MANAGEMENT SCIENCE
Vol. 15, No. 1, September, 1968
Printed in U.S.A.

AN n JOB, ONE MACHINE SEQUENCING ALGORITHM FOR MINIMIZING THE NUMBER OF LATE JOBS[1]

J. MICHAEL MOORE

The University of Michigan

An algorithm, computationally feasible for large problems, has been formulated for sequencing n jobs through a single facility to minimize the number of late jobs. This algorithm is then extended to solve the problem in which each job is associated with a continuous, monotone non-decreasing deferral cost function. The object is to produce a schedule where the maximum deferral cost incurred is minimal.

Introduction

The problems considered in this paper belong to a general class of problems in which a finite set of jobs must be sequenced through a single facility, minimizing some function of the lateness penalties incurred. References [1] and [10] contain good summaries of the known results in this area. The special problem of minimizing the sum of the lateness penalties is treated in references [2–9]. References [2], [6], [7] and [8] consider the case of linear penalty functions, reference [9] the case of quadratic penalty functions, and references [3] and [5] the case of general penalty functions. Reference [3] contains the only known algorithm for solving these problems, a dynamic programming technique formulated by Held and Karp. While this method is applicable to the problem defined here, it is generally computationally infeasible for problems of 30 jobs or more. The algorithm developed in this paper, however, consists of only two sorting operations performed on the total set of jobs, and a maximum of $n(n + 1)/2$ additions and comparisons. Consequently, this method will be computationally feasible for very large problems and can be performed manually on many smaller problems.

The problem is to sequence n jobs, $J_1 \cdots J_n$, with known processing times, $t_1 \cdots t_n$, and due-dates $D_1 \cdots D_n$, through a single production facility in such a way as to minimize the number of late jobs. The processing times, which are defined to include set-up and tear-down times, are independent of the sequence. It is assumed that the n jobs are available for production throughout the scheduling period and that the production facility operates continuously until all jobs are completed. No lot-splitting is allowed, so that production on a job continues from start to finish without interruption.

It is assumed that each job can be completed by its' due-date, defined relative to $t = 0$, if the processing of that job were to begin immediately (*i.e.*, $\forall_i$, $t_i \leqq D_i$). If this were not the case, the problem size would be reduced by eliminating those jobs that cannot be completed on time regardless of the sequence.

The Algorithm

The algorithm consists of a decision rule for dividing the set of jobs into two subsets. This division produces an optimal schedule if the jobs in the first set are sequenced according to their due-dates, and are followed by the jobs in the second set in any order.

[1] Received June 1967, and revised March 1968.

Step 1: Order the set of jobs according to the shortest processing time rule and call the resulting ordering $((J_{i_1} \cdots J_{i_n})$ where $t_{i_1} \leqq \cdots \leqq t_{i_n})$ the current sequence.

Step 2: Using the current sequence, find the first late job, J_{i_q}, and go to Step 3. If no such job is found, the algorithm terminates with an optimal schedule obtained by ordering the jobs in the current sequence according to their due-dates followed by the set of jobs that have been rejected in any order.

Step 3: Reorder the jobs, $J_{i_1} \cdots J_{i_q}$, according to the due-date rule producing a sequence of the form:

$$J_{l_1} \cdots J_{l_q}, J_{i_{q+1}} \cdots J_{i_n} \quad \text{where} \quad D_{l_1} \leqq \cdots \leqq D_{l_q}.$$

There are two cases to consider:

1) If all the jobs in the subsequence $J_{l_1} \cdots J_{l_q}$ are early, define the total sequence as the current sequence and go to Step 2.

2) Otherwise reject the job J_{i_q} and remove it from the sequence $J_{l_1} \cdots J_{l_q}$ $J_{i_{q+1}} \cdots J_{i_n}$. Now go to Step 2 with the resulting sequence as the current sequence.

