An optimal tour of the graph of Figure 5.21(a) has length 35. A tour of this length can be constructed if we retain with each $g(i, S)$ the value of j that minimizes the right-hand side of (5.21). Let $J(i, S)$ be this value. Then, $J(1, \{2, 3, 4\}) = 2$. Thus the tour starts from 1 and goes to 2. The remaining tour can be obtained from $g(2, \{3, 4\})$. So $J(2, \{3, 4\}) = 4$. Thus the next edge is $\langle 2, 4\rangle$. The remaining tour is for $g(4, \{3\})$. So $J(4, \{3\}) = 3$. The optimal tour is 1, 2, 4, 3, 1. □

Let N be the number of $g(i, S)$'s that have to be computed before (5.20) can be used to compute $g(1, V - \{1\})$. For each value of $|S|$ there are $n - 1$ choices for i. The number of distinct sets S of size k not including 1 and i is $\binom{n-2}{k}$. Hence

$$N = \sum_{k=0}^{n-2} (n-1)\binom{n-2}{k} = (n-1)2^{n-2}$$

An algorithm that proceeds to find an optimal tour by using (5.20) and (5.21) will require $\Theta(n^2 2^n)$ time as the computation of $g(i, S)$ with $|S| = k$ requires $k - 1$ comparisons when solving (5.21). This is better than enumerating all $n!$ different tours to find the best one. The most serious drawback of this dynamic programming solution is the space needed, $O(n2^n)$. This is too large even for modest values of n.

EXERCISE

1. (a) Obtain a data representation for the values $g(i, S)$ of the traveling salesperson problem. Your representation should allow for easy access to the value of $g(i, S)$, given i and S. (i) How much space does your representation need for an n vertex graph? (ii) How much time is needed to retrieve or update the value of $g(i, S)$?

   (b) Using the representation of (a), develop an algorithm corresponding to the dynamic programming solution of the traveling salesperson problem.

   (c) Test the correctness of your algorithm using suitable test data.

5.10 FLOW SHOP SCHEDULING

Often the processing of a job requires the performance of several distinct tasks. Computer programs run in a multiprogramming environment are input and then executed. Following the execution, the job is queued for output and the output eventually printed. In a general flow shop we may have n jobs each requiring m tasks $T_{1i}, T_{2i}, \ldots, T_{mi}$, $1 \le i \le n$, to be performed. Task T_{ji} is to be performed on processor P_j, $1 \le j \le m$. The time required to complete task T_{ji} is t_{ji}. A schedule for the n jobs is an assignment of tasks to time intervals on the processors. Task T_{ji} must be assigned to processor P_j. No processor may have more than one task assigned to it in any time interval. Additionally, for any job i the processing of task T_{ji}, $j > 1$, cannot be started until task $T_{j-1,i}$ has been completed.

Example 5.27 Two jobs have to be scheduled on three processors. The task times are given by the matrix $\mathcal{J}$

