A Dynamic Programming Approach for Sequencing Groups of Identical Jobs

HARILAOS N. PSARAFTIS
Massachusetts Institute of Technology, Cambridge, Massachusetts

(Received March 1978; accepted March 1980)

A Dynamic Programming approach for sequencing a given set of jobs in a single machine is developed, so that the total processing cost is minimized. Assume that there are N distinct groups of jobs, where the jobs within each group are identical. A very general, yet additive cost function is assumed. This function includes the overall completion time minimization problem as well as the total weighted completion time minimization problem as special cases. Priority considerations are included; no job may be shifted by more than a prespecified number of positions from its initial, First Come-First Served position in a prescribed sequence. The running time and the storage requirement of the Dynamic Programming algorithm are both polynomial functions of the maximum number of jobs per group, and exponential functions of the number of groups N. This makes our approach practical for real-world problems in which this latter number is small. More importantly, the algorithm offers savings in computational effort as compared to the classical Dynamic Programming approach to sequencing problems, savings which are solely due to taking advantage of group classifications. Specific cost functions, as well as a real-world problem for which the algorithm is particularly well-suited, are examined. The problem application is the optimal sequencing of aircraft landings at an airport. A numerical example as well as suggestions on possible extensions to the model are also presented.

THE PROBLEM addressed in this paper is that of sequencing a given set of jobs in a single machine, so that the total processing cost is minimized. We assume that there are N distinct groups of jobs, in which the jobs within each group are identical. We also assume a very general, yet additive, cost function. Specifically, the incremental cost incurred when processing a job belonging to group m followed by a job belonging to group n is defined by the general function $f(m, n, k_1, \cdots, k_N)$, where k_i is the number of jobs of group $i (i = 1, \cdots, N)$ still waiting to be processed. The problem is a "static" one, that is, no intermediate arrivals of jobs are considered.

In Section 1 the above general problem is formulated and solved using Dynamic Programming. For a problem of k jobs per group and N groups, the algorithm's running time is shown to grow as $N^2(k + 1)^N$ and its storage requirement as $N(k + 1)^N$. These are polynomial functions of k

Operations Research
Vol. 28, No. 6, November-December 1980

0030-364X/80/2806-1347 $01.25

but exponential functions of N, so one would expect the algorithm to be practical for applications for which N is small. Still, this performance represents an improvement over the performance of the classical Dynamic Programming algorithm of Held and Karp (1962) if the latter is applied to a problem of the same size. For sequencing a total of kN jobs, the latter algorithm has a running time which grows as $(kN)^2 2^{kN}$.

The given set of jobs is now considered ordered, in the form of a specified initial sequence. CPS prohibits the shifting of any particular job by more than a prespecified number of positions, upstream or downstream, from its initial, First Come-First Served position in the queue. We modify the Dynamic Programming algorithm so that CPS is incorporated with no increase in the order of computational effort.

In Section 3, specific types of cost functions $f(m, n, k_1, \cdots, k_N)$ are presented. These include as special cases the minimum overall completion time problem and the problem of minimizing the total weighted completion time. Areas where our approach can be applied are presented, the main one being the problem of sequencing aircraft landings at a single-runway airport (Aircraft Sequencing Problem).

In Section 4 a numerical example on the Aircraft Sequencing Problem is presented.

Finally, in Section 5 areas for possible extensions of the model are suggested, such as the multiple processor problem and the problem of "dynamic" arrivals of jobs.

1. PROBLEM FORMULATION AND DYNAMIC PROGRAMMING SOLUTION

It is assumed that the set of jobs to be sequenced can be classified into N groups, with k_i^0 identical jobs belonging to group $i (i = 1, \cdots, N)$. It is also assumed that the incremental cost of processing a job belonging to group n immediately after a job belonging to group m (where both m and n are between 1 and N), is given by $f(m, n, k_1, \cdots, k_N)$ where f is a prescribed function of m, n and the vector k, with k_i being the number of jobs of group $i (i = 1, \cdots, N)$ still waiting to be processed. The objective is to find a sequence of jobs which minimizes total processing cost.

