Contents

7

Minsum criteria

Eugene L. Lawler
University of California, Berkeley

This is a fragmented chapter. Our focus is on the nonpreemptive and preemptive minimization of total completion time in polynomial time. We also present a pseudopolynomial-time algorithm for the minimization of total weighted completion time and related problems. We review NP-hardness results without providing proofs, and give pointers to the very few results on approximation and branch-and-bound.

7.1. Total completion time, nonpreemptive scheduling

In the absence of precedence constraints, we know that whatever jobs are performed on a given machine in an optimal schedule, they will be performed on that machine in SPT order. Recall formula (4.1) derived in Chapter 4:

$$\sum C_j = np_1 + (n-1)p_2 + \cdots + p_n.$$

We can solve the problem $P||\sum C_j$ by *coefficient matching*: There are m 1's, m 2's, ..., m $(n-1)$'s, and m n's to match with n p_j values so as to obtain the smallest possible weighted sum. It is clear that an optimal solution to this matching problem is obtained by matching the m 1's with the m largest p_j's, the m 2's with the m next-largest p_j's, and so forth, ending up by matching one of the coefficients $\lceil n/m \rceil$ with the smallest p_j.

Thus an optimal schedule for $P||\sum C_j$ is as shown in Figure 7.1. One can also state an extended SPT rule for identical parallel machines, as follows: Schedule the jobs in time, with a decision point at the completion of a job. At each decision point choose to process the shortest job that has not yet been assigned to a machine.

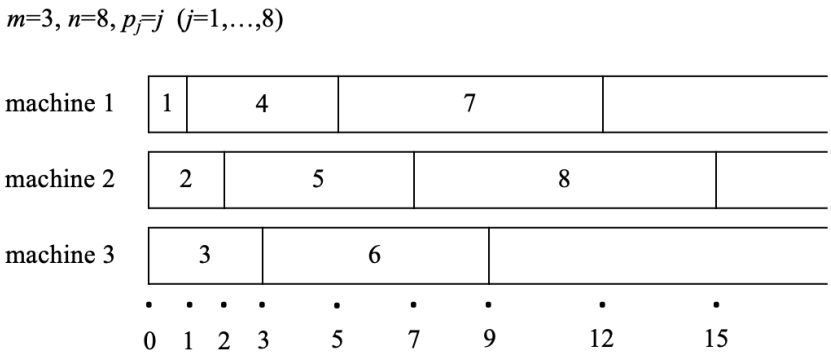

Figure 7.1. An optimal schedule for $P||\sum C_j$.

In the case of $Q||\sum C_j$, we notice that for the jobs assigned to machine i the formula takes on the form:

$$\sum C_j = (n/s_i)p_1 + ((n-1)/s_i)p_2 + ... + (1/s_i)p_n.$$

Like $P||\sum C_j$, the problem $Q||\sum C_j$ can be solved by coefficient matching, except that now the coefficients are of the form k/s_i. The largest p_j should be matched with the smallest of these coefficients ... and the smallest p_j should be matched with the nth-largest of these coefficients.

Here is how to generate the coefficients k/s_i in nondecreasing order of size: Place the fractions $1/s_i$, $i = 1,2,...,m$, into a priority queue Q supporting the operations of **delete-min** and **insert**. Then

for $h = 1,2,\ldots,n$ **do**
  output $k/s_i :=$ **delete-min**(Q);
  insert $(k+1)/s_i$ into Q;
end

Clearly it is possible to match the coefficients and construct an optimal schedule in $O(n\log n)$ time.