Example

This process is illustrated in the following numerical example:

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
Processing Time	10	6	3	1	4	8	7	6
Due-Date	35	20	11	8	6	25	28	9

Step 1: Ordering the jobs according to the shortest processing time rule results in the following current sequence:

Job	J_4	J_8	J_5	J_3	J_2	J_7	J_6	J_1
Processing Time	1	3	4	6	6	7	8	10
Due-Date	8	11	6	20	9	28	25	35
Completion Time	1	4	8	—	—	—	—	—

Step 2: The result of computing the first 3 completion times indicates that J_5 is the first late job. Therefore, Step 3 is performed.

Step 3: The result of re-ordering the first three jobs according to the due-date rule is shown below.

Job	J_5	J_4	J_3	J_2	J_8	J_7	J_6	J_1
Processing Time	4	1	3	6	6	7	8	10
Due-Date	6	8	11	20	9	28	25	35
Completion Time	4	5	8	14	20	—	—	—

As indicated above, the first three jobs are now early. Hence, Step 2 is reperformed using the above sequence as the current sequence.

Step 2: The first late job is now J_8 as is shown in the table above. Consequently, Step 3 is performed.

Step 3: Ordering the first 5 jobs according to the due-date rule gives the se-

quence below:

Job	J_5	J_4	J_3	J_8	J_2	J_7	J_6	J_1
Processing Time	4	1	6	3	6	7	8	10
Due-Date	6	8	9	11	20	28	25	35
Completion Time	4	5	11	14	20	—	—	—

Since J_8 and J_3 are both late, J_8 is rejected and the new current sequence is shown below:

Job	Current Sequence							Rejected Jobs
	J_5	J_4	J_3	J_2	J_7	J_6	J_1	J_8
Processing Time	4	1	3	6	7	8	10	6
Due-Date	6	8	11	20	28	25	35	9
Completion Time	4	5	8	14	21	29	—	—

Step 2: The first late job is now J_6.

Step 3: Re-ordering the first six jobs according to due-dates gives the following sequence:

Job	Current Sequence							Rejected Jobs
	J_5	J_4	J_3	J_2	J_6	J_7	J_1	J_8
Processing Time	4	1	3	6	8	7	10	6
Due-Date	6	8	11	20	25	28	35	9
Completion Time	4	5	8	14	22	29	—	—

Since J_7 is late, J_6 is rejected, resulting in the following new current sequence.

Job	Current Sequence						Rejected Jobs	
	J_5	J_4	J_3	J_2	J_7	J_1	J_6	J_8
Processing Time	4	1	3	6	7	10	6	8
Due-Date	6	8	11	20	28	35	9	25
Completion Time	4	5	8	14	21	31	—	—

Step 2: All jobs in the current sequence are now early and the algorithm terminates with the above sequence $(J_5, J_4, J_3, J_2, J_7, J_1, J_8, J_6)$ as an optimal schedule.

On the final step of the algorithm it is not necessary to re-order the current sequence according to the due-date rule. Doing so, however, will still produce an optimal schedule. The proof for the optimality of this algorithm follows.

Theoretical Development

Definition: A schedule, S, is defined as a specific ordering, $(J_{i_1} \cdots J_{i_n})$, of the set of jobs to be processed.

For each schedule, S, two sets, E and L, are defined as follows:

$$J_i \, \varepsilon \, E, \text{ iff } C_i \leqq D_i ,$$

$$J_i \, \varepsilon \, \mathrm{L}, \text{ iff } C_i > D_i ,$$

where C_i is the completion time of job J_i in the schedule S.

Definition: For a given schedule, S, let A denote the ordered set defined by ordering the elements of the set E according to their order in the schedule S.

Similarly, let R denote the ordered set defined by ordering the elements of the set L according to their order in the schedule S.

Finally, let π denote an arbitrary permutation of the set R, and let P denote the resulting ordered set.

Lemma 1: Any optimal schedule, S, for a set of jobs, $J = \{J_1 \cdots J_n\}$, with known processing times, $t_1 \cdots t_n$, and due-dates, $D_1 \cdots D_n$, is equivalent (in number late) to the schedule, $S^* = (A, R)$, and to any schedule of the form, $S^{**} = (A, P)$.