This problem calls for the determination of a sequence of group indices $(L_1, L_2, \cdots, L_T)$, with $1 \leq L_j \leq N (j = 1, \cdots, T)$ and $T = \sum_{i=1}^{N} k_i^0$ so as to minimize the sum $\sum_{j=0}^{T-1} f(L_j, L_{j+1}, k_1^j, \cdots, k_N^j)$. Here k_i^j is the number of jobs of group i still waiting to be processed during the interval the jth job is being processed and T is the total number of jobs in the initial set.

The initial conditions of the problem are specified by $(k_1^0, \cdots, k_N^0)$ and L_0, the so-called 0-th job. L_0 is between 1 and N and is the job group which is being processed just prior to processing the first job of the set.

L_0 is not a decision variable to the problem, but an initial condition which will, in general, affect the cost of sequencing the first job. If no job is being processed before the first job, then $L_0 = 0$ (dummy 0-th job). In this case, we set $f(0, n, k_1, \cdots, k_N) = 0$ for all $n, k_1, \cdots, k_N$.

It is relatively straightforward to solve the above problem by Dynamic Programming. Define the optimal value function $V(L, k_1, \cdots, k_N)$ as the minimum total cost to process k_i jobs of group $i (i = 1, \cdots, N)$ still waiting to be processed, given that a job belonging to group L is just started. V obeys the following recursive relation:

$$V(L, k_1, \cdots, k_N) = \begin{cases} 0, \text{ if } k_1 = \cdots = k_N = 0 \\ \min_{x \in X}[f(L, x, k_1, \cdots, k_N) + \\ \quad V(x, k_1', \cdots, k_N')] \text{ otherwise} \end{cases} \tag{1}$$

where

$$X = [x: k_x > 0] \tag{2}$$

and, for $i = 1, \cdots, N$:

$$k_i' = \begin{cases} k_i - 1, \text{ if } i = x \\ k_i \text{ otherwise.} \end{cases} \tag{3}$$

This problem is solved as follows: Starting from $k_1 = \cdots = k_N = 0$, where $V = 0$ for all L between 1 and N, move to lexicographically higher values of the k-vector. At each N-tuple, apply (1) for all possible values of L from 1 to N. Each time (1) is applied, record x^*, the best next group to process. This information can be kept in an array, called here NEXT(L, $k_1, \cdots, k_N$). Arbitrarily set NEXT(L, $0, \cdots, 0$) = 0 for all L between 1 and N.

Upper bounds on each of the k_i's can be the given initial values k_i^0. However, the operation of the algorithm will be much more efficient in the long run if (1) is solved for all k_i between zero and some specified upper bound value $k_i^{\max}$. In this way the optimality recursion is performed essentially *only once*. After this single "production-run" is executed, we solve the problem as many times as desired and for any given set of initial conditions ($L_0, k_i^0, \cdots, k_N^0$), as long as $1 \le L_0 \le N$ and $0 \le k_i^0 \le k_i^{\max}$ for $i = 1, \cdots, N$. The additional computational effort to do this will be of the order of $T = \sum_{i=1}^N k_i^0$; the algorithm moves forward T steps, from (L_0, $k_1^0, \cdots, k_N^0$) to the terminal state ($L_T, 0, 0, \cdots, 0$), using the information already tabulated in the array NEXT. A minor modification concerns the case where $L_0 = 0$ (no initial job specified). In that case, L_1, the first job, will be the one that minimizes $V(L_1, k_1^0, \cdots, k_N^0)$.

The running time associated with the execution of the optimality recursion is of the order of $N^2 \prod_{i=1}^N (1 + k_i^{\max})$, for there are $N \prod_{i=1}^N (1 + k_i^{\max})$ possible states, and at each of them N next states are examined. The storage requirement is of the order of $N \cdot \prod_{i=1}^N (1 + k_i^{\max})$ (arrays for

V and NEXT). If $k_i^{\max} = k$ for $i = 1, \cdots, N$, these bounds become $N^2(1 + k)^N$ for the running time and $N(1 + k)^N$ for the storage requirement. Note that these functions are polynomial with respect to k, the maximum number of jobs per group, but exponential with respect to N, the number of groups. Thus, from a practical point of view the algorithm is expected to be applicable mainly to problems where N is small. One real-world problem, where N is small, is identified in Section 3.

