC_j$. This implies that the mean busy times for this schedule are $C_j^* - p_j/2$, and hence the preemptive ratio rule produces a schedule whose mean busy times M_j, $j = 1,\ldots,n$, satisfy the inequality

$$\sum_{j=1}^{n} w_j M_j \leq \sum_{j=1}^{n} w_j (C_j^* - p_j/2).$$

However, the randomized α_j-point algorithm produces a schedule with completion times $\overline{C}_j$, $j = 1,\ldots,n$, such that, by Lemmata 4.36 and 4.37,

$$E[\sum_{j=1}^{n} w_j \overline{C}_j] = \sum_{j=1}^{n} w_j E[\overline{C}_j] = \sum_{j=1}^{n} w_j E[P_j + R_j] \leq \sum_{j=1}^{n} w_j (p_j + 2M_j) \leq 2 \sum_{j=1}^{n} w_j C_j^*,$$

which concludes the proof of the theorem. □

Note that the randomized α_j-point algorithm works online. One can also derandomize it to obtain a deterministic 2-approximation algorithm; however, for this we need to see the entire instance in advance, and so it does not work online. We conclude this chapter by describing a (deterministic) online algorithm that produces a solution that is within a factor 2 of the offline optimum.

The *delayed ratio rule* works as follows: At time t, let j be an available job with highest w_j/p_j ratio. If $t \geq p_j$, then start job j. Otherwise, do nothing until either time p_j or a job with better ratio is released, whichever comes first.

Let σ_0 be the schedule returned by the delayed ratio rule. As any other feasible nonpreemptive schedule for $1|r_j|\sum w_jC_j$, σ_0 can be decomposed into blocks of consecutive jobs and idle time between the blocks. For the analysis, it will suffice to look at an arbitrary block. Let I be the instance defined by the jobs in one such block, and let σ be the restriction of σ_0 to I. We define another instance I' from I

by changing the release dates of all jobs j in I to $r'_j = \min\{S_j, 2r_j\}$, where S_j is the starting time of job j in the schedule σ. Moreover, we add a new job k to I' with $r_k = 0$ and $p_k = t_I$, where $t_I = \min\{S_j : j \in I\}$. In order to define w_k, let us consider the situation in σ_0 right before time t_I, when the machine was idle (unless $t_I = 0$, in which case we set $w_k = 0$). This idle time can be caused by either the fact that no job is available or the job h with the currently highest ratio w_h/p_h is delayed because of $p_h \geq t_I$. If h belongs to I, we define $w_k = w_h p_k / p_h$; otherwise $w_k = 0$. In either case, we have

$$\sum_{j \in I} w_j p_j \geq w_k p_k. \tag{4.15}$$

Finally, let σ' be defined from σ by assigning job k to the time period $[0, t_I)$.

Lemma 4.39. *The schedule σ' is an optimal preemptive schedule for the instance I' and the weighted mean busy time objective.*

Proof. According to Theorem 4.34, we only need to show that σ' processes at any point t in time a job with the highest ratio among all the jobs that are not yet completed. We distinguish two cases.

In case $t < t_I$, σ' is currently processing job k. Suppose there is another job $j \in I'$ available at that time; i.e., $r_j \leq r'_j \leq t$. Job j is not started in σ_0 before time t_I, so $w_h/p_h \geq w_j/p_j$. If $h \in I'$, then $w_k/p_k = w_h/p_h$, and k is a job of highest ratio available at time t. If $h \notin I'$, the algorithm did not wait until time p_h before it started processing other jobs. By the definition of the delayed ratio rule, this implies $w_j/p_j > w_h/p_h$ for all jobs $j \in I$, a contradiction.

