ALGORITHMS FOR SOLVING PRODUCTION-SCHEDULING PROBLEMS

B. Giffler and G. L. Thompson*

International Business Machines Corporation, White Plains, New York

(Received October 15, 1959)

Algorithms are developed for solving problems to minimize the length of production schedules The algorithms generate any one, or all, schedule(s) of a particular subset of all possible schedules, called the *active schedules* This subset contains, in turn, a subset of the optimal schedules It is further shown that every optimal schedule is equivalent to an active optimal schedule Computational experience with the algorithms shows that it is practical, in problems of small size, to generate the complete set of all active schedules and to pick the optimal schedules directly from this set and, when this is not practical, to random sample from the set of all active schedules and, thus, to produce schedules that are optimal with a probability as close to unity as is desired The basic algorithm can also generate the particular schedules produced by well-known machine loading rules

PRODUCTION-scheduling problems considered in this paper occur in the following context †

1 There will be production demands for m commodities Each commodity must be processed by one or more of n facilities (We call the processing of a commodity by a facility an *operation* Operations are the elements, or 'things,' to be scheduled)

2. An operation, once started, may not be interrupted Subsets of facilities may be equivalent

3. Some pairs of operations *may* have to be performed in succession Some pairs of operations *must* be performed in succession

4 There is given for each operation a finite interval of time, namely, the time needed to perform the operation, and, for each pair of operations which must or may be performed in succession, a minimum delay after the start of the first operation before the second may be started (In some problems the minimum delay and the time to perform the first operation are equivalent)

5 We assume, for convenience only, that the operations to produce each commodity must be performed in a linear sequence and that no commodity is processed

[*] The second author is now at the Carnegie Institute of Technology, Pittsburgh, Pennsylvania His work on the final draft of this paper was supported, in part, by the Office of Naval Research through a grant to the Graduate School of Industrial Administration at the Carnegie Institute of Technology

[†] Other discussions for these kinds of problems have been given elsewhere *See*, for instance, references 1, 4, 5, and 6

twice by any facility or pair of equivalent facilities These are assumptions of convenience because the designation of commodities is arbitrary If necessary, each operation could be said to produce a different commodity, in which case compliance with the assumptions is automatic

6. In the first sections of this work we will sometimes assume, in order to simplify explanations, that each operation must be performed to completion before any operation that it precedes may be started and that the time to perform operations is independent of the order in which they are performed

To explain more exactly what is meant by performing operations 'in succession,' we define the order relation of *next-follows* '$<<$' In this definition we represent operations by an ordered pair of indices, for example, (c_i, f_j) This expression means 'the processing of commodity i by facility j' We define $(c_i, f_j) << (c_h, f_k)$ to mean that the processing of commodity i on facility j is next-followed by the processing of commodity h on facility k

The next-follows relation is irreflexive, intransitive, and antisymmetric It can be extended to a transitive relation which we call *follows* and designate by the symbol '$<$' We say that $(c_i, f_j) < (c_h, f_k)$ if, and only if, there is at least one chain of relations '$<<$' from (c_i, f_j) to (c_h, f_k) The follows relation may be further extended to a reflexive relation, *see* reference 2

An operation may next-follow two or more other operations For example, if (c_i, f_j) is not the first operation to produce c_i, it may next-follow another operation on c_i and another operation performed by f_j If (c_i, f_j) is the first operation to produce c_i, it may next-follow any number of last operations to produce other commodities Thus, one should note, it is not generally correct to assume that an operation can start whenever an operation which it next-follows is completed

With each operation (c_i, f_j) we associate a time t_{ij} This is the actual time it takes to perform the operation We assume the time is known exactly and in advance The $m \times n$ matrix $\boldsymbol{T} = \| t_{ij} \|$ is called the *time matrix* of the scheduling problem

Having defined the basic order relations in schedules, we are now able to define more exactly what is a feasible schedule, what is an optimal schedule, and the approach we intend to use in determining optimal schedules

DEFINITION By a feasible schedule we shall mean an assignment of an order of processing commodities on each facility such that

(a) The assignment will be consistent, that is, the number of operations in each chain of next-follows relations will be finite (This is the same as to say that the schedule can be completed in a finite time)

(b) The assignment will respect all given next-follows relations, the given times to perform operations, and the minimum allowable delays between the starts of successive operations (This group of requirements is commonly referred to as the *technological requirements*)