We make two additional observations concerning the computational effort implied by the algorithm:

1. Since the optimality recursion needs to be executed only once, the long term importance of the fact that the running time is growing as $N^2(1 + k)^N$ is greatly diminished. It has already been stated that the *marginal* running time of the algorithm is a linear function of T, the total number of jobs. By contrast, storage requirement remains an important issue. If we are certain that $L_0 \neq 0$, then the array V can be totally eliminated after the end of the recursion. The array NEXT should be retained with $N(1 + k)^N$ pieces of information in storage.

2. The fact that the algorithm has taken advantage of group classification can be seen to result in savings in both running time and storage requirements. If we consider each of the kN jobs as a separate entity and apply the classical Dynamic Programming algorithm of Held and Karp, a running time of the order of $(kN)^2 2^{kN}$ will be incurred for the optimality recursion and storage space of the order of $kN2^{kN}$ will be needed.

2. CONSTRAINED POSITION SHIFTING

We now modify the formulation of the problem slightly by introducing priority considerations. Assume that the set of T jobs to be sequenced is now *ordered*, namely there is an initial *sequence* of jobs $(i_1, i_2, \cdots, i_T)$, with $1 \le i_j \le N$ for $j = 1, \cdots, T$.

Imagine, for instance, that this sequence is the sequence in which the jobs arrived for processing. Also assume that *Constrained Position Shifting* (CPS) rules must be followed. According to these, no job can be shifted by more than a prespecified number of positions, upstream or downstream, from its First Come-First Served position in the initial sequence. This prespecified number is called the *Maximum Position Shift* (MPS) and, together with the initial sequence $(i_1, \cdots, i_T)$, it is a new input to the problem. If, say, MPS = 3, the job holding the 9th position in the initial sequence is restricted to be assigned a position inside the "window" from the 6th up to and including the 12th position in the final optimal sequence. Clearly MPS = 0 corresponds to FCFS sequencing and no optimization is involved; at the other extreme, MPS $\geq T - 1$ corresponds to the earlier unconstrained case.

The practical importance of CPS is great in a "dynamic" sequencing of jobs and will be discussed in Section 5. But for the moment, continue assuming that the problem is "static," namely that no newly arriving jobs (that is, jobs other than those in the original set) are accepted for sequencing.

The CPS problem has been examined by Dear (1976) in conjunction with his early work on the "dynamic" case of the Aircraft Sequencing Problem. He used a complete enumeration procedure to examine all possible permutations in the last (MPS + 1) positions of the queue. Such an approach exhibits a computational effort growing as (MPS + 1)!, being thus limited to small values of MPS. In addition, examining only (MPS + 1) positions while "freezing" the rest of the queue leads to sub-optimal solutions.

In this section, we shall solve optimally the CPS problem by Dynamic Programming for any value of MPS. The objective function is still the minimization of total processing cost.

The state representation $(L, k_1, \cdots, k_N)$, introduced earlier, is sufficient for the CPS problem as well, for if we assume an "internal" FCFS discipline for jobs belonging to the same group, then for a given initial sequence $(i_1, \cdots, i_T)$ not only do we know *how many*, but also specifically *which* jobs per group have (or have not) been processed so far.

An effect of the MPS constraints will be that now there will be states which will be *infeasible*. In order to identify the infeasible states, we proceed as follows:

For a given state $(L, k_1, \cdots, k_N)$, we consider the number $m = T - \sum_{i=1}^{N} k_i$. This is the total number of jobs processed so far, and therefore, it is the position which the job being currently processed (from group L) holds in the *current* processing sequence.

From the initial sequence $(i_1, \cdots, i_T)$ we can uniquely identify the position the job being currently processed (L) held in the *initial* FCFS sequence. To do this, simply *scan* the above sequence from i_1 to i_T, until a total of $k_L^0 - k_L$ jobs of group L is encountered. The last of these jobs is the one being currently processed. Denote by LAST$(L, k_L^0 - k_L)$ this uniquely identified position; this is the last job from group L held in the initial FCFS sequence. Arbitrarily set LAST$(L, 0) = 0$ for all $L = 1, \cdots, N$.

