DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Some pages in the original document contain color pictures or graphics that will not scan or reproduce well.
SCHEDULING OF FLEXIBLE FLOW SHOPS

by

K.L. Hitz*

This research was partially supported by National Science Foundation Grants NSF/RANN APR76-12036 and NSF DAR78-17826, "Systems Aspects of Flexible Manufacturing Systems".

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the National Science Foundation.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

*On leave from Dept. of Mechanical Engineering, University of Newcastle, NSW, Australia.
ABSTRACT

The report discusses the scheduling of production in a deterministic Flexible Flow Shop. This is a serial arrangement of buffered multipurpose machines connected by a conveyor system which allows fully mechanized parts handling. Machines can be bypassed, and simultaneous processing of parts with different machine routes is possible. The problem considered is that of producing a range of part types in specified ratios, with known and fixed transport and processing times. Optimal schedules are characterized as periodic loading sequences which minimize idle time on bottleneck machines once they have begun operating, while satisfying constraints on buffer capacity. The computation of optimal schedules by an implicit enumeration algorithm is discussed. Early computational results indicate that this formulation of the scheduling problem leads to significant savings when compared to the minimization of makespan or mean weighted finish time customary in the scheduling literature.
ACKNOWLEDGEMENTS

I am grateful to my own University, and to Prof. M. Athans, Director of the Laboratory for Information and Decision Systems at M.I.T., for the opportunity of a sabbatical leave at MIT.

The research reported here is one of the results. It has been helped along by many stimulating discussions with the members of the Manufacturing Group at LIDS. Special thanks are due to Prof. Athans, Dr. S.B. Gershwin, Mr. J. Ward and M. J. Kimemia. I am also indebted to Mr. R. Hildebrand of the C.S. Draper Laboratories, Cambridge, Mass., who independently, and some time before me, arrived at the notion of a minimal part set.
CONTENTS

ABSTRACT i

ACKNOWLEDGEMENTS ii

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 MAXIMAL STEADY-STATE PRODUCTION IN A FLEXIBLE FLOW SHOP 2

CHAPTER 3 COMPUTING OPTIMAL FFS SCHEDULES 15

CHAPTER 4 EXTENSIONS AND CONCLUSIONS 23

4.1 Closed-Loop Control of a Flexible Flow Shop 23

4.2 Fast Heuristics for Near-Optimal FFS Schedules 25

4.3 Conclusion 27

APPENDIX FORTRAN PROGRAM FOR OPTIMAL FFS SCHEDULES 29

A1: List of Parameters and Variables 30

A2: Program Listing 33

A3: Sample Problems and Printouts of Solutions 52

REFERENCES 71

LIST OF FIGURES

Fig. 1 A 4-Machine Flexible Flowshop 2

Fig. 2 Unfinished Work in a Single Machine and Its Buffer with a Periodic Sequence of Arriving Parts 9

Fig. 3 Example of FFS Scheduling Problem 13
1. INTRODUCTION

Flexible manufacturing systems [1] are networks of multipurpose, automatic, often computer-controlled machines linked by an automated transport system. Such systems are rapidly gaining interest in manufacturing industry. They offer one way of improving productivity in those areas of manufacturing where each plant typically produces a range of similar items in volumes too low to warrant the use of product-specific mass-production machinery such as transfer lines.

The work described in this report deals with some aspects of the problem of scheduling production in a simple type of flexible manufacturing system that might be called a Flexible Flow Shop (FFS). This is a serial arrangement of multipurpose machines connected by a fixed conveyor system. The conveyors are equipped with appropriate sensors and switching devices so that the transport of parts from machine to machine is fully mechanized. The first and last machines in the FFS are loading and unloading stations where parts are loaded onto, or removed from, fixtures or pallets necessary for alignment in the machines or on the conveyor. Each internal machine of the system is assumed to be capable of performing a range of similar operations, or sequences of operations. Moreover, it will be assumed that a changeover from one operation to another can be performed automatically and at negligible cost. However, only one part at a time can be processed at each machine. Typical examples of such machines might be NC machine tools or computer-controlled assembly robots. At each internal machine of the FFS, the conveyor system has by-pass links so that it is possible for parts to visit only some of the machines in their passage through the system. For additional system flexibility, each of the machines has an entry buffer of specified capacity; in the sequel, these buffers will be assumed to operate with a first-in first-out discipline. A schematic diagram of a typical FFS is shown in Fig. 1.

Suppose now that such a system is to be used to produce a specific range of part types in a prescribed constant ratio, and in sufficient
Fig. 1 - A 4-Machine Flexible Flowshop
volume to keep the system occupied for a reasonably long time. Each part type requires a prescribed sequence of operations, and we shall assume that this can be expressed by specifying the sequence of machines to be visited by each type, and the processing time required on each machine. This means, in particular, that whenever an operation can be carried out on more than one machine, an actual part type has been split into a number of "artificial" ones, each of them associated with a distinct feasible machine sequence, and that appropriate production ratios for the artificial part types have been determined independently by considering the balance of work among machines (see [2] for one method of doing this). In the sequel, only flow shops, or serial arrangements of machines, will be discussed. It will always be understood, therefore, that the machine sequence specified for each part type is compatible with the actual serial arrangement of machines.

In the literature on production scheduling, it is usually assumed that a setup cost is incurred in a changeover of a machine or production line from one part type to another. The production scheduling problem is then to determine the sizes and sequence of the batches in which the part types should be made so as to strike the best balance between setup costs and costs of holding in-process and finished inventory. In the flexible manufacturing systems considered here, however, setup or changeover costs are negligible so that an optimal mode of production will minimize work in progress by producing the required part types simultaneously rather than in a sequence of batches. The buffers in the system have the purpose of reducing machine idle time due to unequal processing times, and of providing a cushion against short-term machine breakdowns; their function is not to reduce the number of setups required.

We are thus led to the following production scheduling problem:

Given a description of the system (number of machines, buffer capacities, travel times between machines) and of the desired production (number of part types, machine sequence for each part type, processing times on the machines, production ratios required), determine in what sequence and at what intervals parts should be loaded into the system (i.e., the first machine) so that
(i) the output of finished parts is as large as possible, subject to the constraint that part types are produced in the prescribed ratio, and

(ii) The steady state of maximum production in the prescribed ratios is reached as quickly as possible after startup or a momentary disturbance.

In this report, a deterministic version of this problem will be considered. All processing and travel times are assumed known and fixed, and all machines are considered completely reliable. For simplicity, we shall also assume that processing and travel times are integral multiples of some fundamental time step.

With these assumptions, the problem can be formulated as a special type of jobshop scheduling problem. There is an extensive literature on this problem; excellent recent discussions of methods for scheduling both flowshops (e.g., transfer lines) and general jobshops can be found in [3], [4], [5], together with extensive references to earlier work. Except in a very few simple cases, both flowshop and jobshop scheduling are well known to belong to the category of hard-to-solve, "NP-complete" combinatorial problems [3], [6], [7] which seem to be characterized by such a dearth of structure that the only feasible solution methods found so far are implicit enumeration type tree searches. Particularly in the case of the jobshop problem, these searches often require a massive computational effort for problems of even moderate size. This seems at least in part due to the choice of optimization criterion. In all studies known to the author, the scheduling problem is stated as finding either the shortest total time (makespan) or the shortest average time weighted by jobs, in which a flowshop or jobshop can process a given fixed set of jobs.

For the systems considered in this report, this seems an unnecessarily strict optimization criterion. We shall confine attention to situations where the total number of parts to be produced by the FFS in one run is so large that the system can be considered to be in an optimal state of operation whenever it produces parts in the prescribed ratio and at a steady maximum rate, both averaged over suitable time intervals. The additional time saving possible by using schedules
which minimize makespan or average weighted finishing time, for the total set of parts in the run, will usually be very small.