DEFINITION Consider a feasible schedule and all pairs of operations that are connected by the follows relation For each such pair of operations define the *schedule interval* to be equal to the interval of time between the start of the followed operation and the completion of the following operation Schedule intervals will be made up, in part, of the times to perform operations and, in part, of idle times of facilities and (or) commodities

DEFINITION By an optimal schedule we shall mean a feasible schedule in which the maximum schedule interval between any two operations is minimum *

If there is a single feasible schedule, there must obviously be an optimal schedule This optimal schedule may be found by enumerating and evaluating all possible assignments of commodities to facilities However, such a method of solution may require an examination of $(m!)^n$ different schedules and is, for this reason, computationally impractical—except possibly for problems with very few commodities and facilities

The approach used to solve scheduling problems in this paper is twofold First we show that it is not necessary to search for an optimal schedule over all possible schedules, but only over a subset of the feasible schedules, called the active *schedules* This subset is shown in a later section to contain, in turn, a subset of optimal schedules It is further shown that every optimal schedule is equivalent to an active optimal schedule Second, we develop useful concepts about scheduling processes (a few of these have already been developed) that enable us to reduce the computational effort necessary to search the subset of active schedules

It turns out to be practical, in problems of small size, to examine all active schedules and to pick the optimal schedules directly from this set However, even when it is not practical to examine all active schedules, it is possible, with a reasonable effort, to random sample from this set and, in this way, to produce schedules that are optimal with a probability as close to unity as is desired The computational methods, to be developed, may also be used to generate the schedules that are produced by any of the commonly used machine-loading rules

There is some recent work in which scheduling problems, similar to those considered in this paper, are formulated as integer linear programs We note, however, that this approach has not yet led to computationally practical methods of solution, *see* reference 7

* A different, but equivalent, definition of an optimal schedule is given in the section "Active and Optimal Schedules "

OPEN, DUAL-OPEN, NUMERICAL, AND NONNUMERICAL PROBLEMS

IT IS DESIRABLE to define three particular scheduling problems for closer examination For this purpose it is convenient to use an expanded notation for indicating the order in which the commodities are processed by the facilities and the order in which facilities accept the commodities

We let the ith *facility sequence*, F_i, for each $i=1, \quad , m$, be the ordered set of the indices of the successive facilities that process c_i Similarly, we let the jth *commodity sequence*, C_j, for each $j=1, \quad , n$, be the ordered set of the successive commodities that are processed by f_j In general the commodity and facility sequences will be given only in part by the technological requirements and the scheduling task will be to complete them

We define the *facility sequence matrix*

$$\boldsymbol{F}= \begin{array}{c} c_1 \\ c_2 \\ \\ c_m \end{array} \left\| \begin{array}{c} F_1 \\ F_2 \\ \\ F_m \end{array} \right\| \tag{1}$$

to be the ordered collection of the facility sequences One notes that $\boldsymbol{F}$ will be a rectangular matrix only if every commodity is processed by the same number of facilities

We similarly define the *commodity sequence matrix*

$$\boldsymbol{C}= \begin{array}{c} f_1 \\ f_2 \\ \\ f_n \end{array} \left\| \begin{array}{c} C_1 \\ C_2 \\ \\ C_n \end{array} \right\| \tag{2}$$

The sequence matrix $\boldsymbol{C}$, like $\boldsymbol{F}$, will be a rectangular matrix only if every facility processes the same number of commodities

Finally, we define the square $m\times m$ *structure* matrix, $\boldsymbol{S}$ The (i, j)th element of $\boldsymbol{S}$ is unity if the last operation to produce c_i is next-followed by the first operation to produce c_j and is zero, otherwise It is clearly necessary, if there is to be at least one feasible schedule, that $\boldsymbol{S}$ be a strict triangular matrix or orthogonally equivalent to a strict triangular matrix

In typical problems only parts of $\boldsymbol{F}$ and $\boldsymbol{C}$ will be given and it will be necessary to complete these matrices without violating the transitive and anti-symmetric properties of *follows* These matrices, when completed subject to $\boldsymbol{S}$, contain all the information necessary to construct a feasible schedule In fact, all that remains is to note where operations and facilities must be idle in order to realize the orderings in the $\boldsymbol{F}$ and $\boldsymbol{C}$

Facility and commodity sequence matrices, when augmented by having idle times inserted at the proper places, will be called *Gantt charts* and indicated by $\boldsymbol{G}(\boldsymbol{F})$ and $\boldsymbol{G}(\boldsymbol{C})$, respectively