It is clear that LAST$(L, k_L^0 - k_L) - m$ is the *position shift* of the job being currently processed. This difference can be positive, negative, or zero depending on whether the corresponding job has been shifted downstream (forward), upstream (backward) or not shifted at all with respect to its initial FCFS position.

It then follows that in order for $(L, k_1, \cdots, k_N)$ to be feasible, it must satisfy the following condition:

$$|\mathrm{LAST}(L, k_L{}^0 - k_L) - m| \leq \mathrm{MPS}. \tag{4}$$

It should be noted that condition (4) is only a *necessary*, and not a *sufficient* condition for feasibility. To determine whether a state which satisfies (4) is indeed feasible, it is necessary to check its next states. The following cases may occur:

1. $(L, k_1, \cdots, k_N)$ *has no next states.* This will happen if $k_1 = \cdots = k_N = 0$. If (and only if) this is the case, then (4) is also a sufficient condition for feasibility.

2. $(L, k_1, \cdots, k_N)$ *has next states, but none of them is feasible.* Then $(L, k_1, \cdots, k_N)$ is an *infeasible* state, for there can be no way to proceed from the sequence $(L, k_1, \cdots, k_N)$ without violating the CPS rules at some next state.

3. *At least one of the states next to* $(L, k_1, \cdots, k_N)$ *is feasible.* Then $(L, k_1, \cdots, k_N)$ is a feasible state because there is at least one way to finish the sequence from $(L, k_1, \cdots, k_N)$ and still satisfy the CPS rules to the end.

The above arguments reveal the *recursive* nature of feasibility in the problem. This means that the feasibility of a state not only depends on whether or not the state itself satisfies a particular set of conditions (only condition (4) here, but in general more than one condition), but also on the feasibility of its next states. The recursive nature of feasibility is particularly suited to the backward recursion scheme of the Dynamic Program described earlier. In particular, one can incorporate the reasoning in the algorithm by modifying the optimality recursion (1) as follows:

$$V(L, k_1, \cdots, k_N) = \begin{cases} +\infty, \text{ if } |\mathrm{LAST}(L, k_L{}^0 - k_L) - m| > \mathrm{MPS} \\ 0, \text{ if } |\mathrm{LAST}(L, k_L{}^0 - k_L) - m| \leq \mathrm{MPS} \\ \qquad\qquad \text{and } k_1 = \cdots = k_N = 0 \\ \min_{x\in X}[f(L, x, k_1, \cdots, k_N) \\ \qquad + V(x, k_1', \cdots, k_N')] \text{ otherwise.} \end{cases} \tag{5}$$

In (5) we have assigned an infinite cost to infeasible states. This includes the possibility that while $|\mathrm{LAST}(L, k_L{}^0 - k_L) - m| \leq \mathrm{MPS}$, all V's in the right-hand side are equal to infinity. X and k' are given by (2) and (3) as before.

In terms of computational effort, it can be seen that the recursion in the CPS case is essentially no harder to apply than the recursion in the unconstrained case. Actually, the recursion's running time will be bounded *from above* by $N^2 \cdot \prod_{i=1}^{N} (1 + k_i{}^0)$, the actual running time being lower for values of MPS less than $T - 1$. The lower MPS is, the less frequently the algorithm will have to execute the third leg of (5), because most infeasible states result from failure to pass the screening test (4). Hence, the worst-case performance in the CPS case (MPS $\geq T - 1$, all states feasible) is no poorer than the performance of the much simpler, unconstrained case problem. As before the storage requirement can be seen to grow as $N \cdot \prod_{i=1}^{N} (1 + k_i^0)$.

There is, however, an important difference between the two algorithms. In the CPS case, we can no longer solve the optimality recursion *only once* and examine, after that, whatever initial conditions we wish by simply moving forward according to the already tabulated array NEXT. Neither is it advantageous to solve the problem for upper bounds $k_i^{\max}$ higher than k_i^0. This can be understood from the fact that *it is the initial sequence itself,* together with MPS, which determines feasibility, and if either one of them is changed, one has to execute the optimality recursion again.