$$\mathcal{J} = \begin{bmatrix} 2 & 0 \\ 3 & 3 \\ 5 & 2 \end{bmatrix}$$

Two possible schedules for the jobs are shown in Figure 5.22. □

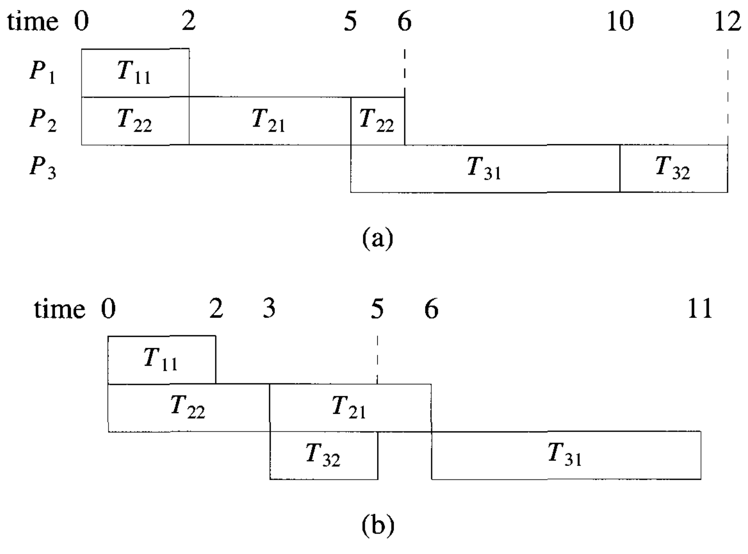

Figure 5.22 Two possible schedules for Example 5.27

A *nonpreemptive* schedule is a schedule in which the processing of a task on any processor is not terminated until the task is complete. A schedule for which this need not be true is called *preemptive*. The schedule of Figure 5.22(a) is a preemptive schedule. Figure 5.22(b) shows a nonpreemptive schedule. The *finish time* $f_i(S)$ of job i is the time at which all tasks of job i have been completed in schedule S. In Figure 5.22(a), $f_1(S) = 10$ and $f_2(S) = 12$. In Figure 5.22(b), $f_1(S) = 11$ and $f_2(S) = 5$. The finish time $F(S)$ of a schedule S is given by

$$F(S) = \max_{1 \le i \le n} \{f_i(S)\} \tag{5.22}$$

The *mean flow time* $\text{MFT}(S)$ is defined to be

$$\text{MFT}(S) = \frac{1}{n} \sum_{1 \le i \le n} f_i(S) \tag{5.23}$$

An optimal finish time (OFT) schedule for a given set of jobs is a nonpreemptive schedule S for which $F(S)$ is minimum over all nonpreemptive schedules S. A preemptive optimal finish time (POFT) schedule, optimal mean finish time schedule (OMFT), and preemptive optimal mean finish (POMFT) schedule are defined in the obvious way.

Although the general problem of obtaining OFT and POFT schedules for $m > 2$ and of obtaining OMFT schedules is computationally difficult (see Chapter 11), dynamic programming leads to an efficient algorithm to obtain OFT schedules for the case $m = 2$. In this section we consider this special case.

For convenience, we shall use a_i to represent t_{1i}, and b_i to represent t_{2i}. For the two-processor case, one can readily verify that nothing is to be gained by using different processing orders on the two processors (this is not true for $m > 2$). Hence, a schedule is completely specified by providing a permutation of the jobs. Jobs will be executed on each processor in this order. Each task will be started at the earliest possible time. The schedule of Figure 5.23 is completely specified by the permutation (5, 1, 3, 2, 4). We make the simplifying assumption that $a_i \ne 0$, $1 \le i \le n$. Note that if jobs with $a_i = 0$ are allowed, then an optimal schedule can be constructed by first finding an optimal permutation for all jobs with $a_i \ne 0$ and then adding all jobs with $a_i = 0$ (in any order) in front of this permutation (see the exercises).

It is easy to see that an optimal permutation (schedule) has the property that given the first job in the permutation, the remaining permutation is optimal with respect to the state the two processors are in following the completion of the first job. Let $\sigma_1, \sigma_2, \ldots, \sigma_k$ be a permutation prefix defining a schedule for jobs $T_1, T_2, \ldots, T_k$. For this schedule let f_1 and f_2 be the times at which the processing of jobs $T_1, T_2, \ldots, T_k$ is completed on processors P_1

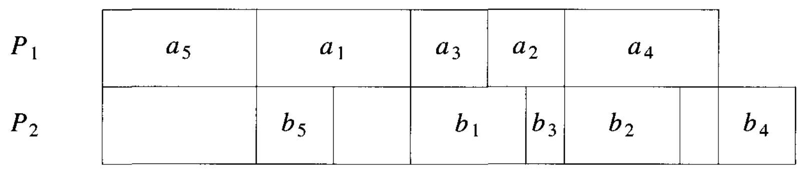

Figure 5.23 A schedule

and P_2 respectively. Let $t = f_2 - f_1$. The state of the processors following the sequence of decisions $T_1, T_2, \ldots, T_k$ is completely characterized by t. Let $g(S, t)$ be the length of an optimal schedule for the subset of jobs S under the assumption that processor 2 is not available until time t. The length of an optimal schedule for the job set $\{1, 2, \ldots, n\}$ is $g(\{1, 2, \ldots, n\}, 0)$.