The remainder of the report focuses on the problem of characterizing and computing loading sequences, or schedules, for the FFS which result in optimal steady state output, subject to constraints on the duration of startup transients, and on available buffer storages. It will be shown that substantial computational savings can be obtained by replacing the minimization of makespan (or mean weighted finishing time) with the requirement that the bottleneck machine or machines be fully occupied once they have started working.

A precise definition of what is meant by an optimal loading sequence is given in Chapter 2. There we will also establish the simple but basic result that loading sequences which lead to optimal steady state production, without constraints on buffer capacities or on the duration of start-up transients, form a very large class and are trivially easy to compute.

In Chapter 3, we shall introduce constraints on the length of transients as well as buffer storage, and describe an implicit enumeration algorithm for computing optimal loading sequences. Some initial computational results will be reported for the particular case of unit buffer capacity at all machines.

Finally, in Chapter 4 we shall briefly discuss how the ideas presented earlier might be extended and, in particular, how they might be incorporated in a closed-loop system of controlling production in a flexible manufacturing system.
2. MAXIMAL STEADY-STATE PRODUCTION IN AN FFS

Consider an FFS with K machines linked by a conveyor system such that the travel time $T_j$ from the machine $j$ to the input buffer of machine $k$ is known and fixed for all $1 \leq j < k \leq K$. Suppose that $M$ part types are to be produced. Let $p_{ij}$ denote the known fixed processing time required for a part of type $i$ on machine $j$; $p_{ij} = 0$ means that a part of type $i$ bypasses machine $j$. Suppose further that the part types are to be produced in the ratios $r_i$, $i = 1, 2, \ldots M$: of the total number $a$ of parts produced since startup, $r_ia$ parts are required to be of type $i$. We shall assume throughout that the $r_i$ are rational fractions. In some cases, particularly where the various part types are required for assembly into a larger unit, a tighter constraint may be appropriate: of every $N$ consecutive parts produced by the system, $n_i$ parts are required to be of type $i$, where $n_i = r_iN$, and where $N$ is a reasonably small number. This leads to the notion of a "Minimal Part Set" (MPS).

Definition: A Minimal Part Set is a set of integers $\{n_1, n_2, \ldots n_M\}$ such that

\[ n_i = r_i \sum_{j=1}^{M} n_j \leq r_i N, \quad i = 1, 2, \ldots M \]  

(1)

and

\[ \text{g.c.d.} \{n_1, n_2, \ldots n_M\} = 1 \]  

(2)

Clearly, the system satisfies the production ratio constraints if in every set of $N$ consecutive completed parts, $n_i$ are of type $i$, $i = 1, 2, \ldots M$. Moreover, it is impossible to be certain that the constraints are satisfied without checking at least the last $N$ parts produced. Thus the minimum time required to produce the $N$ parts is, in a sense, the shortest possible response time of the system. Ideally, this will be equal to the time $T$ required by the bottleneck (or most heavily loaded) machine or machines to process their share of work in an MPS. This time will be called a Period; it is given by
The scheduling problem studied here is to determine a loading sequence\{(\sigma_1, t_1), (\sigma_2, t_2), \ldots\}, (where (\sigma, t) means that a part of type \sigma is loaded into machine 1 at time t), which will cause the system to produce parts in the prescribed ratio and at the maximum possible rate. Clearly, the best steady state operation obtains when the system completes a minimal part set in every period T. This suggests the use of a periodic loading sequence of the form 

\{(\sigma_1, t_1), (\sigma_2, t_2), \ldots(\sigma_N, t_N), (\sigma_1, t_1+T), \ldots(\sigma_N, t_N+T), (\sigma_1, t_1+2T), \ldots\}

where \(t_1 < t_2 < \ldots < t_N, t_N-t_1<T\) and where \(\{\sigma_1, \sigma_2, \ldots \sigma_N\}\) is some permutation of the items in a minimal part set. Any such sequence will be called maximal periodic. It will be convenient to use the same phrase for sequences in which the strict inequalities \(t_i < t_{i+1}\) are relaxed to \(t_i < t_{i+1}^\prime\); such sequences can occur when subsequences merge at a conveyor junction inside the system (see F2 below). In that case, the \(t_i\) will not be instants at which the part enters the system but instants at which it passes the point where the sequence is observed.

Suppose now that the FFS has the following features:

F1: Each machine has a FIFO buffer of unlimited capacity.

F2: The conveyor system can carry an arbitrary number of pieces on each position. This means that no part emerging from a machine will be delayed in its journey to the next machine by traffic on the conveyor. It also implies that several parts may arrive at a machine buffer simultaneously. In order to maintain deterministic behaviour throughout the system, it is assumed that parts sharing a conveyor position and destined for the same machine leave the conveyor and enter the machine's buffer according to some arbitrary but fixed rule, e.g., LIFO.

For a system with these features, any one of a very large class of loading sequences will result in optimal steady-state production, as the following theorem shows.
Theorem: In a flexible flow shop with features F1 and F2, any maximal periodic loading sequence results, after a finite interval of time, in an output sequence which is itself maximal periodic.

To prove the theorem, we first establish a preliminary result on the response of a single machine.

Lemma: Suppose that parts arrive at the buffer of a machine in a periodic sequence such that the sum of processing times of the parts arriving in one period does not exceed the capacity of the machine. Then, after a finite time interval, the sequence of parts leaving the machine will be periodic with the same period as the arrival sequence.

Proof: Let \( w \) be the sum of processing times of the parts arriving in one period, let \( T \) denote the length of the period, and let \( s_1 \) be the instant at which the first part of the periodic input sequence arrives. Denote by \( x(t) \) the time required by the machine to finish the part being worked on and to process the parts waiting in the buffer just prior to the instant \( t \); \( x(t) \) can be regarded as a backlog of unfinished work in the machine and its buffer. Define \( \eta \) as the total time the machine would be idle in the first period, i.e., in the first \( T \) time steps after \( s_1 \), if the machine and buffer were empty just prior to the arrival of the first part at time \( s_1 \):

\[
\eta = \max_{1 \leq k \leq n+1} \left\{ (s_k - s_1) - \sum_{j=1}^{k-1} \pi_j \right\}
\]

where \( n \) is the number of parts arriving in a period, and where \( s_k, \pi_j \) are the arrival instants and processing times, respectively, of the \( k \)-th part to arrive.

For example, in Fig. 2(b), \( \eta \) equals the sum of the intervals labelled "a" and "b".

We distinguish two possibilities:

Case 1: The backlog of unfinished work never becomes zero, i.e., \( x(t) > 0 \) for all \( t > s_1 \). Clearly, this can happen only if the machine
Fig. 2: Unfinished Work in a Single Machine and Its Buffer with a Periodic Sequence of Arriving Parts

\( \omega \) = total work arriving in one period

\( \tau \) = period of input sequence

\( t_0 \) = beginning of periodic input sequence

vertical jumps are the processing times of arriving parts
needs a whole period to process the parts arriving in a period, i.e., if \( w = \tau \). In addition, the initial backlog \( x(s_1) \) needs to be large enough to prevent the machine from clearing its backlog sometime in the first period: \( x(s_1) > \eta \). This case is illustrated in Fig. 2(a). Under these conditions, it is obvious that for \( t > s_1 \), \( x(t) \) is a periodic function with period \( \tau \), and this implies that the sequence of parts leaving the machine is similarly periodic, beginning at the instant of departure \( s_1 + x(s_1) + \Pi_1 \) of the part arriving at \( s_1 \).

Case 2: The backlog of unfinished work becomes zero, i.e., the machine and its buffer become empty, at least once after time \( s_1 \): \( x(t) = 0 \) for at least one \( t > s_1 \). If \( w = \tau \), this will happen during the first period if the initial backlog \( x(s_1) \) is small enough, i.e., if \( x(s_1) < \eta \). See Fig. 2(b). On the other hand, if \( w < \tau \), so that the amount of work arriving in each period is strictly less than the capacity of the machine, then no matter how large the initial backlog \( x(s_1) \), an instant must arrive at which the machine has cleared its backlog, and begins an idle period of finite length. This is shown in Fig. 2(c).

