MANAGEMENT SCIENCE
Vol. 24, No. 10, June 1978
Printed in U.S.A.

ALGORISTICS FOR SINGLE-MACHINE SEQUENCING WITH PRECEDENCE CONSTRAINTS*

THOMAS E. MORTON† AND BALA GANGA DHARAN†

Although Horn, Sidney and Lawler have considered the problem of minimizing weighted mean flow time for n jobs with precedence constraints on one machine, practical exact algorithms for the general case remain elusive. Two relatively sophisticated heuristics are presented which are computationally attractive. Each is optimal on large subclasses of problems; computational study demonstrates each to be extremely close to optimal in general. Such heuristics are dubbed "algoristics". The first-come first-served heuristic serves as a benchmark. Finally, a simple "myopic" heuristic produces about 80% of the savings of the algoristics; it possesses the advantages of a dispatching rule. A standard branch and bound procedure has also been computed for a large subset of problems for comparison purposes. Planning horizon results are also derived.

1. Introduction

Consider the following problem. A set of jobs $1, 2, \ldots, i, \ldots, n$ with processing times t_i and weights w_i are to be processed sequentially without pre-emption on a single machine. If job j is sequenced in position i, use the notation $i = [j]$. A precedence constraint $j \prec k$ is read "j must precede k" or alternatively "it must be true that $[j] < [k]$". Then the problem is to choose the sequence to minimize weighted mean flow time subject to a number of such precedence constraints:

$$\min F_w = \sum_{[j]=1}^{n} w_{[j]} \left\{ \sum_{[k]=1}^{[j]} t_{[k]} \right\}$$
$$\text{s.t. precedence constraints.}$$

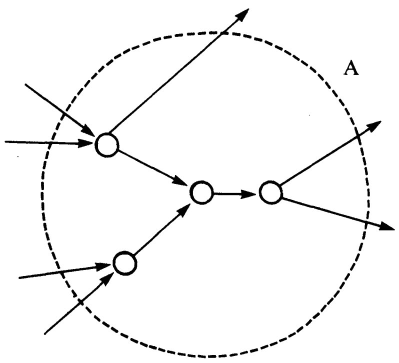
(In this paper, when the "cost" of a particular schedule is referred to, $\bar{F}_W$ is intended.) If $i \prec j$, i is said to be a *predecessor* of j, j to be a successor of i. The relation is *transitive*. If for no job k is it true that in addition $i \prec k \prec j$ then i is a *direct predecessor* of j, j is a *direct successor* of i, written $i \lll j$ or $j \ggg i$. A set of n jobs for which each job has at most one direct predecessor is referred to as a *branching tree*. A *graph* shows the jobs as nodes and the relation "direct successor" as connecting arrows, which by convention proceed from left to right. The graph of a branching tree actually resembles one or more "trees" branching out from left to right. A set of jobs for which each job has at most one direct successor is referred to as an *assembly tree*; its graph is one or more "trees" branching out from right to left.

Define a set of *generators* as any subset of n jobs. The *initial set* of a given set of generators is the set plus all predecessors of any generator. A *final set* is a set of generators and all successors. A *simple initial set* or a *simple final set* has a single generator. The *processing time per unit value* or *normalized processing time* is defined as $z_j = t_j / w_j$.

A subset of jobs which is known to be scheduled as a block in the unknown larger schedule, and for which the interval sequence within the block has been specified, can be aggregated into a *single composite job*, with appropriate composite processing time,

* Accepted by David G. Dannenbring; received July 18, 1977. This paper has been with the authors 2 months, for 2 revisions.
† Carnegie-Mellon University.

weight, normalized processing time, and precedences. All are easily specified:



EXAMPLE. Job Set to be Aggregated.

$$t_A = \sum_A t_i, \quad w_A = \sum_A w_i, \quad z_A = t_A / w_A.$$

Predecessors of the composite job are simply the set union of predecessors of the original set, less members of the composition, similarly for successors. Determination of the direct predecessors and successors requires an algorithm. (See the Appendix.) In the sequel we shall primarily be interested in aggregating simple initial sets, or in aggregating pairs of jobs which are direct predecessor and successor of each other.

With the machinery these definitions provide, historical results for the problem are summarized.