The problem $R||\sum C_j$ can still be solved by coefficient matching. Note that, when job j is scheduled in the kth last position on machine i, it contributes kp_{ij} to the total completion time. We now describe schedules in terms of 0-1 variables $x_{(ik)j}$, where $x_{(ik)j} = 1$ if job j is the kth last job processed on machine i, and $x_{(ik)j} = 0$, otherwise. The problem is then to minimize

$$\sum_{i,k}\sum_{j} kp_{ij}x_{(ik)j}$$

subject to

$$\begin{aligned}
\sum_{i,k} x_{(ik)j} &= 1, \quad \text{for } j = 1,\ldots,n; \\
\sum_{j} x_{(ik)j} &\leq 1, \quad \text{for } i = 1,\ldots,m,\ k = 1,\ldots,n; \\
x_{(ik)j} &\in \{0,1\}, \quad \text{for } i = 1,...,m,\ j,k = 1,...,n.
\end{aligned}$$

The constraints ensure that each job is scheduled exactly once and that each position on each machine is occupied by at most one job. This is a weighted bipartite matching or assignment problem, so that we may replace the integrality constraints by nonnegativity constraints without altering the feasible set. It can be solved in $O(n^3)$ time.

Exercises

7.1. Let us say that two schedules for $P||\sum C_j$ are equivalent if they differ only by a renumbering of machines. Show that there exists at least

$$(m!)^{\lfloor (n/m) \rfloor - 1}$$

nonequivalent optimal schedules.

7.2. Show that the SPT rule solves $P||\sum w_j C_j$ for the special case in which processing times and weights are oppositely ordered.

7.3. Suppose that machine i is not available until time $t_i \geq 0$, i.e., it must remain idle until then. Show how to incorporate this condition into the matching problem formulated for $R||\sum C_j$.

7.4. Show how to reduce the time required to set up the matching problem for $R||\sum C_j$ to $O(n^2 \log n)$ time. Hint: For each job j, one need only consider the smallest n coefficients of the form kp_{ij}. For each job j establish a priority queue Q_j to generate the n smallest coefficients of this form.

7.2. Total completion time, preemptive scheduling

We now turn to preemptive scheduling of parallel machines with respect to the $\sum C_j$ criterion. It turns out that there is no advantage to preemption for the problem $P|pmtn|\sum C_j$ (nor for $P|pmtn|\sum w_j C_j$ – see Section 7.3). That is, a schedule that is optimal for an instance of $P||\sum C_j$ is also optimal for a similar instance of $P|pmtn|\sum C_j$.

The problem $R|pmtn|\sum C_j$ is NP-hard in the strong sense. This is a surprising result, in view of the polynomial-time solvability of $R||\sum C_j$. There are very few problems for which allowing preemption makes the problem harder.

This leaves us with only the problem $Q|pmtn|\sum C_j$ to consider in this section. Knowing what we do about the $\sum C_j$ criterion, the SPT rule, and the solution of

the problem $Q||\sum C_j$, what might we conjecture about the structure of an optimal schedule for $Q|pmtn|\sum C_j$? Our first, rather timid, guess might be that jobs will be completed in SPT order. This conjecture is indeed true, though its demonstration is nontrivial, as we shall see in proving Lemma 7.1. Our next guess might be to try the following greedy prescription:

Preemptive SPT rule: Taking the jobs in SPT order, preemptively schedule each successive job j in the available time on the m machines so as to minimize C_j.

This rule produces a schedule that looks like that shown in Figure 7.2. That is, job 1, the shortest job, is processed entirely on M_1, the fastest machine. Job 2, the second-shortest job, is processed on M_2, the second-fastest machine, until job 1 is completed. Then job 2 is processed on M_1 to completion. In general, job j is scheduled for processing first on M_m, then on M_{m-1},..., and finally on M_1. This makes C_j as small as possible, given the time left available on the machines after jobs $1,2,\ldots,j-1$ have been scheduled.

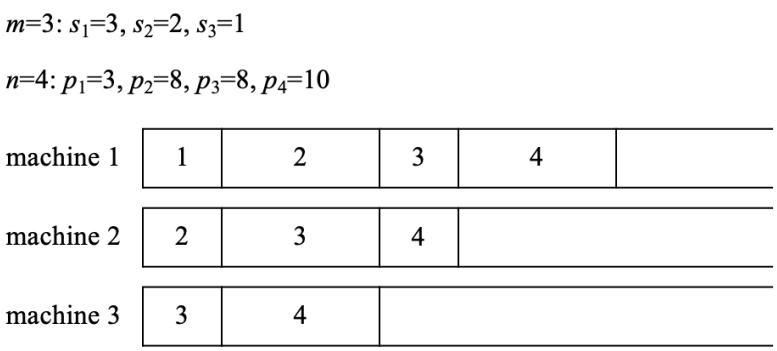

Figure 7.2. Preemptive SPT schedule for $Q|pmtn|\sum C_j$.

