Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

One-Machine Sequencing to Minimize Certain Functions of Job Tardiness

Hamilton Emmons,

To cite this article:
Hamilton Emmons, (1969) One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. Operations Research 17(4):701-715. https://doi.org/10.1287/opre.17.4.701

ONE-MACHINE SEQUENCING TO MINIMIZE CERTAIN FUNCTIONS OF JOB TARDINESS

Hamilton Emmons

Cornell University, Ithaca, New York

(Received September 30, 1968)

This paper first considers the problem of sequencing n jobs on one machine to minimize total tardiness. It proves theorems that establish the relative order in which pairs of jobs are processed in an optimal schedule; frequently they permit the jobs to be completely ordered, thus solving the problem without any searching. In particular, corollaries establish more general conditions than are currently recognized under which sequencing in order of nondecreasing processing times and sequencing in order of nondecreasing due dates are optimal. In general, even large problems may be at least partially ordered to the point that very few schedules remain to be searched. These results are then partly extended to the more general criterion of minimizing a sum of identical, convex, nondecreasing functions of job tardiness, and an efficient algorithm is proposed.

WE CONSIDER the problem of ordering or sequencing a set, $J=\{J_1, J_2, \cdots, J_n\}$, of n jobs on one facility to satisfy the criteria defined below. All jobs are available simultaneously at time zero, and job J_i has a processing time p_i and a due date d_i that are known in advance, for $i=1(1)n$. Setup times are neglected; that is, they are assumed to be independent of sequence (as are processing times) and hence may be included in p_i. Thus, the total completion time or flow time, $p=\sum_J p_i$, is a constant. Generally, we use the notation of CONWAY, MAXWELL, AND MILLER.[1]

The *lateness* of J_i is defined as $L_i=C_i-d_i$ where C_i is the completion time of J_i. It may be positive or negative, depending on whether the job is completed on time ($C_i \leqq d_i$) or not. It is well known that the total lateness, $L=\sum_J L_i$, is minimized by sequencing in order of nondecreasing processing times, known as 'shortest-processing-time,' or SPT, sequencing. *Tardiness* is defined as rectified lateness, i.e., the tardiness of J_i is $T_i=\max(0, L_i)$. Minimum total tardiness, $T=\sum_J T_i$, (or minimum mean tardiness, which differs only by the constant factor $1/n$), is often a more reasonable criterion than minimum total lateness, specifically, when it is important to get jobs done on time, or failing that, as soon as possible after their due dates (but there is no value in increasing the earliness of an already on-time job). Total tardiness is a much less tractable criterion with which to work, however, and no simple rule is known which minimizes

it, even though certain special cases are known, namely: (1) the SPT schedule minimizes total tardiness if all jobs have positive tardiness thereby, and (2) sequencing in order of nondecreasing due dates (the 'earliest due date' or EDD schedule) minimizes total tardiness if at most one job has positive tardiness thereby. If these conditions are not satisfied, several algorithms have been proposed[2,3,4,6] that involve searching some subset of schedules. The algorithms are a significant improvement over exhaustive search, but they remain laborious and applicable only to relatively small problems.

For the objective of minimum total tardiness, the ensuing results will: (1) provide more general sufficient conditions for the optimality of SPT and of EDD scheduling, and (2) in case they are still not satisfied, eliminate most schedules from consideration by providing conditions under which certain jobs can be said to precede other jobs. Often they give enough information to specify an optimal schedule. In any case, they lead to a highly efficient algorithm. Most of the results carry over to the more general objective of minimizing $\sum_J g(T_i)$ where $g(\cdot)$ is any convex, nondecreasing penalty function. Hereafter it will be understood that 'an optimal schedule' refers to this type of criterion. If several alternative optima exist, we shall be content to find any one of them.

EXISTENTIAL VERSUS UNIVERSAL PROPERTIES

THE THEOREMS that follow provide only existential properties: 'There exists an optimal schedule with property A,' rather than universal properties: 'All optimal schedules have property A.' This raises the question: Can we use the theorems repeatedly, and accumulate or pool the results?

Universal properties can, of course, be accumulated. That is, 'All optimal schedules have property A' and 'All optimal schedules have property B' imply 'All optimal schedules have properties A and B.' Thus, in any optimization problem where we wish to maximize or minimize an objective function over a discrete domain, universal properties can be used cumulatively to narrow down the subset of feasible points over which we must search for an optimum.

Existential properties cannot generally be accumulated. Fortunately, the properties we shall derive can be, even though they are not universal. For instance, in Theorem 1, to prove that there exists an optimal schedule in which J_j precedes J_k, we show that if a schedule lacks the property that J_j precedes J_k, we can exchange the two jobs without increasing the penalty function. Thus, if we find by Theorem 1 that there exist optimal schedules with J_a before J_b, J_c before J_d, etc., then any optimal schedule that lacked any of these properties could not be harmed by making as many interchanges as necessary to obtain an optimal schedule with all the properties.