Let \( \bar{t} \) be the smallest \( t > s_1 \), such that \( x(\bar{t}) = 0 \), and define \( \ell \) by that \( \ell \tau + s_1 < \bar{t} < (\ell+1) \tau + s_1 \). Then from time \( \bar{t} \) onwards, the amount of unfinished work in the machine and its buffer depend only on the input sequence, and not on any initial backlog \( x(s_1) \). This in turn implies that \( x(t) \) begins being periodic at some instant during the \( \ell \)-th period, and certainly no later than at time \( s_1 + \ell \tau \). However, a periodic backlog function implies a periodic output sequence. This proves the lemma.

Proof of Theorem: The first machine in the system is a loading station without an input buffer. Clearly, the sequence of parts leaving machine 1 will be maximal periodic from the instant of departure of the first part. Now consider machine 2. The sequence of parts arriving there is a subsequence of the maximal periodic sequence leaving machine 1, delayed by the constant travel time from the loading machine. It follows from the lemma that after some finite time, say \( \tau_2 \), the sequence of
parts leaving machine 2 is also periodic with period $T$. This sequence merges with the sequence of parts which bypass machine 2. Since both have the same period $T$, we conclude that from time $t_2$ onward, a maximal periodic sequence enters the conveyor linking machines 2 and 3. The theorem now follows from a simple repetition of this argument for each subsequent machine.

The theorem assures us that if the system has unlimited buffer capacity at each machine, and if only steady-state output is of interest, the scheduling problem has a simple solution: any maximal periodic loading sequence will do. After a finite time, an initially empty system will produce a minimal part set in every period.

Of course, a poor choice of loading sequence may result in a long transient. During this transient, the system will produce parts at less than the maximum possible rate, and since the sequence of parts will usually be permuted in its passage through the system, there is no assurance that an integral number of minimal part sets will be produced during the transient. Thus the output will in general satisfy the production ratio constraints only if the production during the transient is ignored.

It is possible to overcome this difficulty, as well as reduce the duration of the transient phase, by a partial preprocessing of parts which fills the machine buffers to an appropriate level prior to starting the maximal periodic loading sequence. This yields a heuristic scheduling algorithm which will be discussed in more detail in Chapter 4.

First, we turn to the problem of characterizing and computing maximal periodic loading sequences which are optimal in the sense that they minimize undesirable start-up transients, and satisfy explicit constraints on available buffer storage.

**Definition:** A maximal periodic loading sequence will be called **optimal** if, when it is applied to an initially empty FFS, the following hold:

(i) specified constraints on buffer capacity are satisfied

(ii) the output sequence of completed parts is maximal periodic from the instant at which the first part leaves the system.
We shall use the term "loading schedule", or simply schedule, to denote the first $N$ components of a maximal periodic loading sequence, i.e., the ordered set of pairs

$$\sigma = \{(\sigma_1, t_1), (\sigma_2, t_2), \ldots, (\sigma_N, t_N)\}$$

where, as before, $t_i < t_{i+1}$, $i = 1, 2, \ldots, N-1$, where $t_N - t_1 < T$ and where $\{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ is a permutation of the MPS.

Suppose now that parts are loaded into an initially empty FFS according to a given schedule $\sigma$. Let $S(l,j)$ denote the starting time, on machine $j$, of the first part of $\sigma$ to arrive at machine $j$, and let $C(N,j)$ be the completion time, on machine $j$, of the last part of $\sigma$ to be processed by machine $j$. Then the condition

$$C(N,j) - S(l,j) < T, \quad j = 1, 2, \ldots, K$$

is clearly sufficient for the schedule $\sigma$ to yield an optimal loading sequence: on every machine, and for all $k = 1, 2, \ldots$, processing of all the parts in the first $k$ minimal part sets is completed before the first part of the $(k+1)$-st MPS arrives. Hence the processing sequence on every machine, including the last, is periodic from the instant at which the first part reaches that machine.

An example of an FFS scheduling problem is given in Fig. 3, together with an optimal schedule which satisfies the constraint of unit buffer capacity at each machine. On the chart, two-digit numbers are used to identify individual parts. For example, "52" refers to the second part of type 5. This particular part is loaded at time 16, leaves machine 1 at time 18, bypasses machine 2 to arrive at the buffer of machine 3 at time 28, waits there for 3 time steps while part 31 is processed, leaves machine 3 at time 36, and so on. It leaves the system at time 74.

Note that condition (4) is not necessary for optimality of a schedule. Suppose that $C(N,j) - S(l,j) > T$ for some schedule $\sigma$ and some machine.

*Note, however, that this notation is not meant to imply that each machine is visited by all $N$ parts in an MPS.*
(a) Problem Data:

No of machines \( K = 6 \)
No of part types \( M = 5 \)
Minimal Part Set = \( \{2, 2, 1, 1, 3\} \); Period \( T = 42 \)

Travel times \( T_{jk} \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>17</td>
<td>12</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Operation times \( j \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Part</td>
<td>3</td>
<td></td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>4</td>
<td></td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>26</td>
<td>30</td>
<td>42</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>42</td>
<td>42</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

(b) Gantt Chart of Optimal Schedule

Optimal schedule \( C = \{(1, 0), (5, 4), (3, 6), (1, 10), (4, 14), (5, 16), (2, 21), (5, 25), (2, 27)\} \). Rectangular blocks above machine bars indicate occupied buffer shaded areas are idle times in one period.
j. Then $j < K$ since otherwise the output sequence could not have a minimal period. It may happen that the processing on machine $j$ of the last few items in the first MPS in the interval $[S(l,j) + T, C(N,j)]$ does not interfere with the processing of the first items of the next MPS in the same interval. If this is so for all machines for which $C(N,j) - S(l,j) > T$, then clearly the schedule can be repeated indefinitely; the processing of the $k$-th MPS will not affect the timing of the operations for the $(k+1)$-st MPS, and the system output has the same dead beat transient response as if (4) were satisfied.

However, from now on, we shall identify optimal schedules with those that satisfy (4), treating as negligible the possibility that the only optimal schedules for a problem violate (4). We do this because (4) leads to a fairly simple and efficient algorithm for computing optimal schedules; permitting some overlap in the processing of successive minimal part sets would make the algorithm very much more complex and computationally expensive.
3. COMPUTING OPTIMAL FFS SCHEDULES

In this chapter, we discuss the computation of schedules which satisfy condition (4), by means of an implicit enumeration algorithm. This algorithm can be regarded as a formal statement of the procedure one would naturally adopt in a manual construction of optimal schedules using Gantt charts [3]. To simplify the exposition, we shall first restrict the discussion to so-called permutation schedules [4]. In such schedules, processing of different parts on common machines must always occur in the order in which the parts entered the system. For example, the schedule in Fig. 3 is not a permutation schedule: part 21 enters the system before part 53, yet is processed on machines 3 and 4 after part 53.