The procedure we have described requires $O(n\log n)$ time for the initial sort of the jobs into SPT order and $O(mn)$ time to construct the schedule. The schedule has at most $(m-1)(n-m/2)$ preemptions (see Exercise 7.6), and no preemptions at all if the machines are identical. This latter observation proves our assertion that there is no advantage to preemption in the case of $P|pmtn|\sum C_j$ – provided the schedule constructed by the preemptive SPT rule is indeed optimal. We shall now prove the optimality of the schedule.

Lemma 7.1. *For any instance of the $Q|pmtn|\sum C_j$ problem, there exists an optimal schedule in which the completion times of the jobs are in SPT order.*

Proof. Let S be an optimal schedule in which $C_i > C_j$ with $p_i < p_j$. Prior to time C_j there are some periods during which job i but not j is processed, some periods in which job j but not i is processed, and some periods in which both jobs i and j are processed. Also, in the interval $[C_j, C_i]$ there are some periods in which job i is

processed. We propose to replace the processing of job i in the interval $[C_j, C_i]$ with processing of job j, and to interchange some fraction λ, $0 < \lambda < 1$, of the processing of i and j prior to time C_j, thereby obtaining a feasible schedule S' with completion times C'_i and C'_j. Of each active period for job i prior to C_j, a fraction λ of that active period is given over to job j in S', and vice versa. Observe that $C'_i \leq C_j$ and $C'_j = C_i$.

We must now show that there is a value of λ in the interval $(0,1)$ such that the schedule S' can be constructed as we have suggested. Let t_i, t_j denote the total lengths of time, and let σ_i, σ_j denote the effective rates at which jobs i and j are processed prior to time C_j in schedule S. Let t'_i be the total length of time and σ'_i be the effective rate at which job i is processed in the interval $[C_j, C_i]$. Then we have

$$\sigma_i t_i + \sigma'_i t'_i = p_i < p_j = \sigma_j t_j. \tag{7.1}$$

In order for S' to be feasible, we must have

$$(1-\lambda)\sigma_i t_i + \lambda \sigma_j t_j = p_i,$$

$$(1-\lambda)\sigma_j t_j + \lambda \sigma_i t_i + \sigma'_i t'_i = p_j.$$

Solving for λ, we obtain

$$\lambda = \frac{\sigma'_i t'_i}{\sigma_j t_j - \sigma_i t_i}$$

It follows immediately from (7.1) that

$$\sigma_j t_j - \sigma_i t_i > \sigma'_i t'_i > 0,$$

and hence $0 < \lambda < 1$.

Note that $C'_i + C'_j \leq C_i + C_j$, and the completion times of all other jobs are the same in S' as in S. It follows that a finite number of modifications of the type we have described will transform S into an optimal schedule with completion times in SPT order. □

Theorem 7.2. *The preemptive SPT rule computes an optimal schedule for* $Q|pmtn|\sum C_j$.

Proof. For convenience, assume that $n = m$, and that $s_1 \geq s_2 \geq \cdots \geq s_m$. (If $n < m$, discard the $m-n$ slowest machines, and if $n > m$, create $n-m$ zero-speed dummy machines.) Let S be the schedule computed by the preemptive SPT rule, with $C_1 \leq C_2 \leq \cdots \leq C_n$. From the structure of the schedule S, it is apparent that

$$\begin{aligned} s_1 C_1 &= p_1, \\ s_2 C_1 + s_1 (C_2 - C_1) &= p_2, \\ s_3 C_1 + s_2 (C_2 - C_1) + s_1 (C_3 - C_2) &= p_3, \end{aligned}$$

and so forth. Adding these equations yields

$$\begin{aligned} s_1C_1 &= p_1, \\ s_2C_1 + s_1C_2 &= p_1 + p_2, \\ s_3C_1 + s_2C_2 + s_1C_3 &= p_1 + p_2 + p_3, \end{aligned}$$

and so forth.