In case $t \geq t_I$, suppose that job i is processed at time t and job j is available. Hence, $r'_j \leq t \leq S_i + p_i \leq 2S_i$. The last inequality follows from the fact that the delayed ratio rule only starts a job after its processing time has passed. Since j does not start at its release time r'_j, we have $r'_j = 2r_j$. Therefore, $r_j = r'_j/2 \leq S_i$, and j was available when job i was started. Hence, $w_i/p_i \geq w_j/p_j$. □

Let σ^* be an optimal preemptive schedule for the instance I and the weighted completion time objective. We denote the mean busy times and completion times of a job j in a schedule π by M_j^π and C_j^π, respectively. The following equations are easily verified:

$$\sum_{j \in I} w_j C_j^\sigma = \sum_{j \in I} w_j M_j^\sigma + \frac{1}{2} \sum_{j \in I} w_j p_j, \tag{4.16}$$

$$\sum_{j \in I} w_j M_j^\sigma = \sum_{j \in I'} w_j M_j^{\sigma'} - \frac{1}{2} w_k p_k, \tag{4.17}$$

$$\sum_{j \in I} w_j M_j^{\sigma^*} \leq \sum_{j \in I} w_j C_j^{\sigma^*} - \frac{1}{2} \sum_{j \in I} w_j p_j. \tag{4.18}$$

From the preemptive schedule σ^* we define a "pseudo-schedule" by letting the machine process all jobs at half speed. That is, if a fraction of job j was previously scheduled in the interval $[a_j, b_j)$, it is now scheduled in $[2a_j, 2b_j)$. Consequently,

the mean busy time of each job increases by a factor of 2. We also process job k at half speed from time 0 to $2t_I$. This pseudo-schedule satisfies the release dates of the instance I', and we use it to derive a feasible preemptive schedule $\hat{\sigma}$ for I' with no further increase in any mean busy time. We can now put the pieces together:

$$\begin{aligned}
\sum_{j\in I} w_j C_j^{\sigma} &\overset{(4.16)}{=} \sum_{j\in I} w_j M_j^{\sigma} + \frac{1}{2}\sum_{j\in I} w_j p_j \\
&\overset{(4.17)}{=} \sum_{j\in I'} w_j M_j^{\sigma'} - \frac{1}{2} w_k p_k + \frac{1}{2}\sum_{j\in I} w_j p_j \\
&\overset{\text{Lemma 4.39}}{\leq} \sum_{j\in I'} w_j M_j^{\hat{\sigma}} - \frac{1}{2} w_k p_k + \frac{1}{2}\sum_{j\in I} w_j p_j \\
&\leq 2\sum_{j\in I} w_j M_j^{\sigma^*} + \frac{1}{2}\sum_{j\in I'} w_j p_j \\
&\overset{(4.18)}{\leq} 2\sum_{j\in I} w_j C_j^{\sigma^*} - \frac{1}{2}\sum_{j\in I} w_j p_j + \frac{1}{2} w_k p_k \\
&\overset{(4.15)}{\leq} 2\sum_{j\in I} w_j C_j^{\sigma^*}.
\end{aligned}$$

Theorem 4.40. *The delayed ratio rule is a deterministic online* 2*-approximation algorithm for* $1|r_j|\sum w_jC_j$.

Exercises

4.39. Define the mean busy time M_S of a subset $S \subseteq N$ of jobs as the average point in time that the machine is busy scheduling a job from S. Show that $\sum_{j\in S} p_j M_S = \sum_{j\in S} p_j M_j$.

4.40. Give an example for $1|r_j|\sum w_jC_j$ that shows that scheduling the jobs nonpreemptively in order of nondecreasing mean busy times in the optimal preemptive ratio schedule can lead to a schedule that is arbitrarily worse than the optimal one.

4.41. Prove that the preemptive ratio rule is a 2-approximation algorithm for $1|r_j, pmtn|\sum w_jC_j$. Show that this analysis is tight.

4.42. Give an example that shows that the randomized α_j-point algorithm can return an exponential number of different schedules.