Proof: Suppose $S = (J_{i_1} \cdots J_{i_n})$ is not of the form (A, R). Then there must exist a pair of adjacent jobs, J_{i_k} and $J_{i_{k+1}}$, in S such that

$$C_{i_{k+1}} \leqq D_{i_{k+1}} ,$$

and

$$C_{i_k} > D_{i_k} .$$

Hence, define a new optimal schedule, S_1, by interchanging J_{i_k} and $J_{i_{k+1}}$ and S_1 will have $A_1 = A$ and $R_1 = R$. If S_1 is of the form (A, R), we are done. Otherwise, operate on S_1 to obtain S_2, *etc.* This process terminates in a finite number of steps with an optimal schedule S_q where $A_q = A$, $R_q = R$, and $S_q = (A, R) = S^*$.

Lemma 2: Given an optimal schedule, S, of the form, (A, R), a new optimal schedule, $S_D = (A_D, R)$, may be defined by re-ordering the set A according to the due-date rule.

This lemma follows directly from a lemma due to Jackson (4).

Lemma (Jackson): Given a set of jobs, $\{J_1 \cdots J_n\}$, with known processing times, $t_1 \cdots t_n$, and due-dates, $D_1 \cdots D_n$, there exists a schedule having no late jobs if and only if there are no late jobs in the schedule defined by ordering the jobs according to their due-dates.

Since all jobs in the set A are on time, they will still be on time if re-ordered according to their due-dates.

These lemmas show that there is associated with any optimal schedule, S, an optimal schedule $S_D = (A_D, P)$. Consequently, the original problem can be restated as one of producing a schedule of this form where the cardinality of the associated set, E_D, is maximal.

Lemma 3: Suppose there exists an optimal schedule, S, for a set of jobs, J, where L is not empty. Let J^* denote any subset of L. Then for any optimal schedule, $S' = (A', R')$, for the set of jobs $J' = J - J^*$, an optimal schedule, $S'' = (A'', P'')$, for the complete set of jobs, J, can be defined by letting $A'' = A'$ and $L'' = J^* \cup L'$.

Proof: Without loss of generality, assume that S is of the form (A, R) and consider any optimal schedule, $S' = (A', R')$, for the set of jobs J'.

If the cardinality of the set E', denoted $/E'/$, equals the cardinality of the set E, denoted $/E/$, we are done. Therefore assume $/E'/ \neq /E/$.

a) If $/E'/ < /E/$, we can define a new optimal schedule, $S_1 = (A_1, P_1)$, for the set of jobs, J', by letting $A_1 = A$ and $L_1 = L - J^*$. But $/E_1/ = /E/ > /E'/$. Hence, S' is not an optimal schedule for the set of jobs J'—a contradiction. Therefore, $/E'/ \geqq /E/$.

b) If $/E'/ > /E/$, we can define a new optimal schedule, $S_2 = (A_2, P_2)$, for the set of jobs, J, by letting $A_2 = A'$ and $L_2 = L' \cup J^*$. But $/E_2/ = /E'/ > /E/$ and hence S is not an optimal schedule for the set of jobs, J—a contradiction. Consequently, $/E'/ = /E/$.

The problem is now one of developing an algorithm which will find a job, $J_i \varepsilon J$, such that $J_i \varepsilon L$ for some optimal schedule when not all of the jobs in the original set can be completed on time (*i.e.*, the schedule S_d has at least one late job). An optimal schedule for the original set of jobs can then be obtained by repeatedly applying the algorithm and eliminating the jobs found. The process will terminate at some point where the set of jobs, $J^* = (J_{i_1} \cdots J_{i_q})$ has been eliminated, but the algorithm fails to find a job in the set $J - J^*$. An optimal schedule for the original set of jobs, J, is now defined by letting:

$$E = J - J^*,$$

$$L = J^*,$$

and, hence,

$$S = (A_D, P),$$

where A_D is the ordered set obtained by ordering the set E according to the due-date rule. One can show that this schedule is optimal by repeatedly applying lemma 3. (By Jackson's lemma, $S_q = A_D$ is an optimal schedule for the set of jobs $J - J^*$. Then, using lemma 3, $S_{q-1} = (A_D, J_{i_q})$ is an optimal schedule for the set of jobs $\{J - J^*\} \cup \{J_{i_q}\}$, *etc.*).