Suppose S^* is an optimal schedule. By Lemma 7.1 we may assume $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$. For each j, $1 \leq j \leq n$, let us consider a lower bound on the value of C_j^*. The amount of processing done prior to C_j^* is at least $p_1 + p_2 + \cdots + p_j$. The fastest way to perform this processing is to use machines $1,2,\ldots,j$ before C_1^*, machines $1,2,\ldots,j-1$ in the interval $[C_1^*, C_2^*]$, etc. This yields the system of inequalities of the form

$$\begin{aligned} &(s_1 + s_2 + \cdots + s_j)C_1^* + (s_1 + s_2 + \cdots + s_{j-1})(C_2^* - C_1^*) + \cdots + s_1(C_j^* - C_{j-1}^*) \\ &\quad \geq p_1 + p_2 + \cdots + p_j. \end{aligned}$$

Adding these inequalities, we obtain,

$$\begin{aligned} s_1C_1^* &\geq p_1, \\ s_2C_1^* + s_1C_2^* &\geq p_1 + p_2, \\ s_3C^*1 + s_2C_2^* + s_1C_3^* &\geq p_1 + p_2 + p_3, \end{aligned}$$

and so forth, which gives us

$$\begin{aligned} s_1C_1^* &\geq s_1C_1, \\ s_2C_1^* + s_1C_2^* &\geq s_2C_1 + s_1C_2, \\ s_3C_1^* + s_2C_2^* + s_1C_3^* &\geq s_3C_1 + s_2C_2 + s_1C_3, \end{aligned}$$

and so forth. Now let us multiply the jth inequality above by a positive value λ_j and add the inequalities, thereby obtaining

$$\begin{aligned} &(\lambda_1 s_1 + \lambda_2 s_2 + \cdots + \lambda_n s_n)C_1^* + (\lambda_2 s_1 + \cdots + \lambda_n s_{n-1})C_2^* + \cdots + \lambda_n s_1 C_1^* \\ \geq\ &(\lambda_1 s_1 + \lambda_2 s_2 + \cdots + \lambda_n s_n)C_1 + (\lambda_2 s_1 + \cdots + \lambda_n s_{n-1})C_2 + \cdots + \lambda_n s_1 C_n. \end{aligned}$$

If the λ_j values are chosen so that

$$\begin{aligned} \lambda_n s_1 &= 1, \\ \lambda_{n-1} s_1 + \lambda_n s_2 &= 1, \\ \lambda_{n-2} s_1 + \lambda_{n-1} s_2 + \lambda_n s_3 &= 1, \end{aligned} \tag{7.2}$$

then we will have

$$\sum C_j^* \geq \sum C_j,$$

from which it follows that S, the schedule determined by the preemptive SPT rule, is optimal. Indeed, it follows from the fact that $s_1 \geq s_2 \geq \cdots \geq s_m$ that there is a solution to (7.2) in nonnegative λ_j's. □

Exercises

7.5. Devise a simple example to show that an optimal schedule for $Q|pmtn|\sum C_j$ may require preemptions.

7.6. Show that the number of preemptions created by applying the preemptive SPT rule does not exceed $(m-1)(n-m/2)$. Suggestion: prove that the number of preemptions is equal to the number of active periods $-\ n$.

7.7*. Formulate a general class of problem instances for $Q|pmtn|\sum C_j$ to show that, for any m and n, as many as $(m-1)(n-m/2)$ preemptions may be required for an optimal schedule. Hint: Find a class of problem instances with distinct machine speeds and job processing requirements for which the only schedule that is optimal is the one determined by the preemptive SPT rule.

7.8. Generalize Lemma 7.1 and its proof to apply to the $\sum w_j C_j$ criterion, in the special case that job processing times and weights are oppositely ordered.

7.9. Show that Lemma 7.1 and its proof remain valid for $Q|pmtn,\bar{d}_j|\sum C_j$, provided processing times and deadlines are similarly ordered.