From now on, we shall use the notation '$j \leftarrow k$,' which may be read: 'J_j precedes J_k in an optimal schedule,' to mean strictly: 'There exists an optimal schedule that has all the properties already established and in which J_j precedes J_k.' We must take care that this cumulative feature is maintained, and that internal inconsistencies are forbidden. For instance, we must not permit $j \leftarrow k$, $k \leftarrow m$, and $m \leftarrow j$ simultaneously, as might occur if the relative order of the three jobs were actually immaterial. If Theorem 1 indicates that $j \leftarrow k$ and $k \leftarrow m$, it will indicate also that $j \leftarrow m$, as can easily be verified.

The same ordering notation will be used with respect to sets of jobs. We shall let A_i and B_i be the sets of indices of all jobs that, at any point, have been shown to come after and before J_i, respectively, and shall write '$i \leftarrow A_i$' and '$B_i \leftarrow i$.' If the set of jobs that follow J_i is $\{J_j, J_k, \cdots\}$ we may write '$A_i = \{j, k, \cdots\}$' or '$i \leftarrow j, k, \cdots$' as convenient.

SOME THEOREMS ON ORDERING JOBS TO MINIMIZE TOTAL TARDINESS

HEREAFTER, JOBS will be understood to be indexed in order of nondecreasing processing times ($p_1 \leqq p_2 \leqq \cdots \leqq p_n$) and, in case of equality, in order of nondecreasing due dates. Thus, $j < k$ implies $p_j < p_k$ or $p_j = p_k$ and $d_j \leqq d_k$.

First, it is obvious that if, for any job J_i, $d_i > p$, where p is the total processing time for all jobs, then J_i can be processed last, since it is never tardy in any case. Such jobs should be removed from the problem initially. In doing so, p is reduced and other jobs may successively become eligible for removal. We assume hereafter that the problem has been reduced as far as possible by this means so that $d_i < p$, for all i.

We now consider the important special case of minimum total tardiness. This simplifies the arguments and enables us to prove a stronger Theorem 1. We shall show later that Theorems 2 and 3, and part of Theorem 1, hold for the more general objective.

THEOREM 1. *For any two jobs J_j and J_k with $j < k$, if* (1) $B_k \leftarrow k$ *and* (2) $d_j \leqq \max(\sum_{B_k} p_i + p_k, d_k)$, *then* $j \leftarrow k$ (*that is,* $j \epsilon B_k$ *and* $k \epsilon A_j$).

Proof. Consider any schedule that satisfies hypothesis (1) and in which J_k precedes J_j. We shall show that interchanging the two jobs must decrease, or possibly leave unchanged, the total tardiness. Denote by W and C the times at which J_k begins and J_j ends, respectively (*see* Fig. 1). Clearly, all jobs that precede J_k or follow J_j in the original schedule are unaffected. All jobs between J_k and J_j are advanced in time by an amount $p_k - p_j \geqq 0$, which can only decrease or leave unchanged their tardiness. The changes in tardiness of J_j and J_k are considered as follows, after noting that hypothesis (2) means that either $d_j \leqq d_k$ or $d_j \leqq \sum_{B_k} p_i + p_k$ (or both).

(a) Suppose $d_j \leqq d_k$. We consider three subcases.

1. If $d_j \leqq d_k < C$, as illustrated in Fig. 1, then the *decrease* of tardiness of J_j is $\Delta T_j = C - \max(W+p_j,\ d_j)$, the *increase* of tardiness of J_k is $\Delta T_k = C - \max(W+p_k,\ d_k)$, and the net decrease due to the two changes is $\Delta T_j - \Delta T_k = \max(W+p_k,\ d_k) - \max(W+p_j,\ d_j)$, which is nonnegative since $d_k \geqq d_j$ and $p_k \geqq p_j$.

2. If $d_j \leqq C \leqq d_k$, $\Delta T_j - \Delta T_k = \Delta T_j \geqq 0$.

3. If $C < d_j \leqq d_k$, $\Delta T_j = \Delta T_k = 0$. Thus in all cases the total decrease of tardiness is positive, or at worst zero, so the change should be made.

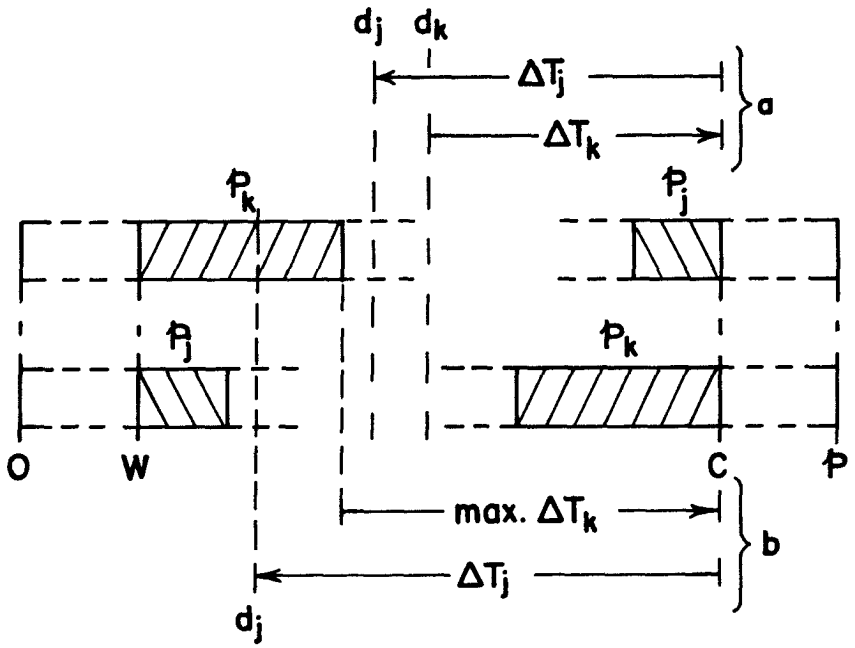

Fig. 1. The effect of interchanging two jobs.