Since the principle of optimality holds, we obtain

$$g(\{1, 2, \cdots, n\}, 0) = \min_{1 \le i \le n} \{a_i + g(\{1, 2, \cdots, n\} - \{i\}, b_i)\} \tag{5.24}$$

Equation 5.24 generalizes to (5.25) for arbitrary S and t. This generalization requires that $g(\phi, t) = \max\{t, 0\}$ and that $a_i \neq 0,\ 1 \le i \le n$.

$$g(S, t) = \min_{i \in S} \{a_i + g(S - \{i\}, b_i + \max\{t - a_i, 0\})\} \tag{5.25}$$

The term max $\{t - a_i, 0\}$ comes into (5.25) as task T_{2i} cannot start until $\max\{a_i, t\}$ (P_2 is not available until time t). Hence $f_2 - f_1 = b_i + \max\{a_i, t\} - a_i = b_i + \max\{t - a_i, 0\}$. We can solve for $g(S, t)$ using an approach similar to that used to solve (5.21). However, it turns out that (5.25) can be solved algebraically and a very simple rule to generate an optimal schedule obtained.

Consider any schedule R for a subset of jobs S. Assume that P_2 is not available until time t. Let i and j be the first two jobs in this schedule. Then, from (5.25) we obtain

$$\begin{aligned} g(S,t) &= a_i + g(S - \{i\}, b_i + \max\ \{t - a_i, 0\}) \\ g(S,t) &= a_i + a_j + g(S - \{i,j\}, b_j + \max\ \{b_i + \max\ \{t - a_i, 0\} - a_j, 0\}) \end{aligned} \tag{5.26}$$

Equation 5.26 can be simplified using the following result:

$$\begin{array}{rcl} t_{ij} & = & b_j + \max\ \{b_i + \max\ \{t - a_i, 0\} - a_j, 0\} \\ & = & b_j + b_i - a_j + \max\ \{\max\ \{t - a_i, 0\}, a_j - b_i\} \\ & = & b_j + b_i - a_j + \max\ \{t - a_i, a_j - b_i, 0\} \\ t_{ij} & = & b_j + b_i - a_j - a_i + \max\ \{t, a_i + a_j - b_i, a_i\} \end{array} \qquad (5.27)$$

If jobs i and j are interchanged in R, then the finish time $g'(S,t)$ is

$$g'(S,t) \quad = \quad a_i + a_j + g(S - \{i,j\}, t_{ji})$$

$$\text{where } t_{ji} \quad = \quad b_j + b_i - a_j - a_i + \max\ \{t, a_i + a_j - b_j, a_j\}$$

Comparing $g(S,t)$ and $g'(S,t)$, we see that if (5.28) below holds, then $g(S,t) \leq g'(S,t)$.

$$\max\ \{t, a_i + a_j - b_i, a_i\} \leq \max\ \{t, a_i + a_j - b_j, a_j\} \qquad (5.28)$$

In order for (5.28) to hold for all values of t, we need

$$\max\ \{a_i + a_j - b_i, a_i\} \leq \max\ \{a_i + a_j - b_j, a_j\}$$

$$\text{or } \ a_i + a_j + \max\ \{-b_i, -a_j\} \leq a_i + a_j + \max\ \{-b_j, -a_i\}$$

$$\text{or } \ \min\ \{b_i, a_j\} \geq \min\ \{b_j, a_i\} \qquad (5.29)$$

From (5.29) we can conclude that there exists an optimal schedule in which for every pair (i,j) of adjacent jobs, $\min\{b_i, a_j\} \geq \min\{b_j, a_i\}$. Exercise 4 shows that all schedules with this property have the same length. Hence, it suffices to generate any schedule for which (5.29) holds for every pair of adjacent jobs. We can obtain a schedule with this property by making the following observations from (5.29). If $\min\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\}$ is a_i, then job i should be the first job in an optimal schedule. If $\min\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\}$ is b_j, then job j should be the last job in an optimal schedule. This enables us to make a decision as to the positioning of one of the n jobs. Equation 5.29 can now be used on the remaining $n - 1$ jobs to correctly position another job, and so on. The scheduling rule resulting from (5.29) is therefore:

1. Sort all the a_i's and b_j's into nondecreasing order.

2. Consider this sequence in this order. If the next number in the sequence is a_j and job j hasn't yet been scheduled, schedule job j at the leftmost available spot. If the next number is b_j and job j hasn't yet been scheduled, schedule job j at the rightmost available spot. If j has already been scheduled, go to the next number in the sequence.

Note that the above rule also correctly positions jobs with $a_i = 0$. Hence, these jobs need not be considered separately.