3. SPECIAL CASES AND POTENTIAL APPLICATIONS

Dynamic Programming algorithms have already been proposed for the solution of general, as well as specific, problems in job shop scheduling; see, for example, Held and Karp (1962), Baker and Schrage (1978), Schrage and Baker (1978), and Lawler (1964). Our algorithm can be used to solve a variety of specialized cases as well.

First, if $f(m, n, k_1, \cdots, k_N) = t(m, n)$, then the problem is a minimum overall completion time (makespan) scheduling problem (Coffman [1976]), which is exactly equivalent to a classical Traveling Salesman Problem.

If $f(m, n, k_1, \cdots, k_N) = t(m, n) \cdot \sum_{i=1}^{N} k_i p_i$, then the problem is a minimum total weighted completion time scheduling problem (Lawler [1978]). Here $(p_1, \cdots, p_N)$ is a given set of "weights," one for each group. p_i can be thought of as the per unit time cost of keeping each individual job of group i waiting to be processed. If all $p_i = 1$, then $f(m, n, k_1, \cdots, k_N) = t(m, n)\sum_{i=1}^{N} k_i$ and the problem becomes a minimum mean completion time scheduling problem.

In addition to job-shop scheduling, a very important real-world problem for which our approach is particularly well suited for application is the Aircraft Sequencing Problem (ASP). This is the problem faced by the air traffic controller who must decide on a landing sequence for a set of airplanes waiting to land, such that a certain measure of performance is optimized. The "job groups" for the ASP are the distinct *categories* into which the set of aircraft can be classified. Here N is usually a small number (of the order of 3 and at most 5: wide-body jets, medium-sized jets, etc.). Due to landing kinematics and safety regulations the minimum permissible time interval $t(m, n)$ between the landing of an aircraft of category m, followed by the landing of an aircraft of category n, is a quantity which is not constant but depends on m, n, their relative order, landing velocities and the length of common final approach (Blumstein [1959]). FCFS landing disciplines at airports are among the factors contributing to delays during peak traffic periods. These delays can, at least theoretically, be reduced by a suitable optimization in the landing sequence. In this respect, the D.P. algorithm can be used to examine the minimization of the Last Landing Time (LLT), that is, land the last airplane as soon as possible. The algorithm can also be used to minimize the Total Passenger Delay (TPD), that is, minimize the sum of the "waiting-to-land" times for all passengers in the system (or equivalently, the average per passenger delay). In the LLT case, $f(m, n, k_1, \cdots, k_N) = t(m, n)$ and in the TPD case $f(m, n, k_1, \cdots, k_N) = t(m, n) \cdot \sum_{i=1}^{N} k_i p_i$ where k_i is the number of airplanes of category i which are still waiting to land and p_i is the (average) number of passengers (or the number of seats) of an aircraft belonging to category $i (i = 1, \cdots, N)$.

What makes our D.P. approach particularly attractive for the ASP is its CPS feature. CPS is actually a concept which was first developed in conjunction with the ASP. The main role of CPS in the ASP is to prohibit sequences which are intolerably biased against some aircraft categories, particularly in a "dynamic" environment (in which the landing of some airplanes may be continually postponed due to arrival of aircraft categories which receive higher priority).

The need for implementing an optimization scheme in the sequencing of aircraft operations at an airport has been recently recognized by the Federal Aviation Administration as one of the most important directions for increasing the current operational capacity of airports (Talley [1978]). A comprehensive survey of theoretical and implementation problems regarding the "dynamic" scheduling of aircraft arrivals is given in Dear (1976), while more details on the D.P. approach to the ASP, including the two-runway configuration, are presented in Psaraftis (1978). Section 4 presents a numerical example on the problem.