(b) Suppose $d_j \leqq \sum_{B_k} p_i + p_k$. By hypothesis (1), $W \geqq \sum_{B_k} p_i$, so that $d_j \leqq W + p_k$. Then again $\Delta T_j - \Delta T_k = \max(W+p_k,\ d_k) - \max(W+p_j,\ d_j)$ is nonnegative, since $W+p_k \geqq \max(W+p_j,\ d_j)$. An instance of this, too, is shown in Fig. 1.

Corollaries 1.1–1.4 follow from Theorem 1 with $B_k = \Phi$.

COROLLARY 1.1. *If J_1 has the property $d_1 \leqq \max(p_i,\ d_i)$, for all $i>1$, then J_1 is first in an optimal schedule.*

Note that as soon as we find $d_1 \leqq p_j$ for some j, we need not check the

remaining jobs J_i, $i>j$, since $\{p_i\}$ forms a nondecreasing sequence. For instance, J_1 is first if $d_1 \leqq p_2$.

COROLLARY 1.2. *If J_n has the property $\max(p_n, d_n) \geqq d_i$, for all $i<n$, then J_n is last in an optimal schedule.*

COROLLARY 1.3. *The SPT schedule is optimal if it is identical with the EDD schedule.*

Proof. Since $d_1 \leqq d_i$ for all $i>1$, J_1 is first by Corollary 1.1. Similarly, Theorem 1 tells us that $2 \leftarrow i$ for all $i>2$, so that J_2 is second; etc.

This last corollary, which simply requires that jobs with larger processing times have later due dates, can be applied when due dates are assigned proportional to processing times ($d_i = cp_i$ with c a positive constant, for all i) and when due dates are all equal. Simple though it is (and note that it does not make use of the full power of the theorem), apparently it has not previously been remarked. (The special case of constant due dates was noted by ROOT.[5]) The proof suggests a possible general procedure for finding the optimal schedule: Find the job that must be done first or last, remove it from the problem, and repeat. To remove a job, say J_k, from the problem, given that we have decided to process it first, we have only to re-reference our time scale to start at time p_k and proceed to analyze the remaining $n-1$ jobs. This merely means setting d_i to $d_i - p_k$, for $i \neq k$. Some due dates may become negative, but this need not worry us. Using this approach, we can establish:

COROLLARY 1.4. *The SPT schedule is optimal if $d_j + p_j \leqq \sum_{i=1}^{j+1} p_i$ for $j = 1(1)n-1$.*

Proof. For $j=1$, the inequality becomes $d_1 \leqq p_2$, which by Corollary 1.1 implies that J_1 is first. Removing it from the problem, we renumber the jobs, getting new parameters $p_i' = p_{i+1}$, $d_i' = d_{i+1} - p_1$, for $i = 1, 2, \cdots, n-1$. Now for $j=2$, the inequality $d_2 \leqq p_1 + p_3$ becomes $d_1' \leqq p_2'$, so that J_2 is second; etc.

Thus, $d_j \leqq C_j + (p_{j+1} - p_j)$ may replace $d_j < C_j$ as a sufficient condition for the optimality of the SPT schedule.

Theorem 1 gives us conditions under which a shorter job can be said to precede a longer one. We next consider when it is possible to say that a longer job comes earlier.

THEOREM 2. *For any two jobs J_j and J_k with $j<k$, if (1) $k \leftarrow A_k$ and $B_k \leftarrow k$, (2) $d_j > \max(\sum_{B_k} p_i + p_k,\ d_k)$, and (3) $d_j + p_j \geqq \sum_{A_k'} p_i$, where $A_k' = \{i : i \notin A_k\}$, then $k \leftarrow j$.*

Proof. Suppose J_j precedes J_k in any schedule satisfying hypothesis (1). Let W and C be the times at which J_j begins and J_k ends, respectively. Thus, $C \leqq \sum_{A_k'} p_i$ and $W \leqq \sum_{A_k'} p_i - p_j - p_k$. Consider moving J_j to a position immediately after J_k (*see* Fig. 2). Again, jobs before W

and after C are unaffected. J_k and all jobs between J_j and J_k are now processed earlier by an amount p_j. Only J_j can have an increase in tardiness:

$$\Delta T_j = \begin{cases} 0, & d_j \geqq C, \\ C - d_j, & d_j < C. \end{cases}$$

Note that the alternative, $\Delta T_j = C - (W + p_j)$, is ruled out by hypothesis (3), which implies that $d_j \geqq W + p_j$ because $W \leqq \sum_{A_k'} p_i - p_j - p_k$ and $W + p_j \leqq \sum_{A_k'} p_i - p_k \leqq \sum_{A_k'} p_i - p_j$.