EXAMPLE Suppose $m=3$ and $n=2$ and the facility sequence matrix is

$$F=\begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array}\left\|\begin{array}{cc} 1 & 2 \\ 1 & 2 \\ 2 & 1 \end{array}\right\|$$

This facility sequence matrix indicates that commodity 1 and commodity 2 are processed first on facility 1 and then on 2, and that commodity 3 is processed first on facility 2 and then on 1

A possible commodity sequence matrix is

$$C=\begin{array}{c} f_1 \\ f_2 \end{array}\left\|\begin{array}{ccc} 2 & 1 & 3 \\ 3 & 1 & 2 \end{array}\right\|$$

This sequence matrix indicates that facility 1 processes the commodities in the order 2, 1, and 3, and that facility 2 processes them in the order 3, 1, and 2 **C** is said to be possible in that its orderings, taken together with the orderings in **F**, can be used to construct feasible schedules (i e , Gantt charts) as follows

$$G(F)=\left\|\begin{array}{cccc} - & 1 & 2 & - \\ 1 & - & - & 2 \\ 2 & - & 1 & - \end{array}\right\|,$$

$$G(C)=\left\|\begin{array}{cccc} 2 & 1 & 3 & - \\ 3 & - & 1 & 2 \end{array}\right\|$$

Here the dashes indicate idle times for either the facilities or the commodities The structure matrix, in the above example, would serve (if it were not null) to place additional restrictions on **F** and **C** in that no operation (c_i, f_j) could be allowed to be performed before any operation (c_h, f_k) if $s_{ih}=1$

The Gantt charts constructed above are said to be nonnumeric because they do not take account of the actual time intervals necessary to perform each operation, or equivalently, because they assume $t_{ij}=1$ for all i and j where t_{ij} is the time to perform operation (c_i, f_j)

Gantt charts become numeric when they take account of the actual time to perform operations and of the actual duration of idle times To incorporate this numeric aspect into the chart we assume that the times are each integral multiples of some basic unit of time and repeat each index or dash in **G(F)** and **G(C)** a number of times equal to this multiple For instance, if we assume in the above example that each $t_{ij}=i+j$, we should convert the preceding nonnumeric Gantt charts to the numeric Gantt charts,

$$G(F)=\left\|\begin{array}{cccccccccccc} - & - & - & 1 & 1 & 2 & 2 & 2 & - & - & - & - \\ 1 & 1 & 1 & - & - & - & - & - & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & - & - & - \end{array}\right\|,$$

$$G(C)=\left\|\begin{array}{cccccccccccc} 2 & 2 & 2 & 1 & 1 & 3 & 3 & 3 & 3 & - & - & - \\ 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{array}\right\|$$

One should note, at this point, that **G(F)** and **G(C)** will always have the same number of columns if they are obtained from the same **F** and **C** However, they will have the same number of rows only if the number of commodities should happen to equal the number of facilities, that is if $m=n$

The number of columns in $G(F)$ and $G(C)$ is the length of time to complete all operations in the schedule and is called the *length* of the Gantt chart This number is also equivalent to the maximum schedule interval that is to be minimized A proof of the equivalence is given further on

We now define three scheduling problems that are to be considered in detail Each is a variant of the general scheduling problem described in the introduction

The open scheduling problem In the open scheduling problem the facility sequence and structure matrices F and S are given and it is required to find the Gantt charts of an optimal schedule In other words, the commodity sequence matrix C plus necessary idle times must be determined for an optimal schedule

The dual-open scheduling problem In the dual-open problem the commodity sequence and structure matrices C and S are given and it is required to find a facility sequence matrix F and the Gantt charts of an optimal schedule

The mixed scheduling problem In the general case, part of the F sequence matrix and part of the C sequence matrix are given and it is required to find the rest of each matrix, subject to S, plus the Gantt charts of an optimal schedule

ACTIVE AND OPTIMAL SCHEDULES

FOR THIS section and most of the next, we shall concentrate on the open scheduling problem and return to the other kinds later In order to cut down the number of schedules that have to be considered, we restrict our attention to active schedules, the characterization of which is given in the next three definitions

DEFINITION Consider the Gantt chart $G(C)$ of a feasible schedule Suppose that in the ith row (corresponding to the ith facility) of $G(C)$ the operation (c_i, f_j) is assigned beginning at time t and ending at time $t+q$ Consider the altered chart $G'(C)$ obtained from $G(C)$ by removing this operation and assigning f_j to be idle from time t to $t+q$ We now want to reassign the operation (c_i, f_j) so that it begins at time u and ends at time $u+q$ We shall say that it is *permissible* to shift the operation (c_i, f_j) from time t to time u if and only if (a) f_j is idle from time u to $u+q$, and (b) the index i (and the index h if S_{ih} or $S_{hi} \neq 0$) does not occur in the kth column of $G'(C)$ for k ranging in the interval $\mathbf{min}(t, u) \leqq k \leqq \mathbf{max}(t+q, u+q)$ Moreover, if $t<u$ we call the shift a *permissible right shift*, and if $t>u$, we call it a *permissible left shift*