THEOREM (SMITH [8]). *With no precedence constraints, an optimal solution is obtained for any sequence for which*

$$z_{[1]} \leq z_{[2]} \leq \cdots \leq z_{[n]}.$$

THEOREM (HORN [3]). *For branching trees the single component initial set with minimal z_A may be scheduled first.*

THEOREM (SIDNEY [7]). *For assembly trees the single component initial set with minimal z_A may be scheduled first.*

By symmetry, the corresponding result for assembly trees is to find the equivalent final set with maximal z_A to be scheduled last. Baker [1, pp. 88 to 92] notes that "The problem of calculating . . . efficiently is still a significant one, and the computational aspects . . . of (Horn's) solution procedure have not been thoroughly investigated."

Recently Lawler [4] has given an algorithm which is $O(n \log n)$ for the more general case in which precedence constraints are "series parallel".

THEOREM (SIDNEY [7]). *For the general case, the initial set S^* with minimal z_A may be scheduled first.*

In discussing Sidney's result Baker essentially notes that the problems in implementing the Theorem are that there are up to 2^n initial sets to check, and that no efficient procedure exists for scheduling the jobs within the optimal initial set.

Lawler [4] has recently given an algorithm for finding S^* which is polynomial in the number of jobs n. However, the *power* of the polynomial may be arbitrarily large, depending on the range of the z_j.

In §2 several results are developed on the optimal aggregation of pairs of jobs, which lead to an algoristic which can be shown to be optimal for branching and assembly trees. Several variants of the procedure are developed and advantages and disadvantages are discussed.

In §3 an algoristic is produced from the Sidney result by the simple expedient of restricting the search for minimal z_A to be over *simple* initial sets. The resulting procedure is optimal for assembly trees. The algoristic is easier to program and faster in computation than the algoristic just discussed. A mirror image algoristic, which does not in general yield the same solution, is easily produced by considering only simple final sets. Since both procedures can share the same precedence matrix and most of the same logic, it seems reasonable to run both in practice and pick the better result.

In addition, there are two simple heuristics which merit consideration. A *dispatch rule* is a sequencing procedure which is simple enough that a dynamically arriving job may simply be inserted appropriately into a previously optimal schedule. In practice a manager may be willing to sacrifice a fair amount of theoretical efficiency for the ease and robustness of a dispatch rule. In the present problem, there are two dispatch rules of obvious interest. The first is *first-come first served* (FCFS): process next the *feasible* job with lowest index. This is the simplest possible procedure; its performance can serve as a benchmark. The second dispatch rule is the *myopic* rule: choose the next feasible job with smallest z_i.

In §4, computational results are presented for the four scheduling procedures mentioned: (i) FCFS, (ii) myopic, (iii) tree-optimal, (iv) Sidney-type, for 375 test problems. Problems ranged in size from 10 to 50 jobs, and in density of the precedence matrix from 5% to 30%. Other parameters were generated randomly. For 165 of the problems a branch and bound procedure was also utilized to find optimal answers for comparative purposes. However, the latter procedure was terminated after 1000 branches.

To summarize the results, branch and bound performed better than the best algoristic in only 3 of the 165 problems, with a maximum improvement of 7/10 of 1%. The difference in performance between the two algoristics was about 1/10 of 1% in favor of the tree-optimal procedure. The simple heuristics cost 31% and 6% respectively more than the best algoristic on average. The performance of the myopic procedure deteriorated somewhat with problem size.

Finally in §5, appropriate regeneration sets are identified so that the planning horizon theory due to Lundin and Morton [3], and Morton [4] may be directly applied.

2. The Tree-Optimal Algoristic

First some further definitions are needed. Let $N = \{\text{all jobs}\}$. Define:

j^* by $z_{j^*} = \min_{j \in N}\{z_j\}$,

j^{**} by $z_{j^{**}} = \max_{j \in N}\{z_j\}$,

k^* by $z_{k^*} = \max_{k \ll j^*}\{z_k\}$,

k^{**} by $z_{k^{**}} = \min_{k \gg j^{**}}\{z_k\}$,

$n^* = |\{k \mid k \ll j^*\}|$,

$n^{**} = |\{k \mid k \gg j^{**}\}|$.

That is to say, j^* and j^{**} are the jobs minimizing and maximizing z_j over all jobs, k^* and k^{**} are the direct predecessor and direct successor maximizing or minimizing z_k respectively, and n^* and n^{**} are the number of direct predecessors of j^* and direct successors of j^{**} respectively. (In the next event of ties, all results below apply to each possible index (j^*, j^{**}, k^*, k^{**}).)