If $\Delta T_j = 0$, then clearly moving J_j after J_k is desirable. For $\Delta T_j > 0$ we must have $d_j < C$, which by hypothesis (2) means that $d_k < C$. It follows that the decrease of tardiness of J_k is $\Delta T_k = C - \max(C - p_j, d_k)$. Finally, we can conclude that $\Delta T_k \geqq \Delta T_j$, i.e., $d_j \geqq \max(C - p_j, d_k)$, since

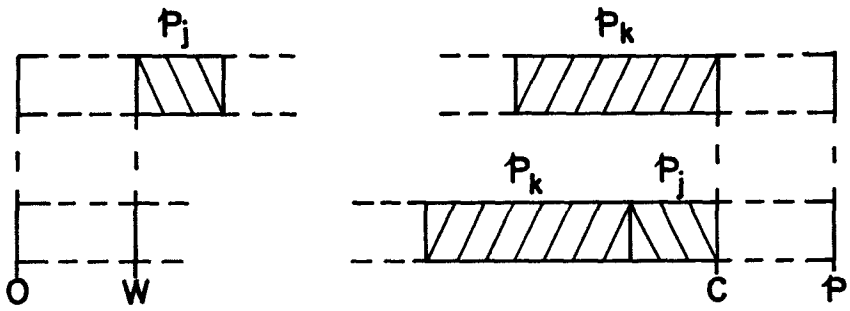

Fig. 2. The effect of postponing a job.

$d_j > d_k$ by hypothesis (2) and $d_j \geqq \sum_{A_k'} p_i - p_j \geqq C - p_j$ by hypothesis (3). Thus, in this case too, postponing J_j is advantageous.

Corollaries 2.1–2.2 follow from Theorem 2 with $A_k = \Phi$.

COROLLARY 2.1. *If* $d_j = \max_i\{d_i\}$ *and* $d_j + p_j \geqq p$, *then* J_j *is last in an optimal schedule.*

Proof. Since $d_j > d_i$ for all $i < j$, jobs J_i precede J_j by Theorem 1. Similarly, jobs J_i, $i > j$, precede J_j by Theorem 2.

This Corollary in turn implies:

COROLLARY 2.2. *The EDD schedule is optimal if it results in waiting (or starting) times* $W_i \leqq d_i$ *(i.e.,* $C_i \leqq d_i + p_i$*) for all jobs* J_i.

Proof. We know that, by putting the job J_j with the latest due date in last position, we get $W_j \leqq d_j$. The last job in a schedule has the property, $W_j = p - p_j$. Thus, $p \leqq d_j + p_j$ and Corollary 2.1 tells us that J_j should indeed be last. Remove it from the problem, redefine p, and repeat.

Observe that this is much stronger than the well known result cited earlier: The EDD schedule is optimal if at most one job has $C_i > d_i$. Corol-

lary 2.2 permits any or all jobs to have positive tardiness, as long as the tardiness of each job does not exceed its processing time.

COROLLARY 2.3. *If J_j and J_k with $j<k$ are to occur consecutively after a waiting time W, then they should be sequenced according to the rule: $j\leftarrow k$ if and only if $d_j \leqq \max(W+p_k, d_k)$.*

This corollary [of which an equivalent statement has been given by SMITH:[7] $j\leftarrow k$ if and only if $\max(W+p_j, d_j) \leqq \max(W+p_k, d_k)$] could be used to find a good, though not always optimal, schedule very simply. Start with as good a schedule as possible (for instance, the SPT schedule) and make all adjacent-pair comparisons, interchanging whenever an improvement results and continuing as long as improvements are possible.

One other result that is easy to establish and sometimes useful will now be presented.

THEOREM 3. *For any two jobs J_j and J_k with $j<k$, if (1) $j\leftarrow A_j$ and (2) $d_k \geqq \sum_{A_j'} p_i$, where $A_j' = \{i: i \notin A_j\}$, then $j\leftarrow k$.*

Proof. As usual, we suppose J_j follows J_k in any schedule satisfying hypothesis (1), and show that interchanging the two jobs can only decrease total tardiness. Hypothesis (1) implies that $C \leqq \sum_{A_j'} p_i$ (*see* Fig. 1) and hence, by hypothesis (2), $d_k \geqq C$. This means that J_k maintains zero tardiness after the interchange. Since all other jobs either remain unmoved or are advanced in time, the only possible changes in tardiness are decreases.

Observe that the kind of inconsistency warned against in the second section could arise here. If J_j and J_k both have very late due dates, Theorem 2 might indicate that $k\leftarrow j$ while Theorem 3 demonstrates the opposite. This would occur only in case it made no difference which came first, since both would be early either way. We might simply permit the contradiction to take place, with the later test overruling the earlier, but instead we shall assume that once a pair of jobs has been ordered, no further comparisons are made between them.