Another variation of our D.P. approach can be used for the solution of the single-vehicle, many-to-many immediate request dial-a-ride problem, for both "static" and "dynamic" cases (Psaraftis [1980]). "Dial-a-ride" is a general name given to demand-responsive (or flexibly routed) transportation systems, several versions of which have been in operation in the United States and abroad (Rochester, N.Y.; Ann Arbor, Mich.; Tokyo, Japan, etc.). The particular problem examined in Psaraftis (1980) involves the dispatching of a vehicle to carry customers from distinct origins to distinct destinations. A generalized objective has been examined, consisting of a weighted combination of the total length of the route and of the total "degree of dissatisfaction" caused to the customers until they are delivered. It is assumed that, for every customer, the above dissatisfaction is a linear combination of the time the customer waits for the vehicle and the time (s)he spends in the vehicle until delivery. CPS considerations have been incorporated into this problem as well as vehicle capacity constraints. CPS is particularly useful in the "dynamic" case for eliminating the undesirable possibility of indefinite deferment of a customer's request, deferment which may be caused by the possibly unfavorable geographical location of that customer.

4. A NUMERICAL EXAMPLE

We now present a straightforward application of the algorithm of Section 2, for a particular instance of the Aircraft Sequencing Problem introduced in Section 3. In our example, $N = 3$: category 1 consists of wide-body jets (B747), category 2 of conventional large jets (B707) and category 3 of medium-sized jets (DC-9). The time separation matrix was calculated (Psaraftis [1978]) to be:

$$[t(m, n)] = \begin{bmatrix} 96 & 181 & 228 \\ 72 & 80 & 117 \\ 72 & 80 & 90 \end{bmatrix} \text{ (in seconds)}$$

The (average) number of passengers per category is given by $(p_1, p_2, p_3) = (300, 150, 100)$.

We assume that $T = 15$ airplanes are waiting to land. We also assume the following "pseudorandom" initial FCFS sequence: $(i_1, i_2, \cdots, i_{15}) \equiv (1, 1, 3, 2, 2, 3, 2, 1, 2, 1, 3, 3, 2, 1, 2)$. This means that our initial conditions are $(k_1^0, k_2^0, k_3^0) = (5, 6, 4)$. We arbitrarily assume that sequencing begins at the instant after an aircraft of category 2 has landed. This is our 0-th airplane, $L_0 = 2$.

We then apply our CPS algorithm for MPS = 5 and 14 and for both LLT and TPD minimization as these were defined in Section 3. The results are shown in Table I. MPS = 14 = $T - 1$ corresponds to the unconstrained case. Note how the optimal sequences and the value of the objective function in Table I change with changes in MPS. Table I also shows the position shifts of each airplane, as well as the percent improvements of LLT and of TPD by comparison to their FCFS values.

A global picture on how LLT and TPD perform for this particular example, as functions of MPS, is shown in Figures 1 and 2. Figure 1 shows the percent improvement in LLT and Figure 2 the percent improvement in TPD over the FCFS sequence. Solid lines in both figures indicate that the measure of performance in question is the objective to be minimized (LLT in Fig. 1 and TPD in Fig. 2). Dotted lines show the behavior of the alternative measure of performance when the former is optimized (TPD

TABLE I

A Numerical Example on the Aircraft Sequencing Problem

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	LLT (seconds)	TPD (pass. seconds)
	LANDING ORDER	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
	INITIAL SEQUENCE	2	1	1	3	2	2	3	2	1	2	1	3	3	2	1	2	1729	2383800
MPS = 5																			
Min LLT	OPTIMAL SEQUENCE	2	1	1	1	2	2	2	2	3	3	3	3	2	2	1	1	1400	2033800
	POSITION SHIFTS	0	0	0	5	0	0	1	2	−5	−3	1	1	1	2	−4	−1		
	% IMPROVEMENT																	19%	14%
Min TPD	OPTIMAL SEQUENCE	2	2	1	1	1	1	2	2	3	3	2	2	1	2	3	3	1528	1883250
	POSITION SHIFTS	0	3	−1	−1	4	5	−1	0	−5	−3	−1	2	2	2	−3	−3		
	% IMPROVEMENT																	11%	20%
MPS = 14																			
Min LLT	OPTIMAL SEQUENCE	2	2	2	2	2	2	2	3	3	3	3	1	1	1	1	1	1323	2241300
	POSITION SHIFTS	0	3	3	4	5	8	9	−4	−2	2	2	−10	−10	−5	−4	−1		
	% IMPROVEMENT																	23%	5%
Min TPD	OPTIMAL SEQUENCE	2	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	1424	1664900
	POSITION SHIFTS	0	0	0	5	6	9	−2	−2	−1	0	3	4	−9	−7	−3	−3		
	% IMPROVEMENT																	17%	30%

in Fig. 1 and LLT in Fig. 2). Not unexpectedly, the solid lines are nondecreasing, while this is not generally the case for the dotted lines. Also, as expected, the solid lines are nowhere below the dotted lines, because the improvement of the measure of performance (which is the objective of the problem [solid line]) is, by definition, the maximum improvement achievable.

