A permutation procedure for job-shop scheduling

By T. A. J. Nicholson and R. D. Pullen*

A new method is proposed for scheduling jobs through a factory. The jobs consist of a sequence of operations, each operation requiring a number of resources of different types. The objective is to plan the start times of the operations so as to minimise the cost of jobs being late, subject to the sequence constraints being satisfied and the demand for resources not exceeding the supply. The problem is formulated in terms of optimising a permutation, and conditions for a locally optimal permutation are defined. A procedure is described for obtaining such locally optimal permutations and subsequent results show significant improvements over heuristic techniques such as the shortest operation and least slack rules.

(First received July 1967)

1. Introduction

Scheduling methods are concerned with progressing the forward work load in a factory so as to complete the jobs by their due dates and use the resources as efficiently as possible. Usually the jobs consist of a sequence of operations which have to be performed in turn, and the problem is to determine the start times of the operations so as to achieve the chosen objective. The difficulties arise because of the queues of operations which build up on particular resources, and a scheduling method is a means of controlling these queues by deciding which operations should be delayed. In small workshops it may be possible to schedule efficiently without any formal schemes, but in larger concerns the volume and variety of the workload generally means that it is essential to use systematic methods for the control to be effective. The implementation of a scheduling system will often be supported by a computer program.

Past research on scheduling problems falls into two categories, and these have been well reviewed (Sisson, 1962; Mellor, 1966). Firstly, there are methods for finding optimum solutions to very simple problems. The three-machine scheduling problem is the classic case. The problems have had to be simplified to keep the formulation mathematically tractable and to apply an optimisation method. The methods include integer-linear programming (Bowman, 1959; Manne, 1960; Wagner, 1959), and branch and bound techniques (Ignall *et al.*, 1965). The weakness of these methods is that they cannot be applied to problems of more than a very few variables, nor do they appear to suggest suitable lines of advance for practical situations.

Secondly, there is a group of heuristic studies which have aimed at finding rules for scheduling efficiently in special practical situations (Bulkin *et al.*, 1966; Gere, 1966; Conway and Maxwell, 1961; Rowe, 1960). The criteria are usually chosen intuitively. Although these methods are usually computationally quick, they provide no indication of how close to the optimum the answers lie, and they tend to be very special purpose.

This paper describes a scheduling method which tries to preserve the best features of both categories. A realistic factory model is formulated and a formal optimisation method is devised for the associated scheduling problem. The factory model is based on the central features experienced in many practical workshops, and the objective commonly occurs in practice.

2. The factory model and notation

The model has the following structure of jobs, operations, and resources.

(i) *The jobs*

There is a known forward load of work consisting of a number of separate jobs which require processing on the available resources. Each job has a given arrival time in the factory and a due time by which the job must be completed, to avoid incurring penalty costs. Jobs are also given a value and a priority relative to other jobs. The cost of a job being late depends on the value of the job, its priority, and the extent to which it is late.

(ii) *The operations*

Each job consists of a number of separate operations which must be performed in a specified sequence. The sequence may vary with different jobs. The durations of all the operations are known in advance, and are independent of one another. Each operation requires a constant number of resources from a specified variety of resource types while it is being processed. Once an operation has been started it cannot be interrupted by any other operation.

(iii) *The resources*

The resources are classified into homogeneous types, and the supply levels of the various types are known and may vary over time.

(iv) *The objective*

The objective is to plan the start times of the operations so as to minimise the cost attributable to jobs exceeding their due dates.

Some common practical features of workshops such as transit and set-up times or limitations on in-process

* *Mathematics Branch, A.E.R.E., Harwell, Didcot, Berkshire.*

stocks are not explicitly included in the model, but it is often possible to reinterpret these aspects within the above framework.

The following notation is used in the formulation of the problem:

N is the number of jobs
M is the number of resource types
$A(j)$ is the arrival time of job j
$T(j)$ is the due time of job j
$v(j)$ is the value of job j
$u(j)$ is the priority of job j
$n(j)$ is the number of operations of job j
$d(j, k)$ is the duration of the kth operation of job j
$r(j, k, m)$ is the number of resources of type m required by operation k of job j at every time interval in which the operation is being performed
$s(l, m)$ is the supply of resource type m at time l
$x(j, k)$ is the scheduled start time of the kth operation of job j.

All quantities are assumed to be positive integers or zero.

3. The formal optimisation problem

The problem variables are the start times $x(j, k)$ of the individual operations. We will denote by the vector $\{X\}$ an assignment of values to the problem variables. All other quantities are given data for the particular problem.