Corollaries 1.1–1.4 and 2.1–2.2 use only the information contained in the definition of the problem. Whenever they prove fruitless, considerable ordering information will generally be obtained by applying the theorems, starting with $A_i = B_i = \Phi$ for all i. As pairs of jobs are ordered, the sets A_i and B_i will grow, thereby strengthening the theorems and often making possible additional ordering. For example, when applying Theorem 2, we can compare a job J_k with all jobs $J_j, j<k$, which satisfy hypothesis (2) (for those that do not, $j\leftarrow k$ by Theorem 1). In case any $p_j + d_j \geqq \sum_{A_k'} p_i$, we can add J_j to the set A_k. This diminishes $\sum_{A_k'} p_i$ and makes it easier to show that J_k precedes other jobs in A_k'. In this way Theorem 2, and similarly Theorems 1 and 3, increase in power as new information is acquired. Eventually, for some J_i, either A_i or B_i may include

all other jobs, in which case J_i will be first or last and the problem can be simplified by its removal.

This is not to say that this procedure will always yield the complete solution directly. For instance, in Theorem 2, unless A_k is already a large set of jobs, hypothesis (3) may require d_j to be very large before we can conclude that $k \leftarrow j$. We can relax this requirement somewhat only at the expense of additional stringent hypotheses. Such theorems can be proved, but they are rarely useful, and the additional computation required to apply them soon gets out of hand. There will, therefore, be occasions when an impasse is reached. One way to break the ice is to branch on the basis of some assumption; for example, the relative order of two jobs, J_j and J_k. The optimal schedules must now be found assuming $j \leftarrow k$ and assuming $k \leftarrow j$, and the total tardiness of each must be computed to find the minimum.

THE ALGORITHM

WE NOW PROPOSE an algorithm to find a schedule that minimizes total tardiness using the above theorems and branching whenever necessary. To organize an attack, we think in terms of singling out, if possible, a first or last job and removing it.

Step 1. Place last, and remove from the problem, the job with largest processing time on the basis of Theorem 1, if possible. Repeat as often as possible.

Step 2. Place last and remove the job with latest due date on the basis of Corollary 2.1, if possible. Repeat as often as possible.

Step 3. Return to Step 1 if Step 2 proved fruitful. Otherwise, proceed.

Note that it is desirable to alternate between Steps 1 and 2 as long as is profitable, since success at either step may make additional placements possible at the other step.

Step 4. Generate the ordered set of jobs, $F=\{J_a, J_b, \cdots, J_e, J_f\}$ (which we may represent by the set of indices, $F=\{a, b, \cdots, e, f\}$) as follows, assuming as usual that the jobs are indexed in order of nondecreasing processing times:

(a) Initialize 'the remaining set' to be J, and its 'first job, J_x,' to be J_1.

(b) Eliminate from the remaining set all jobs $\{J_i\}$ beyond the first, J_x, which satisfy $d_x \leqq \max(p_i, d_i)$.

(c) Remove J_x from the remaining set and enter it as the next job in F.

(d) Return to b as long as jobs remain. Then proceed.

Set F consists of all jobs that could be first (no other job has been shown to precede them) according to Theorem 1. It has the properties $p_a \leqq p_b \leqq \cdots \leqq p_f$ and $d_a > d_b > \cdots > d_f$.

Step 5. Place first, and remove from the problem, the last job in F,

J_f, if possible, using Theorem 2 to show that $f \leftarrow i$ for all $i < f$. If J_f can be shown to be first, return to Step 4. Otherwise, proceed.

The way F was generated ensures that $f \leftarrow i$ for $i > f$. Removing J_f from the problem entails subtracting p_f from all due dates.

Step 6. Place first and remove the penultimate job in F, J_e, if possible by:

(a) Showing that $e \leftarrow i$ for all $i < e$ using Theorem 2.

(b) Showing that $e \leftarrow f$ using either Theorem 1 or Theorem 3. If J_e can be shown to be first, return to Step 4. Otherwise, proceed.

Step 6b will suffice to show that $e \leftarrow i$ for all $i > e$, because $e \leftarrow i$ for all i, $e < i < f$, and $f \leftarrow i$ for all $i > f$.

Step 7. Branch on the assumptions $e \leftarrow f$ and $f \leftarrow e$. Return to step 5.

The branching assumption removes either J_e or J_f from F, thereby hopefully breaking the deadlock. We choose to branch on the two jobs with largest processing times among those still in contention for first place so as to eliminate one candidate for first place and to add jobs to A_e and B_f (or vice versa) that will maximally affect $\sum_{A_k'} p_i$ and $\sum_{B_f} p_i$.

It may be necessary to branch several times to solve a large problem. One possibility that arises is that, having assumed $j \leftarrow k$, we find, after further ordering of jobs and perhaps further branching, that one of our theorems now indicates $k \leftarrow j$. Clearly, any branch that leads to an inconsistency can be abandoned. The success of the algorithm will depend on the power of the theorems, and the degree to which branching is minimized.

Of course, for large problems where we can expect a good deal of branching and consequently of generating alternative partial schedules that eventually must be compared, standard branch-and-bound techniques (as used in reference 2, for example) should be incorporated into the algorithm. Thus, we should compute the total tardiness of each partial schedule as it is extended, and work should proceed on whichever branch currently has the smallest total. A partial schedule should be discarded whenever it is 'dominated' by another; that is, whenever its assigned jobs are a subset of another's and its total tardiness greater. These ideas are not new, and will not be further detailed here.