Note that the relation of permissible shifting is reflexive and symmetric We extend it to a transitive relation by means of the next definition

DEFINITION Two schedules are said to be *equivalent* if and only if one can be obtained from the other by a sequence of permissible shifts

The equivalence relation divides feasible schedules into disjoint equivalence classes

DEFINITION An *active schedule* is a feasible schedule having the property that no operation can be made to start sooner by permissible left shifting

THEOREM 1 *Every feasible schedule $G(C)$ is equivalent to an active feasible schedule whose maximum schedule interval is at most that of $G(C)$*

Proof If $G(C)$ is already active it is then equivalent to itself in the vacuous sense Consider the Gantt chart $G(C)$ of an inactive feasible schedule For each facility that is idle at the first time period determine the operations on that facility, if any, that could be left-shifted to fill the idle time of the first period (These will be operations on commodities that could have started on the facility without being preceded by other operations) If there is one or more such operation, select one and left-shift it to start to time 0 Next, in the revised Gantt chart, consider all facilities that are idle starting at time 1 If there are subsequent operations on such facilities that can be left-shifted to start at time 1, select one and left-shift it, etc By a finite number of such left shifts the inactive schedule is changed into an active one Moreover, since only left-shifting is involved, each schedule interval between jobs will be, at most, decreased so that the maximum schedule interval of the new schedule will be at most as long as that of $G(C)$

It is easy to construct examples whereby the same feasible schedule is equivalent to two or more active schedules Moreover, if both left and right shifts are permitted, it is possible for two active schedules to be equivalent We shall not study the equivalence relation further here

DEFINITION By an *active chain* in a schedule we shall mean a chain of next-follows relations

$$(c_{i_1}, f_{j_1}) << (c_{i_2}, f_{j_2}) << \qquad << (c_{i_k}, f_{j_k})$$

having the property that the schedule performs the operations in the chain without intervening idle times Thus the length of time to perform all the operations in such a chain is equal to the sum of the minimum allowable delays between the starts of the first $k - 1$ operations in the chain plus the time to perform (c_{i_k}, f_{j_k}) to completion

THEOREM 2 *An active schedule possesses at least one active chain with length equal to the maximum schedule interval of the schedule and this is, in turn, the length of the schedule's Gantt chart*

Proof Consider any one of the last operations to be performed in an active schedule If this operation is not also the first operation of the facility, it next-follows one or more other operations One notes that, if it did not, the facility on which it is processed would have been idle for at least one time period during which the commodity is also idle so that the operation we are considering could have been done earlier, contradicting the fact that it was a last operation and our schedule was active Again, the last operation we are considering must next-follow *without idle time* one of the operations it next-follows If it did not, it could have been processed earlier and the schedule would not be active

We move now from the last operation, just considered, to an operation it next-follows without idle time and repeat the above argument Working our way back through the schedule (using mathematical induction) we construct an active chain, and arrive finally at an operation that does not next-follow any other We must now prove that this operation is the first on the facility on which it occurs and, thus, that the maximum schedule interval equals the length of the active chain The proof is that, if the operation in question is not a first operation, then the facility must be idle until this operation is performed But since the operation involved does not next-follow any operation, it could be performed earlier and would be, since the schedule is active Consequently, we conclude that the operation must actually be the first operation on the facility and that it is the first operation in an active chain That this chain is the maximum schedule interval and also the length of the Gantt chart follows automatically

DEFINITION An *optimal schedule* is a feasible schedule whose Gantt chart has length at least as short as the length of any other feasible schedule

In the introduction we gave another definition of an optimal schedule The next theorem shows the equivalence of these two definitions

THEOREM 3 *A feasible schedule having minimum Gantt chart length also is one that minimizes its maximum schedule interval, and conversely*