7.10. Generalize Theorem 7.2 and its proof to apply to the $\sum w_j C_j$ criterion, in the special case that job processing times and weights are oppositely ordered.

7.11*. Obtain a closed form expression for C_j when the preemptive SPT rule is applied to an instance of $Q|pmtn|\sum C_j$.

7.3. Total weighted completion time, complexity

A summary of the results of the previous two sections is as follows: $P||\sum C_j$ can be solved in $O(n\log n)$ time by a simple extension of the SPT rule, and since there is no advantage to preemption, the same rule solves $P|pmtn|\sum C_j$ as well. $Q||\sum C_j$ can be solved in $O(n\log n)$ time by matching the processing times p_j in nonincreasing order with the coefficients k/s_i in nondecreasing order. $R||\sum C_j$ can be solved in $O(n^3)$ time by a generalization of the coefficient-matching technique. $Q|pmtn|\sum C_j$ can be solved in $O(n\log n + mn)$ time by the preemptive SPT rule. $R|pmtn|\sum C_j$ is NP-hard.

A summary of the complexity status of generalizations of the problems studied in the previous sections is as follows:

- $\sum w_j C_j$ criterion – $P2||\sum w_j C_j$ is NP-hard. It can be shown that there is no advantage to preemption for $P|pmtn|\sum w_j C_j$. Hence, this NP-hardness result applies to $P2|pmtn|\sum w_j C_j$ as well. Dynamic programming provides a pseudopolynomial algorithm for $Qm||\sum w_j C_j$, as we show below, but this is of very limited practicality. Results on approximation and branch-and-bound for $P||\sum w_j C_j$ are scarce.

- Release dates and deadlines – In the absence of preemption, either release dates or deadlines induce NP-hardness; both $1|r_j|\sum C_j$ and $P2||C_{\max}$ have already been shown to be NP-hard. Also, $P2|pmtn, r_j|\sum C_j$ has been shown to be NP-hard. The status of $P|pmtn, \bar{d}_j|\sum C_j$ is unknown. However, there are polynomial-time algorithms for solving $Q|pmtn, \bar{d}_j = \bar{d}|\sum C_j$.

- Precedence constraints – There is very little that can be done with precedence constraints in polynomial time, since even $P2|chains|\sum C_j$ is NP-hard. Also, $P|prec, p_j = 1|\sum C_j$ is NP-hard, but $P|outtree, p_j = 1|\sum C_j$ can be solved in polynomial time.

Let us now establish the existence of a pseudopolynomial algorithm for the problem $Qm||\sum w_j C_j$. The dynamic programming technique we shall describe also yields similar results for the problems $Rm||C_{\max}$, $Rm||L_{\max}$, $Rm||\sum w_j U_j$, and even $Rm|\bar{d}_j = \bar{d}|\sum w_j T_j$. What all these problems have in common is that there exists an optimal schedule in which the sequence of jobs performed by each of the m machines is consistent with a permutation π of the n jobs that can be prescribed *a priori*. (For $\sum w_j C_j$, π is a ratio order; for $L_{\max}$ and $\sum w_j U_j$, π is an EDD order; for $C_{\max}$, π is any order whatsoever. In the case of $\sum w_j U_j$, only the on-time jobs are in a sequence consistent with π.) Without loss of generality, let $\pi = (1, 2, ..., n)$. Define $F_j(t_1, \ldots, t_m)$ to be the minimum cost of a schedule for the jobs $1, 2, \ldots, j$, subject to the constraint that the last job on machine i is completed at time t_i, for $i = 1, 2, ..., m$. Then, for $f_{\max}$ criteria we have

$$F_j(t_1, \ldots, t_m) = \min_{i=1,\ldots,m} \max\{f_j(t_i), F_{j-1}(t_1, \ldots, t_i - p_{ij}, \ldots, t_m)\},$$

and for $\sum f_j$ criteria,

$$F_j(t_1, \ldots, t_m) = \min_{i=1,\ldots,m}\{f_j(t_i) + F_{j-1}(t_1, \ldots, t_i - p_{ij}, ..., t_m)\}.$$