LEMMA 1. *For at least one optimal sequence*
(a) $[j^*] - 1 = [\tilde{k}]$ *for some* $\tilde{k} \ll j^*$,
(b) $[j^{**}] + 1 = [\tilde{\tilde{k}}]$ *for some* $\tilde{\tilde{k}} \gg j^{**}$.

PROOF. Consider a supposedly optimal solution which violates the first half of the lemma. Now if job j^* is moved to the earliest feasible point in the sequence, mean flow time cannot be increased. Similarly for j^{**}.

LEMMA 2. *If* $i \ll j$ *and an optimal sequence is known to exist with* $[i] = [j] - 1$ *then* i *and* j *can be aggregated.*

Now if a rule could be found for choosing $\tilde{k}$ or $\tilde{\tilde{k}}$ optimally, an exact algorithm could be constructed. Failing this, the following heuristic choice suggests itself. k^* is a "good" choice for k, since it would be intuitive to schedule the largest z_k as late as possible. Similarly k^{**} is a "good" choice for $\tilde{\tilde{k}}$.

Tree-Optimal Algoristic

1. Update j^*, j^{**}, n^*, n^{**}, k^*, k^{**}; if there remains a single job, go to 9.
2. If j^* has no predecessors, schedule it first; remove it from the current set of jobs; go to 1.
3. If j^{**} has no successors, schedule it last; remove it from the current set of jobs, go to 1.
4. If j^* has a single direct predecessor, go to 5, otherwise to 6.
5. Combine k^* and j^*; go to 1.
6. If j^{**} has direct successor go to 7, otherwise to 8.
7. Combine j^{**} and k^{**}; go to 1.
8. If $n^* \leqslant n^{**}$ go to 5, otherwise to 7.
9. Reconstruct the job sequence by disaggregating jobs in reverse order.

PROPOSITION 1. *The Tree-Optimal Algoristic provides an optimal solution for branching tree or assembly tree structures; in fact, an optimal solution is provided if step 8 is never executed.*

3. The Sidney-Type Algoristic

As mentioned in the introduction, the main problems in implementing Sidney's decomposition principle are:
1. Identifying the optimal initial set S^*;
2. Scheduling the members of S^*.

In general neither task is computationally appealing, even given Lawler's [4] result for the first problem. Furthermore, the second problem can in general be shown equivalent to integer programming problems which are *NP*-complete, and for which it is generally believed that polynomial bound algorithms do not exist.

A heuristic for approximately solving problem 1 is to reduce the search for minimal z_A to *simple* initial sets, that is, initial sets with one generator. Sidney suggests as an improvement that this generator be fixed, and then the second generator *added* giving the greatest improvement, and then the third, etc. (Stop at any time no improvement is possible.)

Turning to the problem of scheduling the members of S^*, it is tempting to treat these members as a sub-problem, find the optimal initial set for that, and eventually recursively determine a single job to be scheduled first. Unfortunately the optimal initial set of S^* is S^* itself, so that proper subsets are not obtained. Note also, however, that one of the generators of S^* must be scheduled last. If k is scheduled last, then $S^* - \{k\}$ will be guaranteed to have a strictly smaller optimal initial set.

Another advantage in the first problem of restricting attention to simple initial sets, is that there is a single generator, and thus a unique choice of the job to be scheduled last. For more complicated initial sets, one obvious heuristic would be to delete each possible generator as the last element in turn. For each subproblem thus created find ths "best" initial set. Of these possible inital sets, choose that with smallest z_A and proceed.

An especially simple heuristic results when attention is restricted to simple initial sets.

Sidney-Type Algoristic

1. Set the current job set at all unscheduled jobs.
2. If the set is empty, terminate.
3. Find the simple initial set with minimal z_A.
4. If the resulting set has more than one element, go to 5; otherwise schedule the job and go to 1.
5. Set the current job set to the current initial set less its generator; go to 3.

PROPOSITION 2. *The Sidney-Type Algoristic is optimal for assembly tree structures.*

PROOF. Follows immediately from Horn's [3] result.

While the tree-optimal heuristic is known to be optimal for a larger class of problems than the Sidney-type heuristic, the latter algoristic is much simpler to program and use, and appears computationally to be almost as good as the tree-optimal approach. In addition the Sidney-type algoristic is more easily extended to planning horizon procedures (see §5).