Proof Obviously the maximum schedule interval is at most the Gantt chart length, so that we have only to prove the statement the other way, namely, that the maximum schedule interval is at least the Gantt chart length By Theorem 1, a feasible schedule $\boldsymbol{G(C)}$ is equivalent to an active feasible schedule $\boldsymbol{G'(C)}$, whose length is at most that of the feasible schdule Thus, if the feasible schedule has minimum length, its equivalent schedule also has minimum length Next, by Theorem 2, the schedule $\boldsymbol{G'(C)}$ contains an active chain whose length equals the length of $\boldsymbol{G(C)}$ and $\boldsymbol{G'(C)}$ Since, in this chain, no left shifts are permitted, the same chain exists in $\boldsymbol{G(C)}$ Hence the maximum schedule interval is at least as long as (and is, therefore, equal to) the Gantt chart length of $\boldsymbol{G(C)}$

THE ALGORITHM FOR GENERATING ALL ACTIVE SCHEDULES

WE DEVELOP in this section an algorithm for generating all active schedules We concentrate our attention on the active schedules because (1) they contain (by Theorem 1) a subset of schedules to which all optimal schedules must be equivalent and (2) they are superior, as a class, to the inactive schedules in that one or more of their operations must start earlier than in any inactive schedule to which they are equivalent Also, the set of active schedules is frequently observed to be much smaller than the set of all feasible schedules *

Let n_i be the number of facilities that process the ith commodity, let m_j be the number of commodities processed by a jth facility, assume that there is exactly one of each jth facility Thus $n_i \leqq n$, $m_j \leqq m$ We consider the open problem in which we are given the facility sequence matrix F, and the times, t_{ij}, to perform each operation, (c_i, f_j), to completion We assume for the present, that $S=0$

We construct, in Fig 1, a linear array having $E=\sum_{i=1}^{i=n} n_i$ entries, that is, as many cells as there are operations to be scheduled We use the shortened notation, $c_j(i)$, to represent the ith operation on f_j It may be helpful to consider the successive cells as successive memory locations in a computer

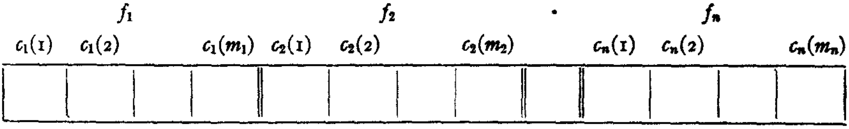

Figure 1

The successive cells in Fig 1 may be regarded as a set of boxes, indexed by facility-commodity pairs Since m_j adjacent boxes represent operations to be performed by a jth facility, it is appropriate to refer to this block of boxes as the jth *facility block* The algorithm, to be developed, will insert in each box, the completion time of the operation that it represents Since the times to perform each operation are given, it becomes a simple matter to deduce the start time of any operation

If the technological requirements are that no facility may be assigned to start an operation until the last started operation has been performed to completion and an assignment is made that does not respect this requirement or the possibility of such an assignment exists, we say there is a *conflict* The implication is that, whenever there is a set of operations in

* An empirical measure of this difference is given in reference 3

conflict at a facility, assigning one of the set to start must automatically delay the start of the others The concept is defined more exactly below *

DEFINITION Suppose that, on a jth facility, operations $c_j(1)$, $c_j(2)$, , $c_j(p)$ have processing times t_{1j}, t_{2j}, , t_{pj} and are assigned to complete at times s_1, s_2, · , s_p Then there is a conflict in this assignment if for some k and h with $k<h\leqq p$, we have $s_h-s_k<t_{hj}$

The interpretation of a conflict is clear For instance, if an operation is assigned to be completed at time 18 and takes 7 units of time, then the completion times of all earlier operations performed by the same facility must occur at time 11 or earlier To put it differently, if all earlier operations are not completed by time 11, then the given operation cannot be completed by time 18

We specify now the algorithm for generating all active schedules We assume that the facility sequence matrix $\boldsymbol{F}$ and all operation times, t_{ij}, are given We assume, for the present, that each operation must be performed to completion before any operation it precedes can start, that operation times are independent of the order in which the operations are performed, and that there is exactly one facility of each type We begin with a linear array as in Fig 1, and perform the following steps

1. Enter the completion times of the first operations to produce each commodity as given by $\boldsymbol{F}$

2. Set T equal to the smallest completion time so entered (*Note* All completion times greater than or equal to T are tentative and may be changed as the algorithm proceeds)

3. In each facility block having one or more operations finishing at time T, check for conflicts between those operations and operations completing at later times The *conflict set* of such a facility block will consist of (a) all operations ending at time T, and (b) all operations that overlap the operations found in (a)