The objective function is defined in terms of the cost of jobs being late. A job is late if it is completed after its due time, and we assume that the cost steadily increases the later the job is completed. We therefore define by $g(j, t(j))$ the cost of job j being completed at time $t(j)$, and the function g will have the properties:

$$g(j, t(j)) = 0 \quad \text{for} \quad t(j) \leqslant T(j) \tag{1}$$

and

$$\frac{\partial g}{\partial t(j)} \geqslant 0. \tag{2}$$

Also the cost may depend on the value and priority of the job, and in particular we will study the objective function for which the cost is the product of the job's value and its lateness taken to the power of its priority. Therefore

$$g(j,t(j)) = v(j).\ \text{Max}\,(x(j, n(j)) + d(j, n(j)) - T(j), 0)^{u(j)}. \tag{3}$$

The objective function then becomes:

$$F\{X\} = \sum_{j=1}^{N} g(j, x(j, n(j)) + d(j, n(j))). \tag{4}$$

The constraints on the values of $x(j, k)$ are expressed as follows. Firstly, a job cannot start before it arrives in the factory. Therefore we require

$$x(j, 1) \geqslant A(j). \tag{5}$$

Secondly, the operations must be processed in the correct sequence, i.e.

$$x(j, k) \geqslant x(j, k-1) + d(j, k-1) \text{ for } k = 2, n(j). \tag{6}$$

Thirdly the demand for resources must not exceed the supply at any time. Therefore

$$\sum_{j=1}^{N} \sum_{k=1}^{n(j)} \delta(j, k, l).r(j, k, m) \leqslant s(l, m) \quad \text{for all } l \text{ and } 1 \leqslant m \leqslant M \tag{7}$$

where

$$\delta(j, k, l) = 1 \quad \text{for} \quad x(j, k) \leqslant l < x(j, k) + d(j, k)$$
$$= 0 \text{ otherwise.} \tag{8}$$

Therefore the optimisation problem is to determine the values $x(j, k)$ so as to minimise the objective function (4) subject to the constraints (5), (6) and (7).

It is not possible to apply standard optimisation methods to this scheduling problem. A total search of the feasible schedules would be a massive undertaking (Giffler *et al.*, 1960; Heller *et al.*, 1960). Branch and bound methods would be equally impractical (Ignall *et al.*, 1965). Also, although in certain circumstances the problem could be translated into the framework of integer-linear programming the computational task would still be impossibly large.

In the following sections we propose a new technique for tackling this problem, which is an extension of a method which has been used successfully on a number of smaller problems of a similar type (Nicholson, 1967). The basic scheme is to convert the optimisation problem into a permutation problem in which a number of items have to be arranged in a sequence of positions. A definite level of optimality is set for the permutation solution and a method is described for constructing such a permutation.

4. The scheduling problem as a permutation problem

The scheduling problem is translated into a permutation problem in the following way. First, the total set of operations is arranged as an order list. Let the number pair $(j(i), k(i))$ denote the operation in position i of the list, being the $k(i)$th operation of job $j(i)$. Then the total set of operations can be represented by a permutation $[P]$ where

$$[P] = [(j(1), k(1)), (j(2), k(2)), (j(T), k(T))] \tag{9}$$

where $T = \sum_{j=1}^{N} n(j)$.

Secondly we need a means of transforming the permutation $[P]$ into a set of operation start times $x(j(i), k(i))$. We therefore construct the following transformation. Determine $x(j(i), k(i))$ for $i = 1$ to T in that order as

$$x(j(i), k(i)) = Z(j(i), k(i)) \tag{10}$$

where $Z(j(i), k(i))$ is the smallest integer which satisfies

the following inequalities:

$$Z(j(k), 1) \geqslant A(j(i)) \tag{11}$$

$$Z(j(i), k(i)) \geqslant X(j(i), k(i) - 1) + d(j(i), k(i) - 1) \quad \text{for } k(i) = 2, n(j(i)) \tag{12}$$

$$\sum_{L=1}^{i} \delta(j(L), k(L), l).r(j(L), k(L), m) \leqslant s(l, m) \text{ for all } m, \tag{13}$$

for $x(j(i), k(i)) \leqslant l < x(j(i), k(i)) + d(j(i), k(i))$.

The transformation represented by (10), (11), (12) and (13) may be denoted by α so that the relationship may be written concisely as

$$\{X\} = \alpha[P] \tag{14}$$

where $\{X\}$ denotes the vector of values $x(j(i), k(i))$ for $i = 1, T$).

In order that this transformation will provide a feasible schedule we need to impose a feasibility condition on the permutation. A feasible permutation is defined as any permutation $[P]$ such that $l > i$,

$$k(i) < k(l) \quad \text{if} \quad j(i) = j(l). \tag{15}$$

It should be noted that this is the only constraint on the permutation. All the other conditions for feasibility have been automatically built into the transformation α.

It remains to show that the optimum to the original optimisation problem is equivalent to the optimum of the corresponding permutation problem with the given transformation α. The proof of this equivalence is established in an Appendix.