Let $T_{nipij}$ denote the total time required by machine $j$ to process its share of work in a minimal part set. Then $U(O,j) = T_j - T_{nipij}$ is the maximum amount of time that machine $j$ can be idle in the interval $[S(l,j), C(N,j)]$ if condition (4) is to be satisfied by the schedule $\sigma$. Thus, an alternative way of stipulating (4) is to require that in an optimal schedule, the sum of the gaps, or idle periods, between successive work periods must not exceed $U(O,j)$ for $j = 1, 2, \ldots, K$. Of course, $U(O,j) = 0$ on a bottleneck machine. Let $\sigma_k$ denote a "partial schedule":

$$\sigma_k = \{(\sigma_1, t_1), (\sigma_2, t_2), \ldots, (\sigma_k, t_k)\}$$

where the $\sigma_i$ form a proper subset of the items in an MPS. For convenience, we usually take $t_1 = 0$. Let $\hat{U}(k,j)$ be the total idle time on machine $j$ between the work periods necessary to process $\sigma_k$; if only one, or no, part of $\sigma_k$ requires processing on machine $j$, set $\hat{U}(k,j) = 0$. Then $U(k,j) = U(O,j) - \hat{U}(k,j)$ is the amount of idle time available on machine $j$, and not yet "used up" by $\sigma_k$. We shall call $\sigma_k$ a feasible partial schedule if

(i) at no machine, the maximum number of buffer positions required in processing $\sigma_k$ exceeds the buffer capacity $B_j$. 

-15-
(ii) \( U(k,j) \geq 0 \) for all \( j = 1,2,\ldots K \).

Suppose that in the search for an optimal schedule for a particular problem, a feasible partial schedule \( \sigma_k \) has been found. We now attempt to find a feasible \( \sigma_{k+1} \). Let \( C(k,j) \) be the finishing time on machine \( j \) of the last item of \( \sigma_k \) to be processed by machine \( j \). Let \( A(k,j) \) denote the earliest time at which a new part from \( \sigma - \sigma_k \) can arrive at the buffer of machine \( j \), subject to the constraints that the buffer cannot be full at time \( A(k,j) \) and that processing of the new part on machine \( j \) cannot begin before time \( C(k,j) \). (Recall that only permutation schedules are allowed for the moment.)

The search algorithm consists of the following:

Step 1: Let \( F_1 = \{1,2,\ldots M\} \); for \( k > 1 \), \( F_k \) is the set of part types which may still be possible in the \( k \)-th place of an optimal schedule having \( \sigma_{k-1} \) in its first \( k-1 \) places.

Step 2: If \( F_1 \) is empty, stop; an optimal schedule does not exist. Otherwise, select a part type \( \sigma_1 \) from \( F_1 \), set \( t_1 = 0 \), and for all \( j = 1,2,\ldots K \), set \( U(1,j) = U(0,j) \) and compute \( A(1,j) \) and \( C(1,j) \). Set \( k = 1 \).

Step 3: Compute the set \( F_{k+1} \) of part types for which \( n_i - m_i > 0 \), where \( n_i \) and \( m_i \) are the numbers of parts of type \( i \) in the MPS and in \( \sigma_k \), respectively.

Step 4: If \( F_{k+1} \) is empty, go to step (7). Otherwise, select a part type \( \sigma_{k+1} \) from \( F_{k+1} \) according to some priority rule. Set \( S(k+1,j) = 0 \) for \( j = 1,2,\ldots K \).

Step 5: Compute the earliest possible times \( S(k+1,j) \) at which a part of type \( \sigma_{k+1} \) can begin being worked on by each of the machines \( j \) in its route, subject to the joint constraints imposed by the \( A(k,j) \). If \( S(k+1,j) - C(k,j) > U(k,j) \) for any machine \( j \) visited by part type \( \sigma_{k+1} \), delete part type \( \sigma_{k+1} \) from \( F_{k+1} \), and return to step 4.
Step 6: A feasible "continuation" of $\sigma_k^{-1}$ has been found:

$$\sigma_{k+1}^{-1} = \{(\sigma_{k, 0}), \ldots, (\sigma_{k, t_k}), (\sigma_{k+1, S(k+1, 1)})\}.$$

If $k = N-1$, stop; $\sigma_{k+1}^{-1}$ is an optimal schedule.
Otherwise, compute and store $U(k+1, j)$, $C(k+1, j)$, $A(k+1, j)$, record $F_{k+1}$ for possible future reference, replace $k$ by $k+1$ and return to step 3.

Step 7: A feasible continuation of $\sigma_k$ does not exist, i.e., no optimal permutation schedule exists which has $\sigma_k$ as its first $k$ entries. Delete part type $\sigma_k$ from $F_k$. If $k = 1$, go to step 2. Otherwise, replace $k$ by $k-1$ and go to step 4.

The major computational effort in the algorithm is required by the computation of the earliest possible starting times in step 6. This (possibly iterative) computation can be accomplished by the following block of Fortran code ($LEV \equiv k$, $i \equiv \sigma_k$):
framework, the algorithm has two distinctive features:

(i) Minimization of a single-valued objective function (e.g., makespan) is replaced by searching for schedules satisfying a set of constraints: the sums of idle times between busy periods on each machine must not exceed the known limits $U(0,j)$.

(ii) During the search, the idle time on machine $j$ of the partial schedule $\{a_k, (a_{k+1}, S(k+1,1))\}$ is used as a lower bound on the idle time on machine $j$ of any permutation schedule of the form $\{a_k, (a_{k+1}, S(k+1,1)), \ldots (a_N, t_N)\}$.

The first of these features by-passes the difficulty in many branch-and-bound schemes of obtaining a tight upper bound early in the search. In our case, perfectly tight upper bounds are known from the outset. The second feature yields lower bounds which are easy to compute, at least in comparison to the effort required to obtain good lower bounds in the branch-and-bound solution of the classic permutation flowshop problem [4].

It was hoped that these features would combine to yield a computationally efficient algorithm. This turned out to be the case, at least in regard to speed of computation. A program was written in Fortran IV for the special case of unit buffer capacities at all machines, and tested on an IBM 370/168 with a number of sizeable problems derived from those described in [1]. The search tree generated was usually quite large; for problems with $K = 6$, $M = 5$ and $N = 25$, several thousand nodes were usually required to find an optimal permutation schedule, or to establish that none existed. However, the computation time per node was only 0.7 msec on average.

It also became apparent that the restriction to permutation schedules is too drastic in an FFS scheduling problem involving different processing times and part types which do not require processing on every machine. In such cases, permutation schedules often fail to take advantage of the key feature which can make the FFS more efficient than a transfer line: the ability of parts to reach a downstream machine and begin being processed there while an upstream machine is still busy on a part which entered the system earlier. The
example in Fig. 3 is a case in point: no optimal permutation schedule exists (the program concluded this after generating 32 nodes in the search tree).

It seems that the simplest way of removing the restriction to permutation schedules is to enable the algorithm to look for feasible continuations of a partial schedule not just by adding a single part, but by adding a group of several parts having "interlocked processing profiles" on a Gantt chart, i.e., which overtake each other in their passage through the system. For example, in the schedule of Fig. 3, parts 41 and 52 form a pair of interlocked parts, or a "composite part". Together they yield a feasible continuation of the partial schedule

\[ \sigma_4 = \{(1,0),(5,4),(3,6),(1,10)\} \]

Individually they do not: neither \( \sigma_4, (4,t_5) \) nor \( \sigma_4, (5,t_5) \) are feasible for the smallest possible values of \( t_5 \). The former continuation would force machine 3 to be idle between time steps 31 and 35, while with the latter, machine 5 would be idle between times 58 and 60.

Naturally, an algorithm which can cope with such "composite parts" is considerably more complex. It seems that the simplest program structure is obtained when composite parts are treated as different part types. The set of possible composites can be pre-computed. However, the timing of the operations required to process the composite, and thus the amount of idle time between these operations, depend on the partial schedule to which the composite is joined and cannot be computed beforehand. For large composites, this computation is quite complex and time-consuming.