4. Computational Results

The four scheduling procedures:

(i) first-come first-served heuristic,
(ii) myopic heuristic,
(iii) tree-optimal algoristic,
(iv) Sidney-type algoristic

have been tested in a large scale computational study. The study randomized job processing times t_i, weights w_i, and the precedence matrix. Times t_i were chosen uniformly on [0, 10], w_i uniformly on [0.01, 1.0]. Given a desired precedence matrix P generate precedences by

$$i < j \Rightarrow \text{Prob}\{i \prec j\} = \rho$$

Then add additional necessary precedences to make the matrix transitive. Problems with 10, 15, 20, 30 and 50 jobs were tested; precedence densities were tested; precedence densities were taken as 0.05, 0.10 or 0.30. 25 replications were run of each of the resulting 15 basic cases, leading to a total of 375 test problems.

In addition to comparing these four heuristics with one other, we desired where possible to compare them with optimal solutions. For this reason a branch and bound routine has been employed. The best algoristic result is used as the initial feasible solution. The problem branches on the next feasible job to schedule, in order of increasing z_j. The bound is the weighted flow if precedence constraints are ignored on remaining jobs. Due to computational considerations, the routine has been limited to 1000 branches. (For 30 job problems this still represents minutes of CPU time on the IBM 360-65 computer.) This procedure would be expected to be efficient for densities either very close to zero, or to one.

The purpose of this branch and bound investigation is *not* to find the most efficient branch and bound, but rather to measure roughly how easy it is to improve the algoristics via heavy computation. Due to its high computation cost, branch and bound "verification" was only carried out on a subset of the full 375 test problems. For $n = 10$ all 75 problems were computed; for $n = 15$, 20, 30, only 30 of the 75 problems were chosen randomly and computed; for $n = 50$ none of the 75 problems were computed. As to be expected, the adequacy of branch and bound decreases sharply with problem size. For $n = 10$ only 7% of the problems terminated before optimality, for $n = 15$ 53%, and for $n = 20$ and 30 100%. The percentage improvement offered by branch and bound over the best algoristic is as follows: 162 cases, none; 1 case, 0.2%; 1 case 0.4%; 1 case, 0.7%, average 0.01%. Thus by this measure the algoristics performed extremely well.

Table 1 summarizes the performance of the various heuristics for the 375 test problems. The cost of the first-come first-serve procedure is normalized to 100.0 so that the others are directly expressed as a percentage of this cost. Each line of the table represents an average of 25 test problems. Note that the myopic, the Sidney, and the three-optimal procedures all perform much better than first-come first-served, averaging about 20 to 25% improvement overall. Sidney and tree-optimal in turn consistently outperform the myopic procedure by a wide margin, but are virtually indistinguishable in performance from each other. Overall, the myopic procedure costs 81.2, the Sidney procedure 76.6, the tree-optimal procedure 76.5. The marginal superiority of the tree-optimal algoristic over the Sidney-type algoristic becomes more noticeable for larger numbers of jobs.

Since the myopic procedure is easy to use, and represents a *dispatching procedure*, the results given by Table 1 are probably not enough to rule out its use in many practical situations. Table 2 investigates this question further. It can be seen that the myopic procedure obtains about 80% of the potential savings of the more sophisticated procedures at low computational cost. Note however that the effectiveness of the myopic procedure falls off for larger numbers of jobs.

TABLE 1

Average Cost of Three Heuristics as a Percentage of the Cost of First-Come First-Served, for 375 Test Problems

n	ρ	Myopic	Sidney	Tree-Optimal	Av. % Myopic Excess Cost
10	5	66.9	64.5	64.5	3.7
	10	72.3	69.6	69.6	3.9
	30	87.8	85.4	85.3	2.9
15	5	66.3	64.2	64.2	3.3
	10	75.5	70.5	70.5	7.1
	30	91.5	88.8	88.6	3.3
20	5	69.3	65.4	65.2	6.2
	10	77.3	70.4	70.3	10.0
	30	91.0	88.4	88.4	3.0
30	5	72.3	66.9	66.9	8.1
	10	88.9	75.2	75.0	9.2
	30	93.3	92.3	91.9	1.6
50	5	79.3	69.7	69.6	13.9
	10	90.2	82.3	82.2	9.8
	30	96.0	95.1	94.7	1.4
Average		81.2	76.6	76.5	6.1

TABLE 2
Myopic Savings (over FCFS) as a Percentage of Available Savings from the Algoristics

n	ρ	Myopic Savings Percentage of Available Savings
10	5	93
	10	91
	30	83
15	5	94
	10	83
	30	75
20	5	88
	10	76
	30	78
30	5	84
	10	44
	30	83
50	5	68
	10	55
	30	75
Average		78