5. Neighbouring exchange optimal permutations

Formally there exists one or more optimal schedules amongst the $T!$ possible permutations. But as in practice T may be of the order of 10^2 or 10^3, it is impossible to conduct a total evaluation of all possibilities. We will therefore define characteristics of permutations which are locally optimal, and in particular we will define permutations which are optimal with respect to the exchange of any two neighbouring operations.

Firstly we define the *neighbours* of an operation in a permutation. Any operation in the permutation has two neighbours a *predecessor* and a *successor* neighbour. Given any operation $(j(i_1), k(i_1))$, in position i_1, the operation $(j(i_2), k(i_2))$ is a predecessor neighbour if it is the operation nearest in the permutation which uses some of the same resources as operation $(j(i_1)k(i_1))$. Therefore $(j(i_2), k(i_2))$ is a predecessor neighbour of $(j(i_1), k(i_1))$ if $i_2 < i_1$,

$$\sum_{m=1}^{M} r(j(i_1), k(i_1), m).r(j(i_2), k(i_2), m) > 0 \tag{16}$$

and

$$\sum_{m=1}^{M} r(j(i_1), k(i_1), m).r(j(l), k(l), m) = 0 \quad \text{for} \quad i_2 < l < i_1. \tag{17}$$

Similarly we define the operation $(j(i_2), k(i_2))$ as a successor neighbour if, $i_2 > i_1$, condition (16) holds, and condition (17) holds for $i_1 < l < i_2$.

We can now define the neighbouring exchange of an operation. Let $(j(i), k(i))$ be any operation and let $(j(l), k(l))$ be a neighbour of operation $(j(i), k(i))$. Then we define a neighbouring exchange of operation $(j(i),k(i))$ by exchanging the permutation $[P]$ into the permutation $[P(i, l)]$ where

$$\begin{aligned}[P(i, l)] = [&(j(1), k(1)), (j(2), k(2)), \ldots, \\ &(j(l-1), k(l-1)), (j(i), k(i)), (j(l), k(l)), \ldots, \\ &(j(i-1), k(i-1)), (j(i+1), k(i+1)), \ldots, \\ &(j(T), k(T))] \quad \text{if} \quad l < i\end{aligned} \tag{18}$$

or

$$\begin{aligned}[P(i, l)] = [&(j(1)), k(1)), (j(2), k(2)), \ldots, \\ &(j(i-1), k(i-1)), (j(i+1), k(i+1)), \ldots, \\ &(j(l), k(l)), (j(i), k(i)), (j(l+1), k(l+1)) \ldots, \\ &(j(T), k(T))] \quad \text{if} \quad l > i.\end{aligned} \tag{19}$$

It should be noted that the permutation $[P(i, l)]$ is feasible only if

$$\left.\begin{aligned} j(L) \neq j(i) \quad &\text{for} \quad L = l, i, \quad \text{if} \quad l < i \\ &\text{for} \quad L = i, l, \quad \text{if} \quad l > i. \end{aligned}\right\} \tag{20}$$

Therefore we define the exchange $[P]$ to $[P(i, l)]$ to be feasible if condition (20) holds.

Finally we define a permutation which is optimal with respect to neighbouring exchanges. The permutation $[P]$ is optimal with respect to all neighbouring exchanges if for any operation in position i and either of its neighbours in position l the condition

$$F[P] \leqslant F[P(i, l)] \tag{21}$$

holds for all i and l, the permutation $[P(i, l)]$ being feasible, and $F[P]$ being the objective function value associated with the permutation $[P]$.

In the following sections we describe a method for constructing permutations to satisfy condition (21). Section 8 describes the construction of an initial permutation and Section 9 describes how to modify this permutation to the required level of optimality. Section 10 describes a fast method for modifying the initial permutation so that the final permutation 'nearly' satisfies condition (21).

6. The construction of an initial permutation

We now describe how to construct an initial feasible permutation. Clearly any feasible permutation would suffice as an initial permutation, as it could subsequently be modified until it was optimal with respect to neighbouring exchanges. But it is important to aim for an initial permutation which is as near optimal as possible so that the number of subsequent exchanges is as small

as possible. Equally, the calculation of the initial permutation should not be too expensive in computing time. We therefore propose the following procedure. The permutation is built up one operation at a time, the tth operation to be selected being placed in position t. We restrict the choice of operations which can be selected at the tth iteration so as to ensure feasibility and reduce computation. Also we estimate statistically the increase in the objective function value which will result from the selection of an operation, and choose that operation for which the estimated increase in the objective function value is a minimum.