4.43. Use the method of conditional probabilities to derandomize the randomized α_j-point algorithm.

Notes and references

4.1. *Smith's Ratio Rule.* The ratio rule is, of course, due to W. E. Smith (1956). The observation

> given two nondecreasing sequences of numbers, $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_n$, the permutation $\sigma = (n, n-1, \ldots, 2, 1)$ minimizes $\sum_{j=1}^{n} a_j b_{\sigma(j)}$ among all permutations of $(1, 2, \ldots, n)$

used in the alternate justification of the SPT rule, appears in Hardy, Littlewood and Polya (1966). Two-dimensional Gantt charts go at least as far back as Eastman, Even and Isaacs (1964). Goemans and Williamson (2000) showed that many results in single-machine scheduling have a simple and intuitive geometric justification using 2D Gantt charts. Mean busy times were introduced by Goemans, Queyranne, Schulz, Skutella and Wang (2002) in the context of linear programming relaxations and approximation algorithms for $1|r_j|\sum w_jC_j$; see Sections 4.5 and 4.10 for further details. Exercise 4.3 is due to Goemans and Williamson (2000). The equivalence in this exercise was shown by Chudak and Hochbaum (1999) and, for the case all $p_j = 1$, von Arnim, Faigle and Schrader [1990].

4.2. *Preference Orders on Jobs.* When Smith (1956) defined a preference order, he did not require the relation to be transitive. This has the consequence that there might not exist a total order consistent with the relation. However, for relations for which there does exist a consistent total order, Smith's definition and the one given in this section are equivalent. Jackson (1955) showed that the earliest due date rule is optimal for $1|\ |L_{\max}$. The total weighted discounted completion time criterion was introduced by Rothkopf (1966). The least cost fault detection problem was studied by Boothroyd (1960), Mitten (1960]), Mankekar and Mitten (1965), and Garey (1966), among others. Monotone cost density functions were introduced by Lawler and Sivazlian (1978). The cumulative cost problem was formulated by Addel-Wahab and Kameda (1978) and generalized by Monma (1980).

4.3. *Preference Orders on Sequences and Series-Parallel Precedence Constraints.* Monma and Sidney (1979) introduced the notion of preference relations on sequences. Parallel-chains precedence constraints were treated by Conway, Maxwell and Miller (1967). Horn (1972), Adolphson and Hu (1973), and Sidney (1975) gave algorithms for tree-like precedence constraints. Lawler (1978) presented the $O(n\log n)$ algorithm for series-parallel precedence constraints. Goemans and Williamson (2000) used 2D Gantt charts and linear programming duality to give an alternative proof of correctness of Lawler's algorithm for $1|prec|\sum w_jC_j$. The equivalent characterization of series-parallel partial orders in terms of a forbidden substructure is due to Duffin (1965). Valdes, Tarjan and Lawler (1982) gave a linear-time algorithm that recognizes series-parallel partial orders and creates a decomposition tree.

4.4. *NP-Hardness of Further Constrained Min-Sum Problems.* The strong NP-hardness of $1|r_j|\sum C_j$ was established by Lenstra, Rinnooy Kan and Brucker (1977). The proofs presented here for this result and the NP-hardness of $1|r_j, pmtn|\sum w_jC_j$ are due to Lenstra. The latter result was originally obtained by Labetoulle, Lawler, Lenstra and Rinnooy Kan (1984). Lawler (1978) showed that $1|prec|\sum w_jC_j$ is NP-hard, even if either all $w_j = 1$ or all $p_j = 1$. The proof of Theorem 4.15 follows that of Lenstra and Rinnooy Kan (1978).

4.5. *The Ratio Rule via Linear Programming.* The parallel inequalities were con-

ceived by Wolsey (1985) and Queyranne (1993), who also showed that these inequalities completely describe the convex hull of feasible completion time vectors for $1||\sum w_jC_j$. The relationship between Smith's ratio rule and optimizing a linear function over the polyhedron defined by the parallel inequalities is laid out further in Queyranne (1993) and Queyranne and Schulz (1994). The separation algorithm for the parallel inequalities discussed in Exercise 4.21 appeared in Queyranne (1993). The "shifted" parallel inequalities in Exercises 4.22 and 4.23 were introduced by Queyranne (1988).