5. SUGGESTIONS FOR POSSIBLE EXTENSIONS

The following directions seem appropriate for possible extensions to our model:

First, the formulation can be extended to include a second processor.

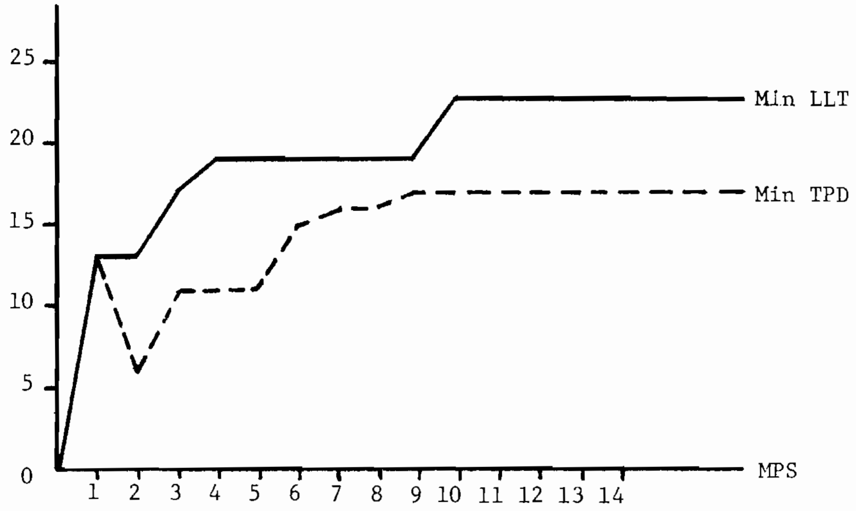

Figure 1. Percentage improvement in LLT with respect to the FCFS discipline.

This can be done by replacing L in the state vector by (L_1, L_2), where L_i is the group currently being processed by machine i. The recursive relation then becomes

$$V(L_1, L_2, k_1, \cdots, k_N) = \begin{cases} 0, \text{ if } k_1 = \cdots = k_N = 0 \\ \min_{x \in X,\ y=1,\ 2}[f(L_1, L_2, x, k_1, \cdots, k_N) \\ \quad + V(L_1', L_2', k_1', \cdots, k_N')] \end{cases}$$

where X and k_i' are again given by (2) and (3) and

$$L_i' = \begin{cases} x \text{ if } y = i \\ L_i \text{ otherwise.} \end{cases}$$

Storage requirement for this formulation grows as $N^2(1 + k)^N$ and running time as $N^3(1 + k)^N$.

An alternative approach was taken in Psaraftis (1978), for tackling the problem of sequencing aircraft landings in two identical and independent runways where no priority constraints exist. This consists essentially of a postprocessing of the optimal information created by a single pass of the optimality recursion presented in Section 1. LLT minimization is now a *minimax* problem. Although an enumeration scheme is used to identify the optimal aircraft partition, running time and storage requirements are of the same order of magnitude with those of the single-runway algorithm. Significant difficulties arise if one attempts to apply this approach for more than two runways or include priority rules.