This procedure may be formalised as follows. Let J_t denote the set of operations available for selection for the tth position in the permutation. Also let $f_t(j, k, t)$ denote the estimated increase in the objective function value which would result from selecting operation (j, k) for position t. Then if (j_t, k_t) denotes the selected operation, (j_t, k_t) is determined for $t = 1$ to T by the equation:

$$f_t(j_t, k_t, t) = \underset{(j,k)\in J_t}{\text{Min}}\, f_t(j, k, t). \tag{22}$$

The set of operations J_t consists of a restricted group of the next operations of all jobs which are not completed. An operation is included in J_t only if its earliest possible start time is not greater than the latest finish time of the operations which have already been scheduled. Furthermore the operation selected by equation (22) is scheduled only on condition that no other operation in the J_t set can be completed before the selected operation will be started. If this is possible then the J_t set is revised to include only those operations which can be completed before the selected operation could start. In this way the J_t set is determined in an iterative manner. This scheme for determining J_t with relatively few operations which can be started in the neighbourhood of the scheduled operations is aimed at reducing computation and encouraging high resource utilisation.

It remains to determine a suitable expression for $f_t(j, k, t)$. The function $f_t(j, k, t)$ measures the increase in the value of the objective function which would result from selecting operation (j, k) for position t. Clearly this increase can only be estimated, as the permutation is incomplete, and we will derive a convenient formula for this estimate.

Firstly it is necessary to estimate the completion time of job j after $t-1$ operations have been scheduled. Denote this estimate by $T_{t-1}(j)$, and let $n_{t-1}(j)$ and $Z(j, n_{t-1}(j))$ be the number of the next operation of job j and the earliest time at which it could be started. The value of $T_{t-1}(j)$ is based on the earliest possible start time of the next operation of job j, together with the processing time of the remaining operations and the delays which these operations expect to experience. Let μ_t denote the expected delay on an operation after t operations have been scheduled. Then $T_{t-1}(j)$ may be estimated as

$$T_{t-1}(j) = Z(j, n_{t-1}(j)) + \sum_{k=n_{t-1}(j)}^{n(j)} (\mu_t + d(j,k)) \tag{23}$$

where μ_t is calculated adaptively as:

$$\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}\Big(x(j_t, n_{t-1}(j_t)) - x(j_t, n_{t-1}(j_t) - 1) - d(j_t, n_{t-1}(j_t) - 1))\Big). \tag{24}$$

We can now estimate the increase in the objective function value which would result from selecting operation (j, k) at selection t. The significance of selecting operation (j, k) is that it may delay the progress of other jobs. We can estimate the rate of increase in the value of the objective function if job i is delayed beyond its currently estimated completion time $T_{t-1}(i)$ as $\left[\frac{\partial g}{\partial t(j)}\right]_{t(j)=T_{t-1}(j)}$ where the function g is defined by expression (3). Since the jobs which are likely to be delayed directly by selecting the next operation of job j will all have their next operations included in the J_t set, the total rate increase in the value of the objective function due to selecting operation $(j, n_{t-1}(j))$ is estimated as

$$\sum_{\substack{(i,n_{t-1}(i))\in J_t \\ i\neq j}} \left[\frac{\partial g}{\partial t(i)}\right]_{t(i)=T_{t-1}(i)}. \tag{25}$$

We will use this rate of increase as a measure of the increase in the value of the objective function.

This formula for $f_t(j, n_{t-1}(j), t)$ may be simplified. The operation on $f_t(j, k, t)$ in equation (22) is a minimisation, and a minimisation is unaffected by subtracting a constant from all members of the set to be minimised. Since $\sum_{(i,n_{t-1}(i))\in J_t} \left[\frac{\partial g}{\partial t(j)}\right]_{t(j)=T_{t-1}(j)}$ is a constant independent of $(j, n_{t-1}(i))$, it can be subtracted from expression (25) to obtain the estimate of $f_t(j, n_{t-1}(j), t)$ as

$$f_t(j, n_{t-1}(j), t) = -\left[\frac{\partial g}{\partial t(j)}\right]_{t(j)=T_{t-1}(j)}. \tag{26}$$

For the particular objective function of expression (3) and (4) we will obtain the value of $f_t(j, n_{t-1}(j), t)$ as

$$f_t(j, n_{t-1}(j), t) = -\nu(j)u(j)\max(T_{t-1}(j) - T(j), 0)^{u(j)-1}. \tag{27}$$

If all the operations in the set J_t are such that $T_{t-1}(j) < T(j)$, then the function $f_t(j, n_{t-1}(j), t)$ will be zero, and the selection procedure of (22) will be indeterminate. This corresponds to a factory situation in which there is more than adequate time to complete the jobs by their due times and it does not matter which operation is selected next. When this situation arises, the convention will be adopted of choosing the next operation as the operation which is nearest to becoming late, i.e. the operation for which $(T(j) - T_{t-1}(j))$ is a minimum. This completes the definition of the function $f_t(j, k, t)$.

When all the operations have been included in the permutation, the initial permutation is complete.