In both cases we have initial conditions

$$F_0(t_1, ..., t_m) = \begin{cases} 0, & \text{if } t_i = 0, \text{ for } i = 1, 2, \ldots, m, \\ +\infty, & \text{otherwise.} \end{cases}$$

There are $O(nC^m)$ values of $F_j(t_1, \ldots, t_m)$ to compute, where C is an upper bound on the completion time of any job in an optimal schedule. The computation of each value requires minimization over m alternatives and hence $O(m)$ time. The cost of an optimal schedule is given by the minimum of the values $F_n(t_1, \ldots, t_m)$, where $t_i \leq C$. Hence, the cost of an optimal schedule can be computed in $O(mnC^m)$ time, which is pseudopolynomial for fixed m. For variations on this dynamic programming scheme, see the exercises below.

Exercises

7.7. Devise a simple example to show that an optimal schedule for $P2|pmtn, r_j|\sum C_j$ may require preemptions.

7.8. A time bound of $O(mnC^m)$ is indicated for the dynamic programming algorithm presented in this section, where C is an upper bound on the completion time of any job in an optimal schedule. Show that $C \leq \sum p_j/m + p_{\max} \leq \lceil (n/m) \rceil p_{\max}$, and hence the time bound can be expressed as $O(n^2 p_{\max} C^m)$.
7.9. Modify the dynamic programming procedure, as necessary, to solve $R||\sum w_j U_j$.
7.10. Show that in the case of uniform machines and the $C_{\max}$, $L_{\max}$, $\sum w_j C_j$ criteria, the dynamic programming running time bound of $O(mnC^m)$ can be reduced to $O(mnC^{m-1})$. Why can this not be done in the case of the $\sum w_j U_j$ criterion?
7.11. Consider the problems $Qm||\sum w_j C_j$, $Rm||C_{\max}$, $Rm||L_{\max}$, $Rm||\sum w_j U_j$, and $Rm|\bar{d}_j = \bar{d}|\sum w_j T_j$.

(a) For which of these problems is it possible to adapt the dynamic programming algorithm to deal with nonuniform release dates? Explain.

(b) Same question, with respect to deadlines.

(c) Both release dates and deadlines?

7.12. Recall the recurrence relations for the dynamic programming solution of the problem $1||\sum f_j$:

$$F(S) = \min_{j \in S} \{ f_j(p(S)) + F(S-j) \}. \tag{7.3}$$

As an alternative to the dynamic programming technique described in this section, let us consider two ways in which we might adapt the recurrence (7.3) to the solution of the problem $R||\sum f_j$.

(a) Let $F_i(S)$ denote the minimum cost of a sequence for the subset S on machine i. Use recurrences (7.3) to compute $F_i(S)$ for each $i = 1, 2, \ldots, m$, and for all $S \subseteq N$. Let $G_i(S)$ denote the minimum cost for a schedule of the subset S on machines $1, 2, ..., i$, where

$$G_i(S) = \min_{S' \subseteq S} \{ F_i(S') + G_{i-1}(S - S') \}.$$

Show that the time required to compute $G_m(N)$, the cost of an optimal schedule for all n jobs on all m machines, is $O(m3^n)$.

(b) View the problem of finding an optimal schedule as that of constructing a single optimal sequence obtained by concatenating the m sequences for the individual machines. (A similar concatenation of sequences was contemplated for the solution of Exercise 1.2 in Chapter 1, in which the reader was asked to show that there are $(n+m-1)!/(m-1)!$ distinct schedules for m parallel machines.) Define $F_i(S,t)$ to be the minimum cost of a sequence for the subset S in which the last job in the sequence finishes at time t on machine i. Then we have as our basic recurrence relations:

$$F_i(S,t) = \min_{j \in S} \{ f_j(t) + F_i(S-j, t-p_{ij}) \}.$$

Indicate how to obtain $F_i(S,0)$ from values for $F_i(S,t)$ and supply appropriate initial conditions. Show that the time required to compute the cost of an optimal schedule is $O(mnC2^n)$, where C is an upper bound on the completion time of any job in an optimal schedule. Show that the time bound can be expressed as $O(n^2 p_{\max} 2^n)$.