Another possible criterion for choosing a heuristic would be that it have a high *probability* of achieving the best result in an individual problem. By this criterion, the tree-optimal algoristic scores 86%, the Sidney-type algoristic 68% and the myopic heuristic only 8%. (See Table 3). Note that there are many cases where the algoristics agree for small numbers of jobs, but few for 50 jobs. It can be deduced that the

TABLE 3
Percentage of the Time Each Heuristic Ties the Best Heuristic for a Problem

n	ρ	Myopic	Sidney	Tree-Optimal
10	5	36	100	100
	10	24	92	100
	30	28	96	96
15	5	0	100	100
	10	8	92	96
	30	8	68	84
20	5	0	80	100
	10	0	76	100
	30	0	52	76
30	5	0	72	84
	10	0	44	84
	30	12	28	76
50	5	0	52	60
	10	0	40	60
	30	0	32	68
Average		8	68	86

probability of an algoristic obtaining the *optimal* answer is lower for 50 jobs, since their answers differ from each other for this case.

In summary, both algoristics seem to perform near perfectly, although performance probably begins to drop off somewhat at 50 jobs and above. The tree-optimal algoristic is marginally superior in performance. Countering this, the Sidney-type algoristic is much easier to program and to document, important practical issues. The hand useable myopic heuristic, which possesses the advantages of a dispatching rule, picks up 80% of the savings of the more sophisticated procedures, although this percentage falls off with problem size. The myopic procedure is thus still competitive for situations in which implementation, floor-acceptance and documentation are serious issues.

For both algoristics computation time is short, and is not an issue. Programming complexity is likely to be the implementation issue. Both the Sidney and myopic procedures are easily adapted to dynamic situations where jobs arrive over time. In addition the Sidney-type procedure is adaptable to produce approximate planning horizons as shown in the following section. Since the tree algorithm does not concentrate on the "initial part" of the network, it cannot be used for planning horizons.

A final point to note is that the mirror image Sidney-type procedure (largest simple final sets last) may easily be programmed to share the same logic. It is *not* an equivalent procedure. Computation of both and choosing the better result would produce a procedure optimal for both assembly and branching trees which might compare very favorable with the tree-optimal approach.

5. Planning Horizons

In this section some planning horizon results are derived. Define the "precedence level" of a job j by

$$\mathrm{PL}(j) = \begin{cases} 1 & \text{if } j \text{ has no predecessors,} \\ 1 + \max\limits_{i \ll j} \mathrm{PL}(i) & \text{otherwise.} \end{cases}$$

Define subproblem k as the set of all jobs with precedence level less than or equal to k.

In practice, information about higher "precedence level" jobs will often be less precise. A *rolling horizon* procedure is therefore commonly used, where the problem will be re-solved with updated forecast information after the first job is completed. Hence, the point of a planning horizon procedure in this case is *not* to find a more efficient way to solve the static n job problem, but rather to find the *first job* to be scheduled quickly, and with as little information about "later" jobs as possible. The planning horizon problem in the spirit of Lundin and Morton [3] and Morton [4] would be the following:

Suppose subproblems are solved in order of increasing k, in a "forward algorithm". When can information through level k be sufficient to specify exactly the *first* job to be scheduled, irrespective of information in later levels? (An *exact* planning horizon) Failing this, when can the choice of first jobs to be scheduled be narrowed to a small subset of the available jobs? (Bound on the optimal policy)

The answer is easily given. Let L be any subset of jobs for which $j \in L$ implies $\mathrm{PL}(j) \leqslant k$, and for which there is at least one member of L say l for which $\mathrm{PL}(l) = k$. Let $\mathfrak{L} = \{L\}$ that is, $\mathfrak{L}$ consists of all such subsets. Let $S^*(L)$ represent the optimal initial set for any subproblem L.

PROPOSITION 3. *$\mathfrak{S}^*(\mathfrak{L}) = \{S^*(L) \mid L \in \mathfrak{L}\}$ is a regeneration set. That is, for every subproblem of level $t \geqslant k$, it is optimal to schedule at least one of the initial sets in $\mathfrak{S}^*(\mathfrak{L})$ first.*

PROOF. Any optimal order has a first occurence of an element at level k with a successor. Let j^* be this element. Then j^* and the jobs preceding it constitute the desired initial set.