7. Obtaining a locally optimal permutation

We now need to modify the initial permutation so as to obtain a permutation which is optimal with respect to neighbouring exchanges as defined in Section 6. To do this we make a series of neighbouring exchanges at each iteration reducing the value of the objective function.

We determine the sequence of exchanges as follows. We select the job which currently contributes most to the objective function and which has not been selected since the last iteration on which the objective function was reduced. If at the tth iteration this job is denoted by j_t and the set of job numbers which have been selected since the last iteration on which the objective function was reduced is U_{t-1}, then j_t is determined by the equation

$$g(j_t, T_{t-1}(j_t)) = \max_{\substack{1 \leqslant n \leqslant N \\ jU_{t-1}}} g(j, T_{t-1}(j)) \qquad (28)$$

where $T_{t-1}(j)$ is the completion time of job j with the current permutation.

We now make a series of neighbouring exchanges with the operations of job j_t starting with the last operation and proceeding to the first operation and repeating the cycle until a full cycle has been made with no improvement in the objective function. If this occurs on the first cycle then the set U_t contains the elements of U_{t-1} and the number j_t. Otherwise the set U_t contains only the job number j_t.

When the set U_t contains all the job numbers $j = 1$ to N then a permutation has been obtained which is optimal with respect to neighbouring exchanges.

8. Nearly optimal permutations

The procedure described in the previous section for modifying the initial permutation to obtain a neighbouring-optimal permutation may be too costly in computing time. In this case we can significantly reduce the amount of computation by considering a job only once for trial exchanges of its operations. The procedure is identical to that described in Section 7 except that the set U_t is always set to the elements of U_{t-1} and the job number j_t. This will mean that we lose any guarantee that an optimal permutation will be obtained, but as the results will show, good answers may be obtained which may be satisfactory for practical purposes.

9. Results

The permutation method was programmed in FORTRAN for the IBM 7030 computer, and it was tested against some well known scheduling rules which have been used for machine loading problems and resource allocation associated with critical path planning procedures. The rules are:

(i) First-come-first-served.
(ii) The shortest operation discipline.
(iii) The least slack rule. The slack associated with any operation is determined dynamically as the difference between the latest time at which the operation may be started, to avoid the job being late, and the earliest time in view of the operations already scheduled. The operation with the least slack is chosen.
(iv) The least slack per operation. In this case the operation slack is divided by the number of remaining operations of that job.
(v) Random selection.

All these methods provide a rule for choosing which operation to schedule next when a queue of jobs is waiting for resources. The queues are determined by simulating the scheduling process over time.

The data for the test cases was generated statistically according to the following scheme:

N, the number of jobs varied between 19 and 32.
M, the number of resource types was 10.
$A(j)$, the jobs arrived randomly in the interval (0, 60), the average inter-arrival time being 2·5.
$T(j)$, the job due time was determined as:

$$T(j) = A(j) + (1 + D) \sum_{j=1}^{n(j)} d(j, k),$$

where D was uniformly distributed in the interval (0·2, 1·2).
$v(j)$ the value of job j was uniformly distributed in the interval (1, V) where V is as shown in the tables of results.
$u(j)$ the priority of job j was uniformly distributed in the interval (U_1, U_2) as shown in the tables of results.
$n(j)$ the number of operations of a job was uniformly distributed in the range (3, 10).
$d(j, k)$ the operation duration is negative exponentially distributed with mean value 3.
$r(j, k, m)$ the operation resource requirements: the number of resource types required by an operation is uniformly distributed in the range (1, R_1), the type numbers being different random integers in the range (1, m). The number of units of each type of resource required by an operation is uniformly distributed in the range (1, R_2). R_1 and R_2 are as shown in the tables.
$s(l, m)$ the supply level of resources was assumed to be constant over time and equal for all resources. The level S is as shown in the tables.

Tables 1, 2, 3 and **4** show the results of 5 cases in each of four different scheduling situations in which the operations have varying kinds of demands for resources. These tables show the objective function values obtained for the initial permutation, the nearly optimal (see Section 8) and the neighbouring optimal permutation. The objective function values of the schedules obtained by the other scheduling rules are also shown. The values are scaled as shown.