The Fortran program in the Appendix is a first attempt at developing a code which can handle composite parts. Only the special case of unit buffer capacity is considered, and composites can consist of only two parts. In other words, the schedules generated are restricted to those in which no more than two successive parts can overtake each other (once, or more frequently), and any such "crossed pair" cannot itself be interlocked with its neighbors.
The program was tested on one small and four substantial problems; see the Appendix for problem data and printouts. The results are summarized below:

<table>
<thead>
<tr>
<th>Problem No.</th>
<th>K</th>
<th>M</th>
<th>N</th>
<th>T</th>
<th>U(0,j)</th>
<th>Result</th>
<th>Nodes Generated in Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Fig. 3)</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>42</td>
<td>(16,12,0,4,0,29)</td>
<td>Opt. Soln.</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>5</td>
<td>.25</td>
<td>119</td>
<td>(49,46,0,2,1,84)</td>
<td>Opt. Soln.</td>
<td>370</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>25</td>
<td>117</td>
<td>(57,44,48,0,5,82)</td>
<td>Opt. Soln.</td>
<td>26961</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
<td>21</td>
<td>143</td>
<td>(75,5,38,0,25,109)</td>
<td>Opt. Soln.</td>
<td>8812</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>21</td>
<td>143</td>
<td>(75,5,38,0,1,109)</td>
<td>No Opt.Soln exists</td>
<td>1667</td>
</tr>
</tbody>
</table>

On an IBM 370/168, the average computing time per node was 1.6 msec (up from 0.7 for the algorithm for permutation schedules), so that a computation time of the order of several seconds can be expected on a powerful computer for problems of reasonable size (K ≤ 6, N ≤ 20).

It is of interest to compare this with the computational results reported in [4] for the minimum-makespan permutation flow shop problem with no buffer constraints. There it is stated that "fewer than half of the 20/5 problems [20 jobs, 5 machines] could be solved within one minute" of CPU time on a CDC Cyber 73-28 computer.

Thus, while the set of problems solved so far is too small to allow an accurate assessment of the performance of the algorithm, the results are sufficiently encouraging to warrant further development.

It seems that the difficulty of solving a problem, apart from sheer size, depends on

(i) the range of processing times among different parts in an MPS,

(ii) the balance of work among machines in an MPS,

(iii) the buffer capacity.
The first of these determines the amount of overtaking possible, and hence the size of composite parts which may be necessary in constructing optimal schedules. The results obtained so far indicate that unless the processing times of different parts on different machines differ by whole orders of magnitude, it may not be necessary to consider composites of more than three parts. Allowing pairwise overtaking already gave a dramatic improvement in the success rate of the algorithm, even in cases where the ratio of maximum to minimum processing times on internal machines was as high as 5 (see problem 5 in the appendix). In networks of NC machine tools, at least, this value can be regarded as a reasonable upper limit on the range of processing times.

The second factor, work balance among machines, affects the tightness of the constraints. If there is a high degree of unbalance, i.e., a large percentage of idle time in an optimal schedule on all machines except one bottleneck one, then in general there will exist a large number of optimal schedules, and only a very "narrow" portion of the search tree needs to be explored to find one. At the other extreme, when many machines are nearly fully occupied in an optimal schedule, the search will also be very brief: because of the tightness of the idle time constraints, even small feasible partial schedules will in general have very few feasible descendants (see problems 1,2,5). The most difficult problems seem to be those where the number of optimal schedules is not very large, but where the idle time constraints are not very tight either, at least early in the search (see problem 3). In that case, pruning of the search tree does not begin until a substantial depth is reached, so that many branches must be explored to a considerable depth. For this case, it may be desirable to augment the simple feasibility tests by calculating a more elaborate but tighter lower bound on the idle time of any completion of a partial schedule.

The third factor, buffer capacity, is, of course, an external constraint. However, unpublished simulation studies at MIT and the C.S. Draper Laboratories [8] seem to indicate that the number of optimal schedules (and hence the speed with which a search procedure can find one) increases dramatically as the buffer capacity at internal machines
is increased. It is of interest to explore this conjecture further.

An algorithm which removes the major limitations of the program in
the Appendix is currently being developed.

The objectives in the revision are to

(i) allow buffer capacities of arbitrary size,
(ii) extend the size of composite parts to three,
(iii) avoid multiple occupancy of conveyor positions. The present program merely avoid simultaneous arrivals of parts at a machine buffer.
(iv) minimize program complexity and overhead during the search. The present program is not very elegant or efficient; a considerable amount of unnecessary overhead is incurred in the storage of partial schedules, and in their modification during backtracking steps.
(v) incorporate a number of options if an optimal schedule does not exist or cannot be found in a specified amount of computing time. There are several possible alternatives: (a) to compute a good completion of the longest feasible partial schedule found, (b) to repeat the search with a larger value of the period T, i.e., with an identical increase in the allowable idle times U(0,j) of all machines, and (c) to repeat the search with greater buffer capacity.

Results of this revision will be presented in a sequel to this report.
4. EXTENSIONS AND CONCLUSIONS

In this chapter, we shall briefly discuss a number of desirable extensions of the ideas presented earlier, and then present some conclusions.

4.1 Closed-Loop Control of a Flexible Flow Shop

The computation of optimal schedules described so far in this report is based on the assumption of a perfectly deterministic system model: transport and processing times are known and fixed, machines never fail and always produce perfect work, and parts and pallets are never in short supply at the loading station. However, even in systems where transport and processing times are accurately known, such as in networks using assembly robots, NC machine tools or automatic inspection equipment, breakdowns will occur, or the work done may deteriorate so that parts will be rejected. Thus, schedules computed on the assumption of a perfect system will be useful only to the extent that they can be incorporated in a closed-loop or feedback control arrangement which can detect deviations from optimal behavior and make appropriate corrections.

We shall briefly outline some of the problems which will need to be solved in deriving such closed-loop control systems.

Consider first the problem of handling machine failures in an otherwise deterministic system. In the case of transfer lines, the standard method of reducing the effect of breakdowns is to provide buffer space between machines [9], [10]. With such intermediate storage, the neighbors of a broken-down machine can continue operating until the buffer upstream of the stoppage fills up, or the downstream buffer is depleted. Thus the effects of short-term stoppages can be effectively localized.

It seems reasonable to expect that the provision of adequate buffer capacity will be just as important in an FFS as in a transfer line, at least in the case of failures of short to medium duration. Naturally, a schedule which is optimal for the case of full machine availability cannot in general be maintained during a breakdown. What is needed then is an algorithm which makes the appropriate changes in the loading sequence, on the basis of the current buffer levels and machine states, so as to drive the system back to optimal operation after a breakdown. It is
tempting to speculate that if the diversity of part types and machine routes is properly exploited in such an algorithm, then a good insulation of the system against short-term individual machine breakdowns can be obtained with less buffer capacity than would be required in a transfer line.

The problem of breakdowns of long duration is more straightforward. If the disabled machine is essential for all part types being produced, then, of course, there is no alternative to shutting the system down once buffer stocks and storage places are exhausted. However, it may be possible to bypass the disabled machine and to have its function taken over by other machines. This will alter the routing for some or all of the part types. A new allocation of items to machine routes can then be computed (e.g., by the methods in [2]), and a new optimal schedule generated for use during the breakdown period. The "dead-beat" response to optimal schedules will ensure a rapid settling down to the new steady state.

It may not be possible to maintain the original production ratios with a breakdown in the system. In that case, it may be advantageous to produce whatever mix of parts can be made efficiently with the reduced system during the breakdown, followed by a "catch-up" phase in which the backlog of missing parts is cleared. The problem then is to determine what production ratios should be used in each of the two phases so that the total production loss due to the breakdown is minimized. How to formulate this problem in the simplest way (possibly as a small integer programming problem) and solve it efficiently, requires further investigation.

Apart from machine breakdowns, an online feedback control system must also be able to handle occasional rejection of unfinished parts. This may occur as a result of outright machine failure, or by a gradual deterioration of work. As a result, the total part production plus the set of parts currently in the system may not add up to an integral number of minimal part sets. The scheduling algorithm must then determine how this backlog can be cleared with a minimum of idle time on bottleneck machines.
4.2 Fast Heuristics for Near-Optimal FFS Schedules

The implicit enumeration algorithm discussed in Chapter 3 for computing optimal schedules (and hence loading sequences) may require more computing time or computer capacity than is available for an online control system. It is therefore of interest to investigate alternative algorithms in which the tight constraints on dead-beat response, on buffer capacity, or on maximal steady-state production are relaxed.