EXAMPLES

To TEST the effectiveness of the algorithm, ten examples were investigated as a start, each consisting of ten jobs where the numbers p_i and d_i, $i = 1(1)10$, were drawn from a table of two-digit random numbers. In seven of the ten examples, direct application of the theorems gave the answer without branching.

Example 1. We illustrate with the example presented in Table I.

We could start by checking Corollaries 1.3, 1.4, and 2.2 (all fail here), though if any of them pertains it will soon become evident in applying the algorithm.

Step 1.

(a) J_{10} is last, since $\max(p_{10}, d_{10}) = 97 \geqq d_i$ for all $i < 10$ (*see* Corollary 1.2). Remove J_{10} and repeat.

(b) J_9 is last (i.e., ninth over-all), since $\max(p_9, d_9) = 80 \geqq d_i$ for all $i < 9$.

(c) J_8 is last, since: (i) With $B_8 = \Phi$, $\max(p_8, d_8) = 66 \geqq d_i$ for $i = 1, 3, 5, 6, 7$, so that $1, 3, 5, 6, 7 \leftarrow 8$. (ii) With $B_8 = \{1, 3, 5, 6, 7\}$, $\max(\sum_{B_8} p_i + p_8, d_8) = 230 \geqq d_i$ for $i = 2, 4$.

(d) J_7 and J_6 follow in the same way as J_8.

(e) For J_5, $B_5 = \Phi$ implies $B_5 = \{1, 3\}$. Now, $\max(\sum_{B_5} p_i + p_5, d_5) = 54 < d_2$ or d_4, so we cannot conclude that J_5 is last.

We have now reduced the problem to the first five jobs.

TABLE I

DATA FOR EXAMPLE 1

i	1	2	3	4	5	6	7	8	9	10
p_i	6	12	16	23	32	49	61	66	80	97
d_i	25	73	31	67	32	31	57	64	15	55

Step 2. For J_2, $d_2 + p_2 = 85 < p = 89$, so that Corollary 2.1 fails to show that J_2 is last.

Step 3. Proceed to Step 4.

Steps 4, 5. $F = \{1\}$, so J_1 is first (*see also* Corollary 1.1). Removing it from the problem, we set (d_2, d_3, d_4, d_5) to $(67, 25, 61, 26)$, and return to Step 4.

Step 4. $F = \{2, 3\}$.

Step 5. J_3 is first (i.e., second over-all), because $A_3 = \{4, 5\}$, $A_3' = \{2, 3\}$, and $d_2 + p_2 = 79 \geqq \sum_{A_3'} p_i = 28$, so that $3 \leftarrow 2$. Remove J_3, set (d_2, d_4, d_5) to $(51, 45, 10)$, and return to Step 4.

Step 4. $F = \{2, 4, 5\}$. All jobs are candidates for first place.

Step 5. J_5 is first, since: (i) With $A_5 = \Phi$, $\sum_{A_5'} p_i = p = 67 \leqq d_j + p_j$ for $j = 4 (d_4 + p_4 = 68)$, but not for $j = 2 (d_2 + p_2 = 63)$; thus, $5 \leftarrow 4$. (ii) With $A_5 = \{4\}$, $\sum_{A_5'} p_i = 44 \leqq d_2 + p_2$, so $5 \leftarrow 2$; remove J_5, set (d_2, d_4) to $(19, 13)$, and return to Step 4.

Step 4. $F = \{2\}$, so J_2 is first.

Finally, the optimal schedule is $(1, 3, 5, 2, 4, 6, 7, 8, 9, 10)$. Since this solution was obtained directly, without any searching, it was never necessary to compute the total tardiness for any schedule.

Example 2. Table II gives the data for a problem in which branching is required. We see that, while $p_i \leqq p_{i+1}$, $d_i > \max(p_{i+1}, d_{i+1})$ for all i. This is a 'perverse' situation: The longer the job, the sooner it is wanted. Whenever it arises, Theorem 1 is useless. When, in addition, $d_i + p_i < p$ for all i so that Theorem 2 is of no help, then Theorem 3 cannot help either (it becomes useful only after some ordering information is available), and we cannot proceed without branching. Following the algorithm, we get:

Steps 1, 2, 3. No result.

Step 4. $F = \{1, 2, 3, 4\}$.

Steps 5, 6. No result.

Step 7.

(a) If $3 \leftarrow 4$, so that $F = \{1, 2, 3\}$:

Step 5. J_3 is first, since $A_3 = \{4\}$ and $\sum_{A_3'} p_i = 85 \leqq d_j + p_j$ for $j = 1, 2$, so that $3 \leftarrow 1, 2$. Remove J_3, set (d_1, d_2, d_4) to $(35, 25, 15)$, and return to Step 4.

Step 4. $F = \{1, 2\}$.

TABLE II

DATA FOR EXAMPLE 2

i	1	2	3	4
p_i	20	30	35	45
d_i	70	60	55	50

Step 5. J_2 is first, since $A_2 = \{4\}$ and $\sum_{A_2'} p_i = 50 \leqq d_1 + p_1 = 55$, so that $2 \leftarrow 1$. Remove J_2, set (d_1, d_4) to $(5, -15)$, return to Step 4.

Steps 4, 5. $F = \{1\}$, so J_1 is first.

Optimal schedule (given $3 \leftarrow 4$) $= (3, 2, 1, 4)$, with $T = 100$.