Table 1

Objective function values $\times 10^{-k}$

Data parameters: $V = 5$, $U_1 = 2$, $U_2 = 3$, $R_1 = 1$, $R_2 = 1$, $S = 1$

CASE (k)	INITIAL PERMUTATION	NEARLY OPTIMAL	NEIGHBOURING OPTIMAL	FIRST COME FIRST SERVED	SHORTEST OPERATION	LEAST SLACK	LEAST SLACK PER OPERATION	RANDOM
1 (2)	44	1	0	538	35	36	13	1462
2 (3)	4	1	1	406	300	372	314	1055
3 (3)	9	1	1	119	29	93	141	1030
4 (2)	14	3	3	1012	386	262	45	1770
5 (3)	68	37	31	228	227	106	129	1022

Table 2

Data parameters: $V = 5$, $U_1 = 2$, $U_2 = 3$, $R_1 = 1$, $R_2 = 3$, $S = 3$

1 (3)	1	1	1	5	108	85	85	53
2 (2)	1	1	1	392	79	133	44	64
3 (2)	0	0	0	81	25	25	24	91
4 (3)	22	5	5	36	8	10	10	137
5 (3)	9	4	3	82	705	316	84	399

Table 3

Data parameters: $V = 5$, $U_1 = 2$, $U_2 = 3$, $R_1 = 3$, $R_2 = 1$, $S = 3$

1 (4)	11	8	7	131	66	51	62	252
2 (2)	26	19	8	37	70	10	5	392
3 (0)	0	0	0	4	6	1	2	29
4 (2)	0	0	0	591	30	37	202	445
5 (0)	0	0	0	513	53	133	53	1493

Table 4

Data parameters: $V = 5$, $U_1 = 2$, $U_2 = 3$, $R_1 = 3$, $R_2 = 3$, $S = 4$

1 (3)	105	5	5	4377	183	488	345	6009
2 (4)	10	6	6	149	102	13	21	215
3 (3)	33	8	8	79	7	69	100	31
4 (4)	97	83	73	725	301	440	375	1694
5 (5)	23	20	17	140	91	85	61	129

Table 5

Average job lateness

Data parameters: $V = 1$, $U_1 = 1$, $U_2 = 1$, $R_1 = 3$, $R_2 = 3$, $S = 4$

CASE	INITIAL PERMUTATION	NEARLY OPTIMAL	NEIGHBOURING OPTIMAL	FIRST COME FIRST SERVED	SHORTEST OPERATION	LEAST SLACK	LEAST SLACK PER OPERATION	RANDOM
1	8·7	3·0	3·0	10·0	7·0	8·1	8·6	22·4
2	9·2	4·4	4·5	14·1	9·2	9·1	10·5	16·1
3	13·6	3·9	3·4	21·8	11·8	13·4	11·2	39·2
4	24·2	9·4	9·4	39·8	27·0	29·5	32·0	40·2
5	5·1	0	0	6·6	4·4	5·6	6·4	8·5

Table 6

Maximum job lateness

Data parameters: $V = 1$, $U_1 = 4$, $U_2 = 4$, $R_1 = 3$, $R_2 = 3$, $S = 4$

1	0	0	0	62	34	12	14	21
2	2	2	2	2	4	4	4	4
3	0	0	0	15	12	8	8	8
4	9	0	0	33	40	59	59	36
5	0	0	0	10	11	10	10	11

In **Table 5** we study the objective of minimising average job lateness by setting all job values and priorities to unity. In **Table 6** we study an objective in which heavy penalties are placed on jobs being late by setting the job priorities to 4. This table records maximum job lateness in the 5 cases.

In **Table 7** we record the computing time on the IBM 7030 required by the various techniques.

Table 7

Computing times in seconds

NUMBER OF OPERATIONS	INITIAL PERMUTATION	NEARLY OPTIMAL	NEIGHBOURING OPTIMAL	OTHER RULES
50	0·4	10	21	0·29
100	1·3	26	40	1·0
150	2·7	80	180	2·6
200	4·9	260	800	4·2
400	10·3	—	—	7·9
600	35	—	—	26

10. Conclusions

From the above tables of results we can draw some clear conclusions about the permutation procedure both as regards the quality of optimisation obtained and the computation involved.

Firstly the permutation procedure offers significant improvement over the heuristic scheduling methods with which it has been compared. This is true for both the neighbouring optimal permutations and the nearly optimal. Furthermore the results reached by the permutation procedures are consistently good whereas the best heuristic technique varies from case to case. Gere (1966) also found the same inconsistency in the heuristic rules he studied. Part of the weakness of the one-pass heuristic scheduling methods seems to lie in their failure to bring predecessor operations forward in time close to the final operation of the job, thereby freeing resources for other final operations which may thus be finished earlier. The neighbouring exchanges achieve this adjustment, and as a by-product this will lead to reductions in work-in-progress.

Secondly the permutation procedure is very flexible and could be used for a wide range of scheduling problems. Tables 5 and 6 have shown how the procedure can deal with the objectives of minimising average job lateness or minimising maximum lateness. But with an adjusted formulation the same type of procedure could be applied either to other objectives such as maximising resource utilisation or to different problem structures, for example where set-ups are important. The scope of a permutation procedure is independent of particular mathematical forms for the objective functions and constraints, and it is therefore of wide application.