One possible approach is to exploit the result of the theorem in Chapter 2, by initially ignoring constraints on buffer capacity. This yields the following procedure:

Step 1: Assume a reasonable schedule \( \{(\sigma_1, t_1), \ldots, (\sigma_N, t_N)\} \).
One plausible scheme is to schedule parts of each type at roughly uniform intervals.

Step 2: Assuming unlimited buffer capacity, compute the steady-state response of the initially empty system to the maximal periodic loading sequence derived from the schedule. By steady-state response we mean the detailed processing sequence of a minimal part set at each machine once the system has settled down to steady state, i.e., once the output sequence has started being periodic with period \( T \). According to our theorem, this must occur after a finite number of periods. This step essentially requires a simulation of the system.

Step 3: Scan the steady-state response obtained in Step 2 for the maximum buffer occupancy at each machine. If this does not exceed available buffer capacity, stop; a feasible schedule has been found with which a maximal period loading sequence can be formed. If the available buffer capacity is exceeded, attempt to modify the current schedule.
so as to reduce peak buffer occupancy and return to step 2. If no further changes are possible or apparent, go to step 4.

Step 4: Expand the current schedule by the minimum amount of time necessary so that the peak buffer occupancies are reduced to match available capacity. If this time is \( \delta \), say, then the resulting schedule can be used to form a periodic loading sequence of period \( T+\delta \). This sequence will be feasible in that buffer constraints are satisfied, but since a minimal part set is produced only once every \( T+\delta \) steps, the production rate may be less than the maximum possible.

An important feature of the optimal schedules generated by the implicit enumeration algorithm in Chapter 3 is that the first part to arrive at each machine does not wait in the buffer. This guarantees that when the corresponding loading sequence is applied to an empty system, the output sequence is immediately periodic. This is no longer true in general for the loading sequences obtained in steps 3 or 4 of the heuristic described above. A finite number of minimal part sets must pass through the system before the output becomes periodic. It may be possible to shorten this transient phase with the following trick.

Let \( \lambda \) be the number of minimal part sets processed in the transient phase starting at \( t_1 = 0 \). This is obtained as a byproduct of the computation in step 2. That computation also establishes for each machine

(i) the instant of arrival at the buffer of machine \( j \) of the first part of the \((\lambda+1)\)-st minimal part set to be processed by machine \( j \), say \( a_j \),

(ii) the state of the buffer, i.e., the number of parts, and their type and order, waiting at time \( a_j \).

Now suppose that it is possible to arrange a pre-processing phase in the system prior to applying the periodic loading sequence at time
$t_1 = 0$, so that the buffer of machine $j$ has the state at time $a_j = a_1$ which it would reach at time $a_j$ if the sequence were applied to an empty system. In that case, the system will settle down to a periodic steady state immediately after the pre-processing stage, and the output satisfies the production ratio constraints from the instant at which the first minimal part set is completed.

The heuristic described above is currently being developed in detail. The important features needed will be a fast procedure for computing the steady-state response to a periodic loading sequence (step 2), and (in step 3) effective rules for changing a schedule so as to reduce peak buffer requirements. The latter may well turn out to be less difficult than might be expected, at least for systems in which a reasonable amount of buffer capacity is available.

4.3 Conclusions

The system described in this report is the first step of an attempt to find out whether on-line production control in a flexible manufacturing system using deterministic scheduling theory is likely to be useful and computationally feasible. The current literature on flowshop and jobshop scheduling is certainly not encouraging: even in problems of moderate size, the minimization of total processing time for a fixed set of jobs is shown to require substantial computational resources.

The present study has examined a reformulation of the traditional scheduling goals, arguing that for the case of reasonably long production runs, schedules which quickly yield a maximum steady-state production rate are just as acceptable as those which minimize the total duration of a production run. At least for the flexible flow-shops considered, it appears that the reformulation described in Chapter 2 substantially reduces the computational burden.

The computation of such schedules by an implicit enumeration scheme was discussed in Chapter 3. The results obtained with a first version of this algorithm are encouraging. They indicate that in certain types of flexible flow shops, on-line control of production based on optimal solutions to deterministic scheduling problems may well be perfectly feasible, and may yield significant economic benefits. Typical examples of such systems are networks of N/C machine tools, which have fairly long but quite accurately known
processing times and are in many cases already equipped with on-line computer facilities. Of course, the development of effective closed-loop control algorithms, using deterministic scheduling, requires much further research; some of the problems to be resolved were outlined earlier in this Chapter.

More research is also needed on how to extend the ideas presented here to systems other than flexible flow shops. Even in the case of an FFS, the formulation of the scheduling problem used here may not be the most natural or simple. An example might be a system having several identical machines. The requirement of the present model, that parts following different machine routes must be of different type, can then lead to an unduly large number of part types (many of them artificially different) and a large minimal part set. For such systems, a scheduling model is required which allows for alternative machine routes in a more direct fashion.

In an FFS, the part routes must be compatible with the actual sequence of machines. In a general flexible manufacturing system, arbitrary part routes are possible; a simple example is a set of machines connected by a loop conveyor. In such systems, the computation of optimal schedules (as defined here) is likely to be much more difficult. It is hard to see how the fairly simple algorithm of Chapter 3 can be extended to allow arbitrary routing of parts. A more promising approach might be to develop a heuristic scheduling procedure along the lines of the algorithm in Section 4.2. As a first step in this direction it will be necessary to generalize the theorem in Chapter 2 to systems which permit arbitrary part routes.
APPENDIX

A FORTRAN PROGRAM FOR COMPUTING
OPTIMAL FFS SCHEDULES
### Al: List of Parameters and Variables

(CPS = current partial schedule)