(c) For what value of n can you reasonably expect $mnC2^n$ to be smaller than $m3^n$?

(d) Describe the adaptations in the procedure necessary to solve $1|r_j,\bar{d}_j|\sum f_j$.

7.4. Other minsum criteria, complexity

Having pretty well disposed of parallel machine problems with $\sum C_j$ and $\sum w_j C_j$ criteria, let us consider the complexity situation with respect to other minsum criteria. What we have is largely a litany of bad news:

- $\sum U_j$, $\sum w_j U_j$ – $P2||\sum U_j$ is NP-hard, as demonstrated by a simple transformation from SUBSET SUM. As we have seen in Section 7.3, dynamic programming yields a pseudopolynomial algorithm for $Rm||\sum w_j U_j$. $P|pmtn|\sum U_j$ turns out to be NP-hard in the ordinary sense; the proof is of interest because the problem is one for which preemption *is* advantageous, and the proof must take this into account. NP-hardness of the problem in the strong sense is an open question. It is possible to solve $Qm|pmtn|\sum w_j U_j$ in $O(n^3 m W^2)$ time, which is pseudopolynomial, but becomes polynomial, $O(n^3 m)$, when all $w_j = 1$.

- $\sum T_j$, $\sum w_j T_j$ – Since $1||\sum T_j$ and $1|pmtn|\sum T_j$ are NP-hard, all parallel machine problems with the $\sum T_j$ criterion are NP-hard as well. Furthermore, since $1||\sum w_j T_j$ and $1|pmtn|\sum w_j T_j$ are NP-hard in the strong sense, it is unreasonable even to hope for pseudopolynomial algorithms for parallel machine versions of these problems with fixed m. (Recall that in Section 7.3 the best we could offer was a pseudopolynomial algorithm for the special case of a common due date, $Rm|d_j = d|\sum w_j T_j$.)

- *Unit-time jobs* – Here is good news: The problem $Q|p_j = 1|\sum f_j$ can be solved in $O(n^3)$ time by a simple extension of the matching technique we applied to solve the problem $1|p_j = 1|\sum f_j$ in Section 2.1. Recall that we solved the single-machine unit-time problem by optimally matching the n given jobs $j = 1,2,\ldots,n$ with the n time slots $t = 1,2,\ldots,n$. This involved formulating and solving an assignment or matching problem for an $n \times n$ matrix with entries of the form $f_j(t)$. The only new wrinkle required to solve $Q|p_j = 1|\sum f_j$ in the same way is in the generation of the values of t for the n time slots to which the jobs matched.

  Because we assume the functions f_j are monotone, we know there is an optimal schedule with no unforced machine idle time. This means that if n_i

unit-length are processed on machine i, those jobs will be completed at times $t = 1/s_i, 2/s_i, \ldots, n_i/s_i$. More generally, the completion times of the jobs in an optimal schedule can be assumed to be the n smallest values of t generated by the algorithm we described in Section 7.1:

for $h = 1, 2, \ldots, n$ **do**
  output $k/s_i :=$ **delete-min**(Q);
  insert $(k+1)/s_i$ into Q;
end

When this loop was used to generate coefficients for the problem $Q||\sum C_j$ in Section 7.1, k denoted the kth-last position on machine i, whereas here k denotes the kth-earliest time slot and k/s_i is the completion time of the job processed in that slot.

For general f_j functions, $O(n^2)$ time is required to construct the data for the matching problem and $O(n^3)$ time is required to solve it. However, for certain scheduling objectives, the matching problem has a trivial solution. For example, in the case of $Q|p_j = 1|\sum w_j C_j$, the problem is simply that of matching the largest weight w_j with the smallest completion time, the second-largest w_j with the second-smallest completion time, and so on. Thus the time required to solve $Q|p_j = 1|\sum w_j C_j$ is only $O(n \log n)$, the time required to sort the w_j and to generate the values of t. For other examples, see the exercises.