Thirdly, the computational time required to obtain a neighbouring optimal permutation is quite small for

small problems but rapidly increases. However, the nearly optimal method offers answers which are nearly as good for a significant reduction in computing time. Also, the initial permutation as derived for this particular problem offers consistently better results than the heuristic methods for a very small increase in computing time. For larger problems the best policy may be to partition the schedule into segments and apply the neighbouring exchange procedure to each segment in turn, or else to terminate the exchanges automatically when a given amount of computing time has been expended.

The permutation procedure therefore promises considerable returns over some of the established job-shop scheduling techniques. The method is being used as the basis for some scheduling studies currently being undertaken in the rubber industry in co-operation with the Rubber and Plastics Research Association.

Appendix

In this appendix we establish the result which was assumed in Section 6, namely that the original optimisation problem formulated in Section 5 can be solved optimally as a permutation problem using the transformation proposed in Section 6. The same notation will be used. We establish the result by proving two theorems and a lemma.

Theorem 1

Given any optimal feasible solution $\{X'\}$, and any positive integer q, there exists another solution $\{X^*\}$ satisfying the following three conditions:

$$\{X^*\} \text{ is feasible} \tag{A1}$$

$$\{x^*(1, 1), x^*(1, 2), \ldots, x^*(j, k) - q, \ldots x^*(N, n(N))\}$$

$$\text{is not feasible for any individual pair } (j, k) \tag{A2}$$

$$\text{and } F\{X^*\} \text{ is a minimum.} \tag{A3}$$

Proof

We assume that an optimal feasible solution exists and denote it by $\{X'\}$. We now note that inequalities (5) and (6) impose a lower bound on the $x(j, k)$ variables. Therefore if each of the variables $x(j, k)$ is taken in turn, as many times as necessary, their values may be reduced by unit steps from $x'(j, k)$ to new values $x^*(j, k)$, maintaining feasibility throughout so that the set $\{X^*\}$ satisfies conditions (A1) and (A2).

Also, as the functions $g(j, t(j))$ satisfy conditions (1) and (2), and $x^*(j, n(j)) \leqslant x'(j, n(j))$, $F\{X^*\} \leqslant F\{X'\}$. But as $\{X'\}$ is an optimal solution, $\{X^*\}$ is also an optimal solution satisfying condition (A3). This proves Theorem 1.

Lemma

Referring to condition (A2), we now establish that the loss of feasibility resulting from reducing the variable $x^*(j', k')$ to $x^*(j', k') - q$ in the solution $\{X^*\}$ is independent of the values of the variables $x(j, k)$ for which

$$x^*(j, k) \geqslant x^*(j', k'). \tag{A4}$$

Proof

When the value of $x(j, k)$ is reduced below $x^*(j, k)$, feasibility is lost through violating one or more of the constraints (5), (6) or (7). Firstly, suppose $k' = 1$ and constraint (5) is violated. Since $A(j)$ is a constant this violation is independent of the values of variables $x(j, k)$ satisfying (A4). Secondly, suppose $k' > 1$, and constraint (6) is violated. This violation depends on the value of $x(j', k' - 1)$. But since $\{X^*\}$ is a feasible solution $x^*(j', k' - 1) < x^*(j', k')$, and again this violation is independent of the values of variables satisfying (A4).

Thirdly, suppose constraint (7) is violated by reducing $x(j', k')$ to the value $x^*(j', k') - q$. This implies that the inequality

$$\sum_{\substack{j=1 \\ (j,k) \neq (j',k')}}^{N} \sum_{k=1}^{n(j)} \delta(j, k, l).r(j, k, m) + r(j', k', m) > S(l, m) \tag{A5}$$

holds for at least one value of l in the interval

$$x^*(j', k') - q \leqslant l < x^*(j', k') + d(j', k') - q.$$

But since $\{X^*\}$ is feasible we have

$$\sum_{\substack{j=1 \\ (j,k) \neq (j',k')}}^{N} \sum_{k=1}^{n(j)} \delta(j, k, l).r(j, k, m) + r(j', k', m) \leqslant s(l, m)$$

in the interval $x^*(j', k') \leqslant l < x^*(j', k') + d(j', k')$. Therefore the inequality (A5) holds only for values of $l < x^*(j', k')$. But for any value of $l < x^*(j', k')$, $\delta(j, k, l) = 0$, for all variables $x(j, k)$ whose values are greater than $x^*(j', k')$. Therefore the violation of the constraint (7) is independent of the values of the variables satisfying (A4). This establishes the Lemma.