4. In a facility block in which there are r ($\geqq 1$) operations in the conflict set determined in step **3**, choose one of the operations Left-shift this operation, if possible Suppose its completion time is T' Replace the completion times of each operation in the conflict set by its operation time plus the larger of T' or the arrival time of the job at the facility Leave the completion times of jobs not in the conflict set unchanged

5. For each new operation assigned in step **4**, look in the $\boldsymbol{F}$ matrix to see the next facility (if any) that will process the commodity of that operation Enter its completion time plus the new operation time in the corresponding box of the next operation If a new operation has been assigned completion time $T'<T$, go back to step **3.** If all new operations have been assigned completion times $T'\geqq T$, go on to step **6.**

* The concept of a conflict is defined here on the assumption that successive operations performed by a facility do not overlap The concept, however, is easily extended to cover that case where there are permissible overlaps, *see* part (d) at the end of this section

6. If T is not the largest entry in the array, find the integer T' in the array that is next larger than T and set $T=T'$, then go back to step 2.

7. If T is the largest entry in the array, stop The completion time of the resulting schedule will be this T

The preceding algorithm generates a particular active schedule To generate all active schedules it is necessary in step 4 to resolve conflicts in all possible ways, for an example see the next section If a random selection of an active schedule is desired, it is only necessary to select operations in step 4 by some acceptable random process

The algorithm, as stated above does not make explicit provision for cases where the structure matrix $S\neq 0$, where there are equivalent facilities, where processing times of operations are dependent on the order in which they are performed, and where operations are allowed to overlap The modifications to the algorithm necessary to account for these cases are given below (following Theorem 4) We separate these cases in order to have a simple first statement of the basic algorithm

THEOREM 4 *The above algorithm will generate all active schedules in the open scheduling problem.*

Proof The result of the algorithm is to produce a feasible schedule since (a) the technological requirements are satisfied because they are used as relevant in the algorithm, and (b) the schedule is feasible since all conflicts are resolved by the algorithm Finally, the resulting schedule is active since at a time when a facility can process completely one or more commodities, some assignment is always made to it Therefore, no further left shifts of operations are possible

On the other hand, it can be shown by induction that any active schedule can be generated by the algorithm To see this, let $G(C)$ be the Gantt chart of an active schedule The initial assignments must be of operations that are not preceded by other operations Thus they can be chosen, using the algorithm, in the same manner as they are in $G(C)$ Assume that the chart $G(C)$ is capable of being generated by the algorithm out to time t. Now if there are no new operations assigned at this time in $G(C)$, then the algorithm cannot choose any either, since $G(C)$ is active But if one or more new operations are assigned in $G(C)$, then it is possible to make the same assignment using the algorithm in exactly the same way Thus we conclude that $G(C)$ could have been constructed by the algorithm and that the algorithm, if worked out in all possible ways, will generate the complete set of active schedules

The algorithm for the dual-open problem is obtained from the above by first considering a linear array with $D=\sum_{i=1}^{i=m} n_i$ entries, grouped in blocks, called *commodity blocks* Then interchange the words commodity and facility in the statement of the algorithm

The algorithm can also be extended to solve mixed scheduling problems In such a problem we are given parts of the F and C matrices and are to fill in the remaining parts The problem can be reduced to a partially solved open problem by simply filling in the remaining entries of the F matrix and then using the algorithm to complete the schedule taking into account the partial C matrix, or, equally, well, the remainder of the C matrix could be filled in and the corresponding partially-solved dual-open problem could be completed For each way of filling in the remainder of these matrices, different optimal schedules result, possibly with different completion times By systematically going through all ways of filling in these partial matrices, the shortest possible schedule will be obtained Monte Carlo variations of these methods are also possible

We consider now cases in which the structure matrix is nonnull, there are equivalent facilities, operation times depend on the order in which the operations are performed, and operations are allowed to overlap

(*a*) *The structure matrix is nonnull* If $S \neq 0$, it is necessary to modify steps **1** and **5** In step **1**, to locate the first operation on facility j, suppose it is c_i, it is necessary both that c_i be the first entry in the jth row of F and that the ith column of S be null In step **5**, suppose c_i is the last entry in the jth row of F, to find the next-following operation look to see if there is exactly one nonzero entry in the ith row of S, if there is, and it is in the column of (say) c_h, then we would have to take as the next-following operation, the operation defined by the first entry in the row of c_h in F By means of the following device we can always ensure that there is at most one nonzero entry in each row of S The device is employed because it simplifies the clerical effort to determine the operation that can start next after each started operation In all cases there will be, at most, one such operation We have to resort to the device only if (c_i, f_j) is a last operation to produce c_i and it would, if the device were not employed, be next-followed by first operations on two or more other commodities