Exercises

7.13. Describe how to solve $Q|r_j, p_j = 1|\sum C_j$ in $O(n \log n)$ time.
7.14. Describe how to solve $Q|p_j = 1|\sum U_j$ in $O(n \log n)$ time.
7.15. Describe how to solve $Q|p_j = 1|f_{\max}$ in $O(n^2)$ time.

Notes

7.1. *Total completion time, nonpreemptive scheduling.* The extension of the SPT rule to $P||\sum C_j$ is due to Conway, Maxwell, and Miller (1967). The solution of $Q||\sum C_j$ is due to Horowitz and Sahni (1976). $R||\sum C_j$ was formulated as a matching problem by Horn (1973) and by Bruno, Coffman, and Sethi (1974).

7.2. *Total completion time, preemptive scheduling.* Lemma 7.1 is due to Lawler (–)]; the surveys by Graham et al. (1979) and Lawler et al. (1993) erroneously attribute the result to Lawler and Labetoulle (1978). Theorem 7.2 was proved by Gonzalez (1977)]. Sitters (2005) established NP-hardness of $R|pmtn|\sum C_j$ by a reduction from 3-dimensional matching.

7.3. *Total weighted completion time, complexity.* NP-hardness of $P2||\sum w_j C_j$ was proved by Bruno, Coffman, and Sethi (1974); Lenstra, Rinnooy Kan, and Brucker

(1977) gave a simpler proof. McNaughton (1959) showed that there is no advantage to preemption in $P|pmtn|\sum w_jC_j$.

For $P||\sum w_jC_j$, an obvious idea is to apply list scheduling with the jobs listed according to nondecreasing ratios p_j/w_j. Eastman, Even, and Isaacs (1964) investigated the performance of this ratio rule and gave a lower bound on the optimum solution value. This lower bound has been the basis for the branch-and-bound algorithms of Elmaghraby and Park (1974), Barnes and Brennan (1977), and Sarin, Ahn, and Bishop (1988). Kawaguchi and Kyan (1986) refined the analysis of the ratio rule and gave a performance ratio of $(\sqrt{2}+1)/2$. Sahni (1976) gave algorithms A_k with running time $O(n(n^2k)^{(m-1)})$ and performance ratio $1+1/k$.

Gonzalez (1977) devised a complicated, but polynomial-time, algorithm for $Q|pmtn,\bar{d}_j=\bar{d}|\sum C_j$. As we note in Exercise 7.9, jobs are executed in SPT order for this problem. This makes it possible to give the problem a linear programming formulation, using ideas presented in Section 9.2. McCormick and Pinedo (1995) have investigated an LP formulation in which the objective is to minimize a linear combination of flow time and makespan, and gave an algorithm that minimizes flow time subject to a fixed makespan deadline. They showed how to generate the entire trade-off curve between total completion time and makespan. The schedules generated put the jobs with the shortest processing times on the fastest machines, except when it is necessary to fit a block of long jobs under the deadline.

NP-hardness of $P2|tree|\sum C_j$ and $P2|chain|\sum C_j$ was proved by Sethi (1977) and Du, Leung, and Young (1991), respectively. The dynamic programming formulation for $Qm||\sum w_jC_j$ is due to Rothkopf (1966) and Lawler and Moore (1969).

We note that this chapter was written in 1990. Many subsequent results on approximating the $\sum w_jC_j$ objective are not covered.

7.4. *Other minsum criteria, complexity.* NP-hardness of $P|pmtn|\sum U_j$ was proved by Lawler (1983). Also the pseudopolynomial-time algorithm for $Qm|pmtn|\sum w_jU_j$ is due to Lawler (1979A). Lawler (–) and Dessouky et al. (1990) noticed that $Q|p_j=1|\sum f_j$ and $Q|p_j=1|f_{\max}$, and variations of these problems, can be formulated and solved as matching problems. Lawler (1976A) proposed an $O(n\log n)$ algorithm for the special case of $P|p_j=1|\sum w_jU_j$.