<table>
<thead>
<tr>
<th>Input data:</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>$K$</td>
<td>no of machines</td>
</tr>
<tr>
<td>$M$</td>
<td>$M$</td>
<td>no of part types</td>
</tr>
<tr>
<td>TRTIME $(I,J)$</td>
<td>$T_{ij}$</td>
<td>travel time from machine $i$ to entry buffer of machine $j$.</td>
</tr>
<tr>
<td>JOBNO $(I)$</td>
<td>$n_i$</td>
<td>no of parts of type $i$ in a minimal part set</td>
</tr>
<tr>
<td>JOBTIM$(I,J)$</td>
<td>$P_{ij}$</td>
<td>processing time of a part of type $i$ on machine $j$.</td>
</tr>
<tr>
<td>MAXNOD</td>
<td></td>
<td>maximum no of partial schedules examined for possible continuation before search is terminated.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters defined from input data:</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$N$</td>
<td>no of parts in an MPS</td>
</tr>
<tr>
<td>MACH$(I)$</td>
<td></td>
<td>no of machines visited by part of type $i$</td>
</tr>
<tr>
<td>JOBRTE$(I,L)$</td>
<td></td>
<td>index of machine visited by a part of type $i$ for its $L$-th operation.</td>
</tr>
<tr>
<td>PERIOD</td>
<td>$T$</td>
<td>time required to process MPS on bottleneck machine without idle time.</td>
</tr>
<tr>
<td>MAXPRT$(J)$</td>
<td></td>
<td>no of parts in the MPS which must be processed on machine $J$.</td>
</tr>
<tr>
<td>CRSPR$(I)$</td>
<td></td>
<td>no of part types from which a crossed pair can be formed with a part of type $i$</td>
</tr>
<tr>
<td>OTAKE$(J,I)$</td>
<td></td>
<td>index of $i$-th part type which can form a crossed pair with, or overtake, a part of type $J$</td>
</tr>
<tr>
<td>REACH$(I,J)$</td>
<td></td>
<td>minimum time in which some part in the MPS can reach machine $J$ after its arrival at machine $i$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMJOB$(I)$</td>
<td></td>
<td>no of parts of type $i$ not included in the CPS (initially, REMJOB$(I) = JOBNO$(I))</td>
</tr>
<tr>
<td>SLACK$(J)$</td>
<td>$U(k,j)$</td>
<td>amount of permissible idle time on machine $J$ not yet used up by the CPS.</td>
</tr>
<tr>
<td>Program Label</td>
<td>Symbol used in text</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>LEVEL</td>
<td>$k$</td>
<td>no of parts in the CPS (=depth of search tree). If LEVEL = N, search terminates; an optimal schedule has been found.</td>
</tr>
<tr>
<td>PFERM($L$)</td>
<td>$\sigma_L$</td>
<td>index of part type of $\lambda$-th part in the CPS ($\lambda = 1, 2, \ldots \text{LEVEL}$)</td>
</tr>
</tbody>
</table>
| XPERM($L$)   |                     | $=1$ if $\lambda$-th part in the CPS either over-takes its successor or is overtaken by its predecessor  
|              |                     | $=0$ otherwise |
| NCOUNT       | $\ast$              | no of nodes (partial schedules) in search tree examined so far |
| BIGLEV       |                     | no of parts in largest feasible partial schedule found so far |
| MCPART($J$)  |                     | no of parts processed by machine J in the CPS |
| MSCHED($I,J$)|                     | index of part type of the $i$-th part processed by machine J in the CPS |
| START($I,J$) |                     | time at which machine J begins processing its $i$-th part in the current partial schedule |
| FINISH($I,J$)|                     | time at which machine J completes its $i$-th part in the CPS |
| BUFF($I,J$)  | $= 0$ if the $i$-th part visiting machine J in the CPS has to wait in the entry buffer of machine J  
|              | $= 1$ if the $i$-th part visiting machine J in the CPS has to wait in the entry buffer of machine J  
|              | $= 0$ otherwise |
| NODE($L,I$)  | $= -1$ if it has been discovered that the partial schedule $\{\text{PFERM}(1), \ldots \text{PFERM}(L), I\}$ is not feasible ($1 < I < M$) because either all parts of type $I$ in the MPS are already included in the first $L$ parts or because the earliest possible addition of $I$ uses up more idle time on some machine than is left in a schedule starting with $\{\text{PFERM}(1), \ldots \text{PFERM}(L)\}$, or because all feasible continuations of $\{\text{PFERM}(1), \ldots \text{PFERM}(L), I\}$ have been found to have no feasible continuations themselves. Thus NODE stores the search tree. If NODE ($1,I$) = -1 for all $I = 1, 2, \ldots M$, the search is complete; an optimal schedule does not exist.  
<p>|              | $= 0$ otherwise |</p>
<table>
<thead>
<tr>
<th>Program Label</th>
<th>Symbol used in text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLAST(J)</td>
<td>C(k,j)</td>
<td>Time at which machine J completes the last operation in the CPS.</td>
</tr>
<tr>
<td>WLAST(J)</td>
<td></td>
<td>Length of last operation on machine J in the CPS.</td>
</tr>
<tr>
<td>STRT1(J)</td>
<td></td>
<td>Tentative starting time on machine J of a possible continuation J1 of the CPS.</td>
</tr>
<tr>
<td>FIN1(J)</td>
<td></td>
<td>Tentative finishing time on machine J of a possible continuation J1 of the CPS.</td>
</tr>
<tr>
<td>STRT2(J)</td>
<td></td>
<td>Tentative starting time on machine J of part J2 in a possible continuation of the CPS by a crossed pair J1, J2.</td>
</tr>
<tr>
<td>FIN2(J)</td>
<td></td>
<td>Tentative finishing time on machine J of part J2 in a possible continuation of the CPS by a crossed pair J1, J2.</td>
</tr>
<tr>
<td>J1</td>
<td>$\sigma_{k+1}$</td>
<td>Part type of possible continuation of the CPS.</td>
</tr>
<tr>
<td>J2</td>
<td></td>
<td>Part type of possible continuation of the CPS by a crossed pair J1, J2.</td>
</tr>
</tbody>
</table>
A2: Listing of Fortran Program
C PROGRAM USES AN IMPLICIT FORMULATION TO SEARCH FOR A SATURATED SCHEDULE C FOR STEADY-STATE PRODUCTION IN A FLOWSHOP OR AT MOST K = 9 MACHINES WITH C (1) FIXED ENTRY TIMES BETWEEN MACHINES (INCL ONE TIME STEP IN ENTRY C (2) NO IDLE TIME AT ANY MACHINES C (3) BY-PASS CONVEYORS AT ALL MACHINES EXCEPT THE FIRST AND LAST C (WHICH ARE LOADING AND UNLOADING STATIONS VISITED BY ALL PARTS)
C THERE ARE UP TO M=9 ITEM TYPES TO BE PROCESSED, EACH WITH JOBNO(1) PARTS C IN A 'MINIMAL PART SET' (MPS): A SMALLEST SET OF INTEGER PART NUMBERS C WHICH SATISFIES THE SPECIFIED PRODUCTION RATIOS.
C THE TOTAL NO OF ITEMS IN AN MPS MUST NOT EXCEED 99
C A SATURATED SCHEDULE IS ONE WHICH PROCESSES THE MPS WITHOUT IDLE TIME C ON THE BOTTLENECK MACHINES AND WITHIN A TIME ON ALL OTHER MACHINES C LESS THAN OR EQUAL TO THAT REQUIRED ON THE BOTTLENECK MACHINES. HOWEVER, C THE FIRST ITEM IN A SATURATED SCHEDULE DOES NOT WAIT IN ANY INPUT C BUFFER LONGER THAN THE MANDATORY ONE TIME STEP.
C IF PRODUCED C卅PE AND OVER, A SATURATED SCHEDULE HtS GIVES MAXIMUM C STEADY-STATE OUTPUT, AS WELL AS A DEADBEAT TRANSIENT RESPONSE.
C THE OUTPUT IS PERIODIC FROM THE INSTANT AT WHICH THE FIRST PART LEAVES C THE SYSTEM.
C THE FOLLOWING LIMITATIONS APPLY TO THE SCHEDULES GENERATED:
C (1) EACH ITEM CAN OVERTAKE (AND MAY BE AGAIN OVERTAKEN BY) AT C MOST TWO OTHER PIECES. OVERTAKING CAN OCCUR ONLY WHEN BY-PASSING C A MACHINE. ALL BUFFERS ARE SINGLE-POSITION AND FIFO.
C (2) THE PROGRAM DOES NOT CHECK FOR POSSIBLE CLASHES ON THE CONVEYSOR C WHEN A PART LEAVES A MACHINE. HOWEVER, IT WILL AVOID SIMULTANEOUS C VIRTUES AT A MACHINE.
C IF A SATURATED SCHEDULE, RESTRICTED AS DESCRIBED, DOES NOT EXIT OR IS NOT C FOUND AFTER GENERATING MAXLED NODES IN THE SEARCH TREE, THE LARGEST C SATISFACHT SCHEDULE FOUND IS PRINTED.
C
C IMPLICIT INTEGER (A-Z)
REAL P, XP
LOGICAL IFSC, IFST, TEST
COMMON/F, N, BI, FF, XPMT(9), XAPRT(9), MAXPT(9), SLACK(9),
1 MACH(9), PSEP (9), STMT (9), REACH (9, 9), JOBIT (9, 9), RTIME (9, 9),
2 MORTT(9, 9), PFTT (9), PVE (9), STK1 (9), STK2 (9), STADT (9, 9), FINISH (9, 9),
3 OFF (9, 9), PRTM (9), XPER (9), NODE (9, 9), TLAST (9),
4, MSCRD (20, 9), SCFF (85, 9), SASF (99, 9), SPRT (99, 9),
5, SPRT (9), SAPP (99)
DIMENSION F(1), APT(9), MSCRD(500, 9), BUPT (500, 9), JOBNO(9)
NPARS=6
WINT=6