The device requires the creation of dummy operations that require zero time for their performance Suppose, for instance, that a is a last operation on a commodity and that it is next-followed by the k first operations on other commodities, b_1, b_2, , b_k In this case we create $k-1$ dummy operations, c_1, c_2, , c_{k-1}, and have $a \prec\prec b_1 \prec\prec c_1 \prec\prec b_2 \prec\prec c_2 \prec\prec$ $\prec\prec b_{k-1} \prec\prec c_{k-1} \prec\prec b_k$ Each dummy operation, c_h, would be 'performed' by the same facility that performs b_h Since each dummy operation requires zero time for its performance, it cannot interfere with the performance of nondummy operations

(*b*) *Equivalent facilities* Suppose that there are k machines available with the characteristics of f_j Then in step **4** we can permit conflict sets to be as large as k without harm But if the conflict set has more than k elements then it is necessary to choose a subset of k commodities to have their completion times assigned—suppose T' is the minimum assigned completion time Then we replace the completion times of each other operation in the conflict set by the larger of the arrival time of the job and T' plus its operation time. In the Gantt chart we provide k rows for the facility f_j and utilize them to list operations that are carried out simultaneously

(c) *Processing times of operations depend on the order of performing the operations* It is possible in some problems that the processing times of operations will depend on the order in which they are performed For example a time to perform an operation may often be decomposed into a time to set-up a facility to perform the operation and a time to perform the operation, given the set-up In these cases it can happen that the time to set-up an operation will depend on the status of the facility after the performance of the operation just previously completed by the facility The algorithm can handle the above situation with little difficulty For example in step 5, we need only observe which operation is being next-followed and to use (in determining tentative completion time) that value of t which applies when the next-following operation follows the particular observed next-followed operation In these cases, of course, we need to be given more data on the time to perform operations Specifically, we would need to know not just one time for each operation, $c_j(i)$, but a time contingent on the selection of $c_j(i)$ to next-follow each of the other operations that may be performed by f_j Thus if f_j may perform m_j operations, we would need to know, not m_j processing times, but $(m_j)(m_j-1)$.

(d) *Successive operations are allowed to overlap* In some scheduling problems it is possible for an operation to start before an operation which it next-follows is fully completed To accommodate the algorithm to this class of problems it is necessary only to consider the processing time of each operation to be actually a 'minimum delay' after the operation starts before its next-following operation can start The case where a facility starts and (or) completes two or more operations simultaneously may be handled as an 'extreme case' of overlap The concept of a conflict, when there are permissible overlaps, is expanded to rule out the possibility of starting two successive operations on a facility in less than the allowable minimum delay The algorithm, when there are overlaps, determines for each operation the earliest time the facility (which performs the operation) could start a next operation However, since we know the minimum delays for all pairs of operations that must or may be performed in succession on a facility and the times to perform each operation to completion, it is an almost trivial task to determine any operation's actual start and (or) completion time

COMPUTATIONAL EXPERIENCE AN EXAMPLE

We have had a fair amount of computational experience with the complete enumeration version of the algorithm and much more experience with Monte Carlo versions This will be more fully reported in reference 3 It is possible to report that a problem to select 200 random schedules with $m\times n=2000$ operations takes about 20 minutes with an IBM 704 While we have at the moment no exact measure of the degree of optimality of the best schedule found in this way, we can show that it is better than humans can do—if, in fact, humans consider problems of this size Generally speaking, the time to select 200 random schedules increases linearly with the total number of operations to be scheduled

We illustrate the basic algorithm by working the following sample problem requiring the processing of three commodities on each of three facilities Assume that

$$F=\left\|\begin{matrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{matrix}\right\|$$

and that $t_{ij}=i+j$ The linear arrays constructed by the algorithm are shown in Fig 2 Those marked with an X involve conflicts and are incomplete Those marked with an L followed by a number are completed schedules, the number giving the length of the schedule Dewey-decimal numbers indicate the connections between various schedules

The shortest schedules found had length 16, the longest had length 32 The Gantt chart, $G(C)$, of optimal schedule 1111 is

$$\left\|\begin{matrix} 1 & 1 & - & - & 2 & 2 & 2 & - & - & - & - & - & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 1 & 1 & 1 & 3 & 3 & 3 & 3 & 3 & - & - & - & - \\ 3 & 3 & 3 & 3 & 3 & 3 & - & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{matrix}\right\|$$

and has length 16 Note that by right shifts one can derive a number (in fact, 29) of equivalent inactive schedules, each having length 16