Example 5.28 Let $n = 4, (a_1, a_2, a_3, a_4) = (3, 4, 8, 10)$, and $(b_1, b_2, b_3, b_4) = (6, 2, 9, 15)$. The sorted sequence of a's and b's is $(b_2, a_1, a_2, b_1, a_3, b_3, a_4, b_4) = (2, 3, 4, 6, 8, 9, 10, 15)$. Let $\sigma_1, \sigma_2, \sigma_3$, and σ_4 be the optimal schedule. Since the smallest number is b_2, we set $\sigma_4 = 2$. The next number is a_1 and we set $\sigma_1 = a_1$. The next smallest number is a_2. Job 2 has already been scheduled. The next number is b_1. Job 1 has already been scheduled. The next is a_3 and we set σ_3. This leaves σ_3 free and job 4 unscheduled. Thus, $\sigma_3 = 4$. □

The scheduling rule above can be implemented to run in time $O(n \log n)$ (see exercises). Solving (5.24) and (5.25) directly for $g(1, 2, \ldots, n, 0)$ for the optimal schedule will take $\Omega(2^n)$ time as there are these many different S's for which $g(S, t)$ will be computed.

EXERCISES

1. N jobs are to be processed. Two machines A and B are available. If job i is processed on machine A, then a_i units of processing time are needed. If it is processed on machine B, then b_i units of processing time are needed. Because of the peculiarities of the jobs and the machines, it is quite possible that $a_i \geq b_i$ for some i while $a_j < b_j$ for some j, $j \neq i$. Obtain a dynamic programming formulation to determine the minimum time needed to process all the jobs. Note that jobs cannot be split between machines. Indicate how you would go about solving the recurrence relation obtained. Do this on an example of your choice. Also indicate how you would determine an optimal assignment of jobs to machines.

2. N jobs have to be scheduled for processing on one machine. Associated with job i is a 3-tuple (p_i, t_i, d_i). The variable t_i is the processing time needed to complete job i. If job i is completed by its deadline d_i, then a profit p_i is earned. If not, then nothing is earned. From Section 4.4 we know that J is a subset of jobs that can all be completed by their

deadlines iff the jobs in J can be processed in nondecreasing order of deadlines without violating any deadline. Assume $d_i \leq d_{i+1}$, $1 \leq i < n$. Let $f_i(x)$ be the maximum profit that can be earned from a subset J of jobs when $n = i$. Here $f_n(d_n)$ is the value of an optimal selection of jobs J. Let $f_0(x) = 0$. Show that for $x \leq t_i$,

$$f_i(x) = \max \ \{f_{i-1}(x),\ f_{i-1}(x - t_i) + p_i\}$$

3. Let I be any instance of the two-processor flow shop problem.
   (a) Show that the length of every POFT schedule for I is the same as the length of every OFT schedule for I. Hence, the algorithm of Section 5.10 also generates a POFT schedule.
   (b) Show that there exists an OFT schedule for I in which jobs are processed in the same order on both processors.
   (c) Show that there exists an OFT schedule for I defined by some permutation σ of the jobs (see part (b)) such that all jobs with $a_i = 0$ are at the front of this permutation. Further, show that the order in which these jobs appear at the front of the permutation is not important.

4. Let I be any instance of the two-processor flow shop problem. Let $\sigma = \sigma_1\sigma_2 \cdots \sigma_n$ be a permutation defining an OFT schedule for I.
   (a) Use (5.29) to argue that there exists an OFT σ such that $\min\ \{b_i, a_j) \geq \min\ \{b_j, a_i\}$ for every i and j such that $i = \sigma_k$ and $j = \sigma_{k+1}$ (that is, i and j are adjacent).
   (b) For a σ satisfying the conditions of part (a), show that $\min\{b_i, a_j\} \geq \min\{b_j, a_i\}$ for every i and j such that $i = \sigma_k$ and $j = \sigma_r, k < r$.
   (c) Show that all schedules corresponding to σ's satisfying the conditions of part (a) have the same finish time. (*Hint*: use part (b) to transform one of two different schedules satisfying (a) into the other without increasing the finish time.)

5.11 REFERENCES AND READINGS

Two classic references on dynamic programming are:

Introduction to Dynamic Programming, by G. Nemhauser, John Wiley and Sons, 1966.

Applied Dynamic Programming by R. E. Bellman and S. E. Dreyfus, Princeton University Press, 1962.