4.6. *Approximation Algorithms for* $1|prec|\sum w_jC_j$. Hall, Schulz, Shmoys and Wein (1997) devised linear programming based approximation algorithms to yield the first constant performance guarantees for a variety of minimum-sum single-machine scheduling problems, including Theorem 4.16, and gave the tight examples asked for in Exercise 4.26. Potts (1980) suggested the integer programming formulation in δ-variables, and Wolsey (1990) showed that any feasible solution to its linear programming relaxation satisfies the parallel inequalities and precedence constraints, which implies Corollary 4.17. A family of instances provided by Chekuri and Motwani (1999) demonstrates that the integrality gap of this linear programming relaxation is essentially 2 (Exercise 4.27). The first part of Exercise 4.24 is due to Peters (1988) and Nemhauser and Savelsbergh (1992). Potts (1980) showed that the linear programming relaxation of his integer programming formulation with the transitivity constraints dropped can be used to obtain an alternative derivation of Smith's ratio rule, which corresponds to the second part of this exercise. 2-approximation algorithms for $1|prec|\sum w_jC_j$ that do not rely on solving a linear program were proposed by Chekuri and Motwani (1999), Chudak and Hochbaum (1999), and Margot, Queyranne and Wang (2003).

4.7. *Sidney Decompositions.* The concept of Sidney decomposition is, of course, due to J. B. Sidney (1975). The proof of Theorem 4.18 presented herein is adapted from Correa and Schulz (2005). Chekuri and Motwani (1999) and Margot, Queyranne and Wang (2003) observed that each sequence that is consistent with a Sidney decomposition is within a factor of 2 from optimum. The 2D Gantt chart illustration of this fact is due to Goemans and Williamson (2000). Correa and Schulz (2005) noted that all known 2-approximation algorithms are of this type. Exercise 4.30 is based on an observation by Margot, Queyranne and Wang (2003).

4.8. *An Integrality Theorem for Series-Parallel Precedence Constraints.* This section is largely based on Correa and Schulz (2005). Chudak and Hochbaum (1999) suggested to drop all transitivity constraints associated with triples of unrelated jobs from Potts' linear programming relaxation. They observed that the resulting linear program is half-integral and can be solved by a min-cut computation, and used this insight to develop a combinatorial 2-approximation algorithm for the single-machine problem with general precedence constraints. Two-dimensional partial orders were introduced by Dushnik and Miller (1941), who also proved that a partial order is

two-dimensional if and only if it has a nonseparating linear extension. The first polynomial-time recognition algorithm for 2D orders was presented by Pnueli, Lempel and Even (1971). Exercise 4.32, which is needed to complete the proof of Theorem 4.24, is based on a conjecture by Correa and Schulz (2005) that was proved by Ambühl and Mastrolilli (2006).

4.9. *Approximation Algorithms for* $1|r_j|\sum C_j$. The optimality of the SRPT rule for $1|r_j, pmtn|\sum C_j$ was shown by Baker (1974). Phillips, Stein and Wein (1998) introduced the idea of scheduling jobs in order of their preemptive completion times into the design of approximation algorithms. Theorem 4.26 belongs to them. The lower bound of 2 on the performance guarantee of any deterministic online algorithm is due to Hoogeveen and Vestjens (1996). Chekuri, Motwani, Natarajan and Stein (2001) presented the randomized α-point algorithm and proved Theorem 4.32. Goemans et al. (2002) showed that the sample space of this algorithm is $O(n)$, which makes the straightforward derandomization possible. The construction of a matching lower bound on the performance guarantee of any randomized online algorithm can be found in Vestjens (1997).

4.10. *Mean Busy Times Revisited & Approximation Algorithms for* $1|r_j|\sum w_jC_j$. The randomized α_j-point algorithm was suggested by Goemans et al. (2002). They also showed that drawing the α_j's according to a truncated exponential distribution results in an expected performance guarantee of 1.69. Working with the same α for all jobs complicates the analysis, and the best density function known leads to an expected performance guarantee of 1.75. Theorem 4.40 is due to Anderson and Potts (2004). The proof presented here follows an interpretation by Sitters (2005) of a simplified proof suggested by Queyranne (2002). Exercises 4.39, 4.42 and 4.43 are from Goemans et al. (2002). The tight example sought in Exercise 4.41 was presented by Schulz and Skutella (2002). Therein, they also proved the upper bound, which is due to Goemans, Wein and Williamson (1997).