(b) If $4 \leftarrow 3$, so that $F = \{1, 2, 4\}$:

Step 5. For J_4, $A_4 = \{3\}$, $\sum_{A_4'} p_i = 95 > d_i + p_i$ for $i = 1, 2$, so we cannot conclude that J_4 is first.

Step 6. For J_2, we cannot show that $2 \leftarrow 4$.

Step 7. (i) If $2 \leftarrow 4$, so that $F = \{1, 2\}$: *Step 5.* J_2 is first, since $A_2 = \{3, 4\}$ and $\sum_{A_2'} p_i = 50 \leqq d_1 + p_1$ so that $2 \leftarrow 1$. Remove J_2, set (d_1, d_3, d_4) to $(40, 25, 20)$, and return to Step 4. *Step 4.* $F = \{1, 3\}$. But this implies that $3 \leftarrow 4$, which contradicts an earlier assumption. Abandon this branch. (ii) If $4 \leftarrow 2$, so that $F = \{1, 4\}$: *Step 5.* J_4 is first, since $A_4 = \{2, 3\}$ and $\sum_{A_4'} p_i = 65 \leqq d_1 + p_1 = 90$, so that $4 \leftarrow 1$. Remove J_4, set (d_1, d_2, d_3) to $(25, 15, 10)$, and return to Step 4. *Steps 4, 5.* $F = \{1\}$, so J_1 is first. Remove J_1, set (d_2, d_3) to $(-5, -10)$ and return to Step 4. *Steps 4, 5.* $F = \{2\}$, so J_2 is first.

Optimal schedule (given $4 \leftarrow 3$) $= (4, 1, 2, 3)$, with $T = 110$.

Thus, it was necessary to evaluate two schedules (no bounding was attempted) to determine the optimum: (3, 2, 1, 4).

An objection that can be raised to the ten sample problems mentioned above, besides their small size, is that the numbers were all drawn from the same uniform distribution with mean 50, thereby making the due dates of the same order of magnitude as the processing times. When the due dates are all small relative to the total flow time p, it only requires a few assignments of jobs at the start to make all the rest tardy, whereupon SPT sequencing becomes optimal. In Example 1, after finding the first three jobs (1, 3, 5, $\cdots$), Corollary 1.1 can be applied repeatedly to place the other seven jobs in SPT order. Indeed, we might expect that often very large problems of this type will be soluble with little greater effort than that required for 10-job problems, since once we have determined the first few jobs no further computation is required, regardless of the number of jobs remaining.

We might anticipate greater difficulty with problems in which due dates tend to be larger than processing times. Recall that we have restricted $d_i < p$ for all i, since otherwise J_i could be left until last without cost. Thus, we are concerned with due dates ranging up to the completion time of the final job. In such a problem, however, Theorems 2 and 3 tend to be more useful, since they require certain due dates to be greater than sums of processing times and this is now more likely.

Example 3. To see how these effects balance out, a 50-job problem was partially investigated with a hand calculator. The processing times were random numbers between 1 and 1000 and the due dates random numbers between 1 and 20,000. (They could not be much larger, since $E(p) = 25{,}000$. As it turned out, $p = 26{,}069$.) Step 1 of the algorithm eliminated the seven longest jobs, and step 2 then assigned seven more jobs. No additional assignments were possible using step 1. Step 5 placed four jobs at the start of the schedule and step 6 assigned two more. Having thus reduced the problem from 50 jobs to 30 jobs, it was necessary to branch for the first time. The calculations were then continued for one of the branches, and whenever further branching was necessary, one of the two paths was followed. Only five branch points were needed to produce a complete schedule. Furthermore, the last branch point occurred with only four jobs left; it was easy to trace both paths to completion and, by comparing the total tardiness of just those four jobs, to eliminate the inferior option. The other branches would require consideration of additional schedules of 13, 24, 27, and 30 jobs. Each of these might, of course, be more or less difficult to solve than the corresponding branch that was traced through. However, one further helpful effect was noted. Often two branches differ only in the first few jobs, e.g., when the assumption

$e \leftarrow f$ leads to the schedule $(e, g, h, f, \cdots)$ while $f \leftarrow e$ results in $(f, e, h, g, \cdots)$. At this point, the same jobs have been assigned and the branches have come together again: We need only choose the best of these four-job sequences and proceed. This is merely one example of the branch-and-bound procedure referred to earlier.

While the algorithm has not been coded, it is reasonable to expect that very large problems can be solved using it. The theorems appear powerful enough to do most of the ordering with only occasional need to branch; hence, the computational effort should increase only a little more rapidly than linearly with the number of jobs.