Selection Algorithm

Given a set of jobs, $J = \{J_1 \cdots J_n\}$, with known processing times, $t_1 \cdots t_n$, and due-dates, $D_1 \cdots D_n$, the following algorithm will find a job $J_i \varepsilon J$ and $J_i \varepsilon L$ for some optimal schedule if such a job exists. Define the initial current sequence as $(J_1 \cdots J_n)$ assuming that $t_1 \leqq t_2 \leqq \cdots \leqq t_n$.

Start: Find the first late job, J_q, in the current sequence where $q > 1$. (If no such job exists, the current sequence has no late jobs and the algorithm terminates.) Re-order the previous $q - 1$ jobs according to the due-date rule to obtain a sequence of the form:

$$(J_{i_1} \cdots J_{i_{q-1}}, J_q \cdots J_n),$$

where

$$t_q \leqq \cdots \leqq t_n,$$

$$D_{i_1} \leqq \cdots \leqq D_{i_{q-1}},$$

and

$$t_q \geqq t_{i_j} \quad \text{for} \quad j = 1 \cdots q - 1.$$

Insert J_q into the sequence $(J_{i_1} \cdots J_{i_{q-1}})$ according to the due-date rule, obtaining the sequence:

$$(J_{i_1} \cdots J_{i_k}, J_q, J_{i_{k+1}} \cdots J_{i_{q-1}}, J_{q+1} \cdots J_n),$$

where

$$D_{i_1} \leqq \cdots \leqq D_{i_k} \leqq D_q \leqq D_{i_{k+1}} \leqq \cdots \leqq D_{i_{q-1}}.$$

There are 3 cases to consider:

1) If all jobs in the sub-sequence $(J_{i_1} \cdots J_{i_k}, J_q, J_{i_{k+1}} \cdots J_{i_{q-1}})$ are early, resume the analysis at start using the sequence just defined as the new current sequence.

2) If J_q is late in the above sequence, then $J_q \varepsilon L$ for some optimal schedule for the jobs in the current sequence.

Proof: Consider an optimal schedule of the form, $S_D = (A_D, P)$ for the set of jobs in the current sequence. If $J_q \varepsilon P$, we are done. Hence, assume $J_q \varepsilon A_D$ and that S_D is of the form:

$$S_D = (J_{l_1} \cdots J_{l_r} \cdots J_{l_t}, P),$$

and

$$D_{l_1} \leqq \cdots \leqq D_{l_r} < D_{l_{r+1}} \leqq \cdots \leqq D_{l_t},$$

where

$$J_{l_r} = J_q.$$

In the current sequence $t_q \leqq \cdots \leqq t_n$ and $t_{i_j} \leqq t_q$ for $j = 1 \cdots q-1$. Thus, the summation of the processing times for the jobs in the set $\{J_{i_1} \cdots J_{i_k}\}$ is minimal over the class of all subsets of k jobs from the current sequence having due-dates less than or equal to D_q. Hence, since J_q is late in the sequence $(J_{i_1} \cdots J_{i_k}, J_q)$, it will be late in all due-date ordered sequences in which it is the $k+1^{st}$ job. But $J_{l_r} = J_q$ is early in the schedule S_D and therefore r must be strictly less than $k+1$. Further