A second extension would be the creation of a "dynamic" model where

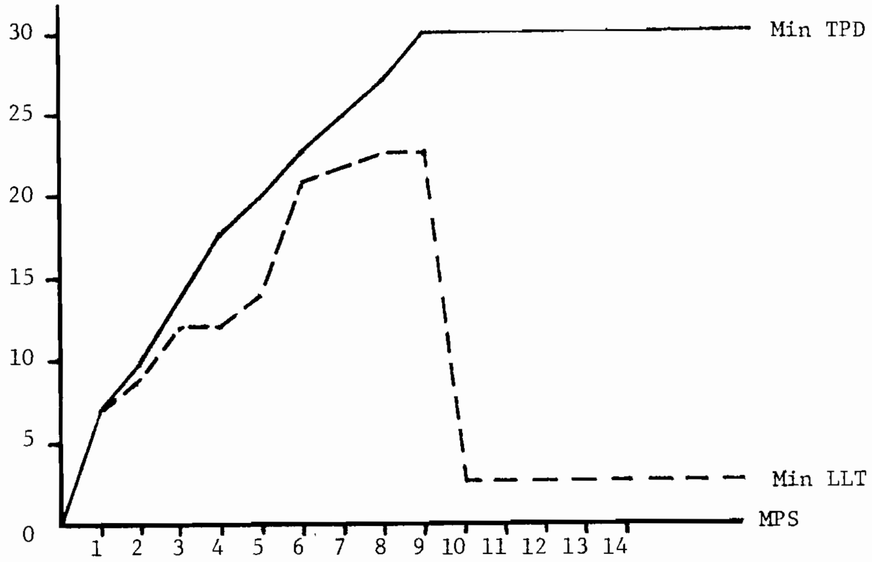

Figure 2. Percentage improvement in TPD with respect to the FCFS discipline.

new jobs enter the processing system continually in time. Such an extension is relatively straightforward and, in fact, has been carried out for the dial-a-ride problem (Psaraftis [1980]). A "dynamic" model will be an open-ended sequence of updates, each being performed upon the arrival of a new job. We would optimize over the existing set of jobs, producing one tentatively optimal sequence per update. This sequence will be subject to revision upon the arrival of a new job. CPS can be particularly important in the "dynamic" case by eliminating the possibility of some jobs being continually held last in the queue because of their "unfavorable" characteristics.

Finally, in the minimum weighted completion time sequencing problem

(Section 3), we have assumed that all jobs within a group i have the same weight p_i. This may not be the case in several problems (for example, actual passenger loads will vary for the ASP among aircraft of the same category). If such internal variations exist, it is easy to show that it generally pays to sequence all jobs of each group by nonincreasing order of weights. However, this issue is complicated if one starts examining interactions between groups, and if priority considerations are added. A detailed investigation of such issues is presented in Psaraftis (1978).

ACKNOWLEDGMENTS

I wish to thank Amedeo Odoni and Christos Papadimitriou, both of M.I.T., for their helpful suggestions. I am also indebted to one of the referees for his careful review and comprehensive comments, and in particular for his suggestion on the formulation of the two-processor problem.

REFERENCES

BAKER, K. R., AND L. E. SCHRAGE. 1978. Finding a Sequence by Dynamic Programming: An Extension to Precedence-Related Tasks. *Opns. Res.* **26,** 111-120.

BLUMSTEIN, A. 1959. The Landing Capacity of a Runway. *Opns. Res.* **7,** 752–763.

COFFMAN, E. G., JR. (ed.) 1976. *Computer and Job Shop Scheduling Theory.* John Wiley & Sons, New York.

DEAR, R. G. 1976. The Dynamic Scheduling of Aircraft in the Near Terminal Area. FTL Report R76-9, Flight Transportation Laboratory, M.I.T., Cambridge, Mass., August.

HELD, M., AND R. M. KARP. 1962. A Dynamic Programming Approach to Sequencing Problems. *SIAM,* **19,** 196–210.

LAWLER, E. L. 1964. Sequencing Problems with Deferral Costs. *Mgmt. Sci.* **11,** 280-288.

LAWLER, E. L. 1978. Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints. *Ann. Discrete Math.* **11,** 75–90.

PSARAFTIS, H. N. 1978. A Dynamic Programming Approach to the Aircraft Sequencing Problem. FTL Report R78-4, Flight Transportation Laboratory, M.I.T., Cambridge, Mass., October.

PSARAFTIS, H. N. 1980. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Trans. Sci.* **14,** 130-154.

SCHRAGE, L. E., AND K. R. BAKER. 1978. Dynamic Programming Solution of Sequencing Problems with Precedence Constraints. *Opns. Res.* **26,** 444–449.

TALLEY, J. R. 1978. Basic Metering and Spacing for ARTS III. Systems Research and Development Service, Progress Report, Federal Aviation Administration, August.