SEQUENCING TO MINIMIZE $\sum_J g(T_i)$, WHERE g IS CONVEX AND INCREASING

Let us now consider the more general objective function, namely $\sum_J g(T_i)$, where the loss function $g(\cdot)$ is convex and increasing for $T_i \geqq 0$. Note that each job has the same loss function. In attempting to extend the foregoing theorems, we again consider shifting the positions of certain jobs in a schedule, but now we must compare the resulting changes in the various $g(T_i)$. Because of the nonlinearity of $g(\cdot)$, such a change, $\Delta g(T_i)$, depends on the actual tardiness of J_i before and after the change, which we may denote T_{ib} and T_{ia}, respectively, as well as on the change in tardiness, $\Delta T_i = |T_{ib} - T_{ia}|$. Thus, if we interchange J_j and J_k as in the proof of Theorem 1, sufficient conditions for $\Delta g(T_j) = g(T_{jb}) - g(T_{ja})$ to exceed $\Delta g(T_k) = g(T_{ka}) - g(T_{kb})$ are: (1) $\Delta T_j \geqq \Delta T_k$, and (2) $T_{jb} \geqq T_{ka}$ (*see* Fig. 3). The second condition is added to guarantee that the decrease in T_j takes place over a portion of the curve that is at least as steep as that for the increase in T_k. With this in mind, it is easily established (proofs will be omitted) that part (a) of the proof of Theorem 1 goes through as before, while part (b) no longer holds. Thus, we can state for our new objective function only the following much weaker result:

Theorem 1*. *For any two jobs J_j and J_k with $j<k$, if $d_j \leqq d_k$, then $j \leftarrow k$.*

This is still enough to give us the important special case:

Corollary 1.3*. *The SPT schedule is optimal if it is identical with the EDD schedule.*

It can similarly be shown that Theorem 2 and its two key corollaries still hold, modified to take into account the alteration in Theorem 1.

Theorem 2*. *For any two jobs J_j and J_k with $j<k$, if (1) $k \leftarrow A_k$, (2) $d_j > d_k$, and (3) $d_j + p_j \geqq \sum_{A_k'} p_i$, then $k \leftarrow j$.*

Corollary 2.1*. *If $d_j = \max_i \{d_i\}$ and $d_j + p_j \geqq p$, then J_j is last.*

Corollary 2.2*. *The EDD schedule is optimal if it produces $W_i \leqq d_i$ for all i.*

Theorem 3 also remains valid for the generalized objective.

The same algorithm could be used with minor modifications; to wit:

(1) Step 4b should end "... which satisfy $d_x \leqq d_i$."

(2) Step 6b should end "... using Theorem 3."

It will, of course, be considerably less efficient (i.e., more branching will be required). Step 1 will now only apply if the job with largest processing time also has latest due date. Thus, the first three steps will be much less productive; indeed, no jobs could be placed last in any of the three examples considered above. In a large problem with due dates much larger than processing times, the power of Steps 4–6 are not appreciably diminished at

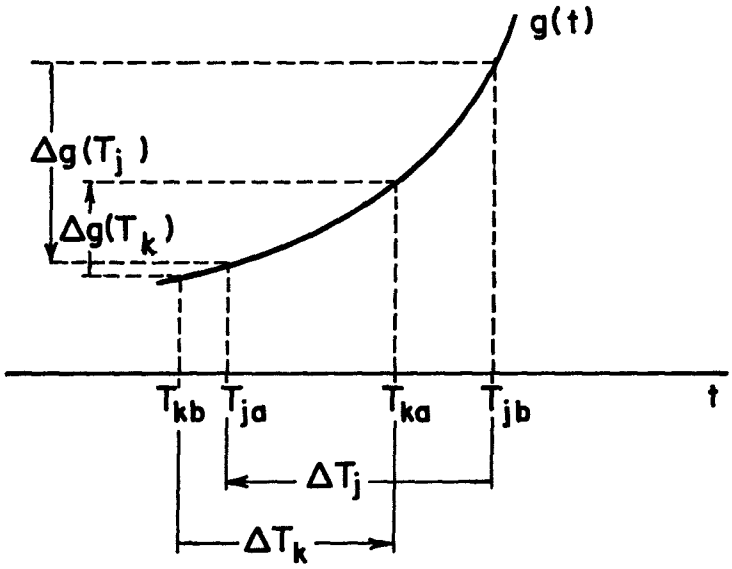

Fig. 3. The effect of job interchange on the generalized objective.

first. For instance, in Example 3, we can still place six jobs at the start of the schedule before branching, then three more before branching again, etc. In this way, a large problem can be efficiently reduced. However, the small problems that result, such as Example 1, will now require considerable branching, and branch-and-bound techniques will be needed.

ACKNOWLEDGMENTS

THE AUTHOR is indebted to G. L. NEMHAUSER for suggesting the generalization of the objective function, and also to the painstaking and constructive criticism of a referee. The research was supported in part by the National Science Foundation.

REFERENCES

1. R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, Mass., 1967.
2. S. E. Elmaghraby, "The One Machine Sequencing Problem with Delay Costs," *J. Ind. Eng.* **19,** 105–108 (1968).
3. M. Held and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *SIAM* **10,** 196–210 (1962).
4. E. L. Lawler, "On Scheduling Problems with Deferral Costs," *Management Sci.* **11,** 280–288 (1964).
5. J. G. Root, "Scheduling with Deadlines and Loss Functions on k Parallel Machines," *Management Sci.* **11,** 460–475 (1965).
6. A. Schild and I. J. Fredman, "Scheduling Tasks with Linear Loss Functions," *Management Sci.* **7,** 280–285 (1961).
7. W. E. Smith, "Various Optimizers for Single-Stage Production," *Naval Res. Log. Quart.* **3,** 59–66 (1956).