It is worthwhile to point out that a planning horizon result which is much more practical to implement arises if the criterion is changed to schedule that job which would be chosen by the Sidney algoristic. In this case let

$J \equiv \{j \mid \mathrm{PL}(j) = k$, or $\mathrm{PL}(j) < k$ and j has no successor$\}$,
$S_s^*(j), j \in J, \equiv$ optimal simple initial set generated by j.

PROPOSITION 4. *For the Sidney algoristic criterion, $\mathfrak{S}_s^*(J) = \{S_s^*(j) \mid j \in J\}$ is a regeneration set. That is, for every subproblem of level $t \geqslant k$ it is optimal to schedule at least one of the simple initial sets in $\mathfrak{S}_s^*(J)$ first.*

PROOF. Similar.

The advantage of Proposition 4 is that $|\mathfrak{S} * (\mathfrak{L})|$ may be exponential in the number of jobs of level less than or equal k, while $|\mathfrak{S}_s^*(J)|$ is not.

We remark that both Propositions 3 and 4 may be strengthened at the expense of complicating the definitions. For Proposition 3, L can be restricted to those subsets possessing an l for which $\mathrm{PL}(l) = K$ and l has a successor. To strengthen Proposition 4, let

$M \equiv \{j \mid \mathrm{PL}(j) = k$, and j has a successor$\}$,
$z^*(j) \equiv z_A$ of $S_s^*(j)$,
$z^u = \max_{j \in M}(z^*(j))$,
$J = M \cup \{j \mid \mathrm{PL}(j) \leqslant k, j$ has no successor, and $z^*(j) \leqslant z^u\}$.

Appendix

ALGORITHM. Direct Predecessors, Composite Jobs

$D = (d_{ij})$ = indicator matrix, *direct predecessors*,

$$d_{ij} = \begin{cases} 1, & i \lll j, \\ 0, & \text{otherwise}, \end{cases}$$

$D_{\cdot j} = j$th column of D = indicator vector for direct predecessors of j,

$E = (e_{ij})$ = indicator matrix, *predecessors*.

For any set S, $I(s)$ = indicator vector of S, with 1's corresponding to elements of S, 0 otherwise.

Logical operators $\vee$(or), $\wedge$(and), $\sim$(negation) as usual.

Then the indicator vector for direct predecessor set of A is given by:

$$W \wedge \Big(\sim \bigvee_{j \in W} E_{\cdot j}\Big) \tag{A1}$$

where

$$W = \Big(\bigvee_{j \in A} D_{\cdot j}\Big) \wedge (\sim I(A)). \tag{A2}$$

REASONING. Direct predecessor set of A must be a subset of all direct predecessors of elements of A, and contains no element of A itself; the set so described is W in equation (A2).

Which elements of W are *not* direct predecessors of A? Precisely those which are predecessors of other elements of W. Thus, expression (A1).[1]

[1] The authors would like to acknowledge Suresh Chand of Carnegie-Mellon University who coded and carried out the branch and bound work in this paper, and the very helpful referees, especially the one who supplied the current Appendix.

This paper was prepared under National Science Foundation Grant ENG75-16454.

References

1. Baker, K. R., *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.
2. Conway, R. W., Maxwell, W. L. and Miller, L. W., *Theory of Scheduling*, Addison-Wesley, Reading, Mass., 1967.
3. Horn, W. A., "Single Machine Job Sequencing with Treelike Precedence Ordering and Linear Delay Penalties," *SIAM J. Appl. Math.*, Vol. 23 (1972), pp. 189–202.
4. Lawler, E. L., "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints," Computer Science Division, Univ. of Calif., to appear.
5. Lundin, R. A. and Morton, T. E., "Planning Horizons for the Dynamic Lot Size Model: Zabel vs. Protective Procedures and Computational Results," Operations Res., Vol. 23 (1975), pp. 711–734.
6. Morton, T. E., "Infinite Horizon Dynamic Programming Models—A Planning Horizon Formulation," Carnegie-Mellon W.P. 5-75-76 (to appear in *Operations Res.*).
7. Sidney, J. B., "Decomposition Algorithms for Single-Machine Sequencing with Precedence Relations and Deferral Costs," *Operations Res.*, Vol. 23 (1975), pp. 283–298.
8. Smith, W. E., "Various Optimizers for Single-Stage Production," *Naval Res. Logist. Quart.*, Vol. 3 (1956), pp. 59–66.