C DATA INPUT: F, N, XAPRT; PTIME; JOBIT; JOBNO. DATA CARDS SHOULD BE FOLLOWED
C DEFINE PROBLEM PARAMETERS: N, REMJOB, JOBTE, MACH, PERIOD, SLACK, MAXPRT, CRSPE.
C OTAKE, BEACH

C

C 60 I=1,N
REMJOB(I) = JOBNO(I)
M=1+JOBNO(I)
L=
DO 12 J=1,K
IF (JOBTIM(I,J) .EQ. 0) GO TO 32
L=I+1
JOBTM(I,L)=J
12 CONTINUE

C 32 CONTINUE
C

C 37 M=1,K
SLACK(MC)=0
MAXPRT('C')=0
DO 36 MC=1,K
IF (SLACK(MC) .GT. PERIOD) PERIOD=SLACK(MC)
36 CONTINUE
C
I -36-

```
C

DO 70 I = 1, K
I = I + 1
DO 70 I = 1, K
70 REACH(I,J) = 10000
C

DO 82 TYPE = 1, M
I = I + 1
TIME = JOBTIM(TYPE, I)
IF (TIME .LE. 0) GO TO 82
WCM = I
DO 84 J = 1, K
TMP = JOBTIM(TYPE, J)
IF (TMP .EQ. 0) GO TO 84
TIME = TIME + TMP
REACH(I,J) = TIME
TIME = TIME + TMP
WCM = J
84 CONTINUE
82 CONTINUE
90 CONTINUE
C

C OUTPUT PROBLEM DATA
C

DO 72 I = 1, K
```

72 \text{\textbf{\texttt{RAND(1)=1}}}
   \left\text{\texttt{WRITE(9,932)}} \right
902 \text{\textbf{\texttt{FORMAT(1H,9X,'PROBLEM DATA:','I2')}}}
   \left\text{\texttt{\textbf{\texttt{V1=1)}}} \right
   \left\text{\texttt{WRITE(9,994) \text{l\texttt{\texttt{R\texttt{AND(1)=1,K)}}}}} \right
904 \text{\textbf{\texttt{FORMAT(1H,5X,'TRAVEL TIMES BETWEEN MACHINES:','J2')}}}
   \left\text{\texttt{\textbf{\texttt{V1=1)}}} \right
   \left\text{\texttt{WRITE(9,935)}} \right
905 \text{\textbf{\texttt{FORMAT(1H)}}}
   \left\text{\texttt{L=74 \textbf{\texttt{I=2,K)}}} \right
78 \text{\textbf{\texttt{WRITE(9,961)}} \texttt{L,1} \left\text{\texttt{(TRIT(R,L),L=1,11)}} \right
906 \text{\textbf{\texttt{FORMAT(1H,4X,I2,1X,914)}}}
   \left\text{\texttt{\textbf{\texttt{WRITE(9,966)}} \texttt{R,1) \textbf{\texttt{RAN}}} \right
908 \text{\textbf{\texttt{FORMAT(1H,5X,'NO OF ITEM TYPES:','J2/I80,22X,'ITEM TYPE','4X,}}}
   \left\text{\texttt{\textbf{\texttt{V1=1)}}} \right
   \left\text{\texttt{\textbf{\texttt{WRITE(9,905)}}} \right
   \left\text{\texttt{DO \textbf{\texttt{L=1,K}}}} \right
76 \text{\textbf{\texttt{WRITE(9,101) \textbf{\texttt{L,JOBT}(R,J),J=1,K)}}}
910 \text{\textbf{\texttt{FORMAT(1H,27X,11,9X,I2,4X,914)}}}
   \left\text{\texttt{\textbf{\texttt{WRITE(9,915)}} \texttt{J=1,K)}} \right
915 \text{\textbf{\texttt{FORMAT(1H,16X,'PERIOD =','J4,6X,'SLACK(J)'),'1X,8I4)}}
   \left\text{\texttt{\textbf{\texttt{WRITE(9,911)}}} \right
911 \text{\textbf{\texttt{FORMAT(1H0,'ITEM TYPE CRESPE OTAKE')}}}
   \left\text{\texttt{DO \textbf{\texttt{R=1,K}}}} \right
36 \text{\textbf{\texttt{WRITE(9,912) \texttt{R,PEAK((R,L),L=1,11)}}}
912 \text{\textbf{\texttt{FORMAT(1H,16,16,4X,912)}}}
   \left\text{\texttt{\textbf{\texttt{WRITE(9,917)}}} \right
917 \text{\textbf{\texttt{FORMAT(1H,'PEACH(MC,MC)','=')}}}
   \left\text{\texttt{DO \textbf{\texttt{MC=1,K}}}} \right
48 \text{\textbf{\texttt{WRITE(9,916) \texttt{PEACH(MC,MC),MC=MC1,K)}}}
916 \text{\textbf{\texttt{FORMAT(1H,'0I4)}}}

\textbf{C}
\begin{enumerate}
\item Do 44 \texttt{PC=1,K}
   \begin{itemize}
   \item \texttt{PEIT(P1)=0}
   \item \texttt{YPERM(PC)=0}
   \end{itemize}
   \begin{itemize}
   \item Do 42 \texttt{MC=1,K}
   \item \texttt{MSCHRY(PC,MC)=MC}
   \item \texttt{START(PC,MC)=0}
   \item \texttt{FINISH(PC,MC)=0}
   \item \texttt{HYPPR(2,MC)=0}
   \end{itemize}
   \begin{itemize}
   \item Contimue
   \end{itemize}
   \begin{itemize}
   \item Do 43 \texttt{PRT=1,K}
   \item \texttt{MODE(PC,PRP)=0}
   \end{itemize}
\end{enumerate}

47 \textbf{\texttt{C}}
\begin{enumerate}
\item Do 44 \texttt{MC=1,K}
   \begin{itemize}
   \item \texttt{TIAS(PC)=0}
   \item \texttt{DIAS(PC)=0}
   \end{itemize}
   \begin{itemize}
   \item \texttt{START(MC)=0}
   \end{itemize}
\end{enumerate}
CASE 2, 0

DATE = 79017 15/06/17

SUBCASE 2, 0

FIND (SC) = 0
FIND (MC) = 0

CALL NEXT (J1, J2, DESC, CROSS, LEVEL)

IF (DESC) GO TO 200
IF (LEVEL, J1) = -1

FOR FIRST CHOICE OF NEXT ITEM TYPE, SELECT ONE MOST IN ARREARS RELATIVE TO

UNIFORM SCHEDULING

PAT = 2,
TEST = .FAIL.
DO 104 PFF=1,1

IF (SCHF(LEVEL, PFF), EQ. (-1)) GO TO 104

104 NPAT(LEVEL, PFF) = -1
GO TO 104

IF (LEVEL, PFF) = -1

PAT=P.

CONTINUE

IF (.NOT. TEST) GO TO 117
DO 114 PFF=1,1

115 CONTINUE

GO TO 114

THE LAST TWO CHAINS IN THE PARTIAL SCHEDULE FORM A CLOSED PAIR. BOTH ARE
C REMOVED FROM THE SCHEDULE, BUT THE PART IS CHosen AS CANDIDATE FOR A NEW 
C CONTINUATION SINCE IT MIGHT EITHER BE FEASIBLE BY ITSELF OR CAN BE CROSSED 
C WITH ANOTHER CHAIN NOT YET ELIMINATED 
C
C CROSS=.TRUE.
C CALL BACKT (CROSS,LEVEL)
C LEVEL=LEVEL-1
C J1=PREV (LEVEL)
C GO TO 110
C
C SIMPLE BACKTRACK STEP: ONLY ONE