$$\{J_{l_{r+1}} \cdots J_{l_t}\} \cap \{J_{i_1} \cdots J_{i_k}\} = \phi,$$

since $D_{i_j} \leqq D_q$ for $j = 1 \cdots k$ and $D_q < D_{l_m}$ for $m = r+1 \cdots t$. Hence, for a member of $\{J_{i_1} \cdots J_{i_k}\}$ to be on time in the schedule S_D, It must be a member of $\{J_{l_1} \cdots J_{l_{r-1}}\}$. Consequently, there are jobs in the set $\{J_{i_1} \cdots J_{i_k}\}$ that are members of the set L for the schedule S_D since $r < k+1$. A new optimal schedule, S_D^*, can now be defined. Order the set of jobs, $\{J_{i_1} \cdots J_{i_k}\}$, according to the shortest processing time rule and select the first r jobs from this set. Replace the initial sequence, $(J_{l_1} \cdots J_{l_r})$, in the schedule S_D by this set of jobs ordered according to the due-date rule and call this schedule S_D^*. The job $J_{l_r} = J_q$ is now a member of L^* for the optimal schedule S_D^*.

3) If J_q is not late in the above sequence but at least one of the jobs in the sequence $(J_{i_{k+1}} \cdots J_{i_{q-1}})$ is late, then $J_q \varepsilon L$ for some optimal schedule for the set of jobs in the current sequence.

Proof: As in the proof for 2), assume we have an optimal schedule, S_D, of the form:

$$S_D = (J_{l_1} \cdots J_{l_r} \cdots J_{l_t}, P),$$

and

$$D_{l_1} \leqq \cdots \leqq D_{l_r} < D_{l_{r+1}} \leqq \cdots \leqq D_{l_t},$$

where

$$J_{l_r} = J_q.$$

a) If $r < k + 1$, the same optimal schedule, S_D^*, can be defined as in 2) where $J_q \varepsilon L^*$.

b) If $r \geqq k + 1$, the argument developed in the proof for 2) can be used to show that the completion time of $J_{l_r} = J_q$ in the schedule S_D is greater than or equal to its completion time in the sequence $(J_{i_1} \cdots J_{i_k}, J_q)$. Further, the jobs in the set $\{J_{i_{k+1}} \cdots J_{i_{q-1}}\}$ must follow J_q in the schedule S_D in that $(J_{l_1} \cdots J_{l_r} \cdots J_{l_t})$ is ordered by due-dates. Therefore, since at least one of the jobs in the set $\{J_{i_{k+1}} \cdots J_{i_{q-1}}\}$ was late in the original sequence, at least one of these jobs, say J_{i_m}, must belong to the set L for the schedule S_D.

But

$$t_{i_m} \leqq t_q,$$

and

$$D_{i_m} \geqq D_q.$$

Hence, J_q can be removed from A_D and replaced by J_{i_m} to form a new optimal schedule S_D^* where $J_q \varepsilon L^*$.

This process continues and each succeeding job is either accepted into the initial due-date ordered sub-sequence or is rejected. The algorithm terminates with a due-date ordered sequence of early jobs, $(J_{a_1} \cdots J_{a_t})$, and a set of jobs, $J_{a_{t+1}} \cdots J_{a_n}$ that have been rejected. An optimal schedule to the original problem will be

$$S_D = (J_{a_1} \cdots J_{a_t}, P),$$

where P is defined for the set of jobs, $\{J_{a_{t+1}} \cdots J_{a_n}\}$.

Minimizing the Maximum Deferral Cost[2]

Retaining the assumptions made in the introduction, a new problem can be formulated. Associated with each job is a continuous, bounded, monotonically non-decreasing function, p_i, called a deferral cost where $P_i(t)$ represents the cost of completing the job at time t. Sequencing problems with these cost structures have been considered by McNaughton (5) and Lawler (4). Their results, however, concern the problem of minimizing the sum of the deferral costs. The objective here is to find a schedule for which the maximum deferral cost incurred is minimal.

Definition: For each P_i, define a function P_i^* as follows:

1) $P_i^*(y) = P_i^{-1}(y)$ if $P_i^{-1}(y)$ (the inverse) exists

2) Otherwise

a) $P_i^*(y) = \max\{t \mid P_i(t) = y\}$, if $\exists\, t \varepsilon P_i(t) = y$,

b) $P_i^*(y) = 0$, if $\forall\, t, P_i(t) > y$,

c) $P_i^*(y) = +\infty$, if $\forall\, t, P_i(t) < y$.