Theorem 2

Given the optimum solution $\{X^*\}$ defined by (A1), (A2) and (A3), there exists a permutation $[P^*]$ such that

$$\{X^*\} = \alpha[P^*]. \tag{A6}$$

Proof

First we construct the following permutation $[P^*]$. Given the solution $\{X^*\}$ we can determine an ordering of the number pairs (j, k) as $(j(i), k(i))$ for $i = 1, T$, such that

$$x^*(j(i), k(i)) < x^*(j(i + 1), k(i + 1)).$$

We can thus use these ordered number pairs to define the permutation $[P^*]$ as

$$[P^*] = [(j(1), k(1)), (j(2), k(2)), \ldots (j(T), k(T))].$$

Secondly we need to show that the transformation α of (10) to (13) when applied to the permutation $[P^*]$ generates the solution $\{X^*\}$. We prove this by induction. Suppose we have generated the first $(l-1)$ values by the transformation α as $x^*(j(i), k(i))$ for $i = 1, l-1$. The transformation now determines the value of $x(j(l), k(l))$ as the first feasible value. We know by condition (A1) that the value $x^*(j(l), k(l)) - q$ is not feasible for any positive integer q. Also by the lemma these results are independent of the value of variables $x(j(i), k(i))$ for $i > l$. Therefore the value of $x(j(l),k(l))$ will be determined as $x^*(j(l), k(l))$.

But as $\{X^*\}$ is feasible, $k(1) = 1$ and by condition (A2), $x^*(j(1), k(1)) = A(j(1))$. Also the transformation α will determine the value of $x(j(1), k(1))$ as $x^*(j(1), k(1))$. Hence by induction

$$\{X^*\} = \alpha[P^*]$$

and theorem 2 is proved.

We have therefore shown that the optimum to the permutation problem is also the optimum to the original optimisation problem expressed in the variables $x(j, k)$.

References

BULKIN, M. H., COLLEY, J. C., STEINHOFF, H. W. (1966). Load forecasting, priority sequencing and simulation in a job shop control system, *Man. Sci.*, Vol. 13, No. 2, p. 29.

BOWMAN, E. H. (1959). The schedule sequence problem, *Opr. Res.*, Vol. 7, p. 621.

CONWAY, R. W., and MAXWELL, W. L. (1961). Network despatching by the shortest operation discipline, *Opr. Res.*, Vol. 10, p. 51.

COX, D. R. (1961). *Queues*, Methuan Monograph.

GERE, W. S. (1966). Heuristics in job shop scheduling, *Man. Sci.*, Vol. 13, No. 3, p. 167.

GIFFLER, B., and THOMPSON, G. L. (1960). Algorithms for solving production scheduling problems, *Opr. Res.*, Vol. 8, p. 487.

HELLER, J., and LOGEMAN, G. (1960). An algorithm for constructing feasible schedules and computing their schedule times, *Man. Sci.*, Vol. 8, p. 168.

KELLEY, J. E. (1961). Critical path planning and scheduling: mathematical basis, *Opr. Res.*, Vol. 9, p. 296.

MANNE, A. S. (1960). On the job shop scheduling problem, *Opr. Res.*, Vol. 8, p. 219.

MELLOR, P. (1966). Job shop scheduling—a review, *Opr. Res., Quat.* June 1966.

NICHOLSON, T. A. J. (1967). A sequential method for discrete optimisation problems and its application to the Assignment, Travelling Salesman and three-machine scheduling problems, (to be published in *Journal of Institute of Mathematics and its Applications*).

ROWE, A. J. (1960). Toward a Theory of scheduling, *Journal of Industrial Engineering*, March 1960.

SISSON, J. (1960). Sequencing Theory, Chapter 7. *Progress in Operation Research*, Ed. R. L. Ackoff, New York: Wiley.

WAGNER, H. M. (1959). An integer linear-programming model for machine shop scheduling, *Naval Logistics Quarterly*, Vol. 6, p. 131.

WOODGATE, H. S. (1966). Some recent developments in network based resource allocation techniques, Proceedings of 1966 British Joint Computer Conference, p. 95.

Book Review

Operations Research in Seller's Competition: A Stochastic Microtheory, by S. SANKAR SENGUPTA, July 1967; 228 pp. (New York and London: *John Wiley & Sons, Inc.*, 80s.)

The publishers' description of the book says 'Given the mode of competition and the existing organisation of the market: how do the actions of individuals bring about specific patterns of relationships among the observable market variables? What are the specific measures of uncertainty which would seem to characterise an individual firm's outcome? How can an understanding of these aspects of selling lead to correct decisions about: pricing, advertising and selling costs, the production inventory complex, and capital investment? These are the problems and questions which Professor Sengupta considers in this book.' The author brings to bear a powerful battery of statistical techniques to study these problems. As an exercise in applied statistics the book is a *tour de force* and is a valuable contribution to the stochastic theory of micro-economics.

It is, however, published as Volume 13 of the series of publications in Operations Research, sponsored by the Operations Research Society of America. It is doubtful if many practising Operational Research scientists will make much of the book because the theory out-runs the practice.

ANDREW YOUNG (Liverpool)