REMARKS AND EXTENSIONS

There are many ways in which the scheduling problem, and correspondingly, the algorithm for its solution can be extended We discuss some of them here We do not necessarily claim priority in the statement of the extension, but merely its application to our algorithm

Routing Problems

We have assumed that technological requirements on the order of processing each commodity on facilities, and each facility on commodities, are confined to stating given next-follows relations between operations In more general circumstances, however, it is possible that the requirements will define only partial orderings on the operations In these cases it is necessary to determine the linear orders and, if an optimal schedule is to be produced, to determine them in such a way that the final schedule is as short as possible We call this more general problem, the *routing problem*

Routing problems occur frequently in the design of assembly lines They can be solved by working out all possible ways of defining a linear order and, for each, a suboptimal schedule The optimum schedule then becomes the shortest of the suboptimal schedules An algorithm to determine all active solutions to the routing problem, or to a random sample from the set of all active solutions, need not be very different from the algorithms already described

	f_1			f_2			f_3			
	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3	
1	2			5	4				6	X
11	2	7		7	4	11			6	X
12	2			5	9		10		6	X
111	2	7		7	4	12	11	12	6	X
112	2	7	15	14	4	11	18	12	6	L18
121	2			5	9	11	10		6	X
122	2	12		5	9		9	17	15	X
1111	2	7	16	7	4	12	11	16	6	L16
1112	2	7	16	7	4	12	16	12	6	L16
1211	2	12	18	5	9	14	10	17	6	L18
1212	2	18	14	5	15	11	10	23	6	L23
1221	2	12	24	5	9	20	9	20	15	L24
1222	2	12	32	5	9	28	9	17	23	L32

Figure 2

Priorities

It is possible to associate with each commodity or operation a priority index and to use these indices to resolve conflicts We can, by this device, give preference, in resolving conflicts, to the operation that would be completed first and so on Many bases for constructing priority indices are easily incorporated in the Monte Carlo algorithms For instance, instead of resolving conflicts by a random selection of a waiting operation, we can observe (or determine) the priority index for each waiting operation and make the selection on the basis of their respective priorities Thus if we were to give priority to the operation that would be completed first, we would resolve all conflicts by selecting the waiting operation whose calculated tentative completion time is least

Random and priority selections may be used jointly to resolve conflicts For instance, if a priority index does not produce a unique resolution to a conflict, one can select a resolution at random (perhaps using the priority indices as weights) from the two or more possible resolutions that are produced Also there may be a choice of priority indices and one can select at random the particular index to use

Completion-Date Problems

Scheduling problems in which there are 'due dates' for one or more commodities presents no new conceptual difficulties What can be done is to work out all active schedules, as already described, and to discard those in which the due dates are not met One might sensibly set-back schedule the given due dates (*see* reference 2) to produce a derived due date for each operation One could then discard an active schedule being constructed by the algorithm on the first instance that an operation's due date is not met

Alternative Criteria for Optimality

We have assumed only one criterion for optimality, namely, that the schedule should be as short as possible (that is, the number of columns in the solution schedule's Gantt chart must be minimum) However, nothing has been derived that is based on the use of this particular criterion Thus we could require of an optimum schedule that it minimize the total idle time of all facilities and/or all commodities or any one of many possible criteria

REFERENCES

1 S B AKERS, JR AND J FRIEDMAN, "A Non-Numerical Approach to Production Scheduling Problems," *Opns Res* **3**, 429–442 (1955)

2 B GIFFLER, "Mathematical Solution of Production Planning and Scheduling Problems," Doct Dissert , Columbia University, School of Engineering, 1960

3 ———, G L THOMPSON, AND V VAN NESS, "Numerical Experience with the Linear and Monte Carlo Algorithms for Solving Production Scheduling Problems," forthcoming

4 JAMES R JACKSON, "Notes on Some Scheduling-Problems," Management Sciences Res Project, Research Report No 35, UCLA, Oct 1, 1954

5 S M JOHNSON, "Optimal Two- and Three-Stage Production Schedules with Setup Times Included," *Naval Res Log Quart* **1,** 61–68 (1954)

6 ALAN J ROWE AND J R JACKSON, "Research Problems in Production Routing and Scheduling," *J Ind Eng* **VII,** 116–121 (1956)

7 G L THOMPSON, "Recent Advances on the Production Scheduling Problem," forthcoming