Jackson's lemma can now be used to find an optimal schedule. For arbitrary $y > 0$, let $D_i = P_i^*(y)$ and let $S_D(y)$ be the schedule obtained by ordering the jobs according to the "due-dates" so defined. In that the P_i's are bounded, there

[2] This extension was suggested by Professor E. L. Lawler, Department of Electrical Engineering, The University of Michigan.

always exists $y > 0$ such that $S_D(y)$ has no "late" jobs.[3] Since the P_i's are monotonically non-decreasing, the P_i^*'s are monotonically non-decreasing and hence there exists $y^* > 0$ such that:

1) $S_D(y^*)$ has no "late" jobs.

2) For all $y < y^*$, $S_D(y)$ has at least one late job.

This value can be found to whatever accuracy is desired by a binary search technique. The schedule $S_D(y^*)$ has minimal maximum deferral cost.

Author's Supplement[4]

It has been brought to the attention of the author that the "compliment" of the presented algorithm should be computationally more efficient. This algorithm is given below:

Step 1: Order the set of jobs according to the due-date rule and call the resulting ordering $((J_{i_1} \cdots J_{i_n})$ where $D_{i_1} \leqq \cdots \leqq D_{i_n})$ the current sequence.

Step 2: Using the current sequence, find the first late job, J_{i_q}, and go to step 3. If no such job is found, the algorithm terminates with an optimal schedule obtained by placing those jobs that have been rejected after the jobs in the current sequence in any order.

Step 3: Find the job in ihe sub-sequences, $J_{i_1} \cdots J_{i_q}$, of the current sequence having the largest processing time and reject it from the current sequence. Now return to step 2, using the resulting sequence as the current sequence.

The principal advantage of this algorithm is that it eliminates a maximum of $n(n + 1)/2$ additions and comparisons needed in the first algorithm to check whether the jobs in the due-date reordered sub-sequences are all on time. The proof of its validity is of the same type given in the paper.

References

1. Conway, R. W., Maxwell, W. L., and Miller, L., "Theory of Scheduling," Addison-Wesley, 1967.
2. Eastman, Willard L., "Comments on a paper by Schild and Fredman," *Management Science*, Vol. 11, No. 5 (March 1965), pp. 754–755.
3. Held, Michael, and Karp, Richard M., "A Dynamic Programming Approach to Sequencing Problems," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 10, No. 1, (March 1962), pp. 196–210.
4. Jackson, J. R., "Scheduling a Production Line to Minimize Maximum Tardiness," Research Report 43, Managements Sciences Research Project, UCLA, January, 1955.
5. Lawler, Eugene L., "On Scheduling Problems with Deferral Costs," *Management Science*, Vol. 11, No. 2 (November 1964), pp. 280–288.
6. McNaughton, Robert, "Scheduling with Deadlines and Loss Functions," *Management Science*, Vol. 6, No. 1 (October 1959), pp. 1–12.
7. Root, James Gordon, "Scheduling with Deadlines and Loss Functions and the Optimal Selection of Resources to Perform Tasks," Ph.D. Thesis, Ohio State University, 1963.
8. Schild, Albert, and Fredman, Irwin J., "On Scheduling Tasks with Associated Linear Loss Functions," *Management Science*, Vol. 7, No. 3 (April 1961), pp. 280–285.
9. ——— and ———, "Scheduling Tasks with Deadlines and Non-Linear Loss Functions," *Management Science*, Vol. 9, No. 1 (October 1962), pp. 73–81.
10. Smith, Wayne E., "Various Optimizers for Single-Stage Production," *Naval Research Logistics Quarterly*, Vol. 3, Nos. 1 and 2 (March and June, 1956), pp. 59–66.

[3] Actually the bounded condition is stronger than necessary. It is sufficient if there exists $y > 0$ such that $S_D(y)$ has no "late" jobs.

[4] The algorithm given here was proposed by Mr. T. J. Hodgson, Department of Industrial Engineering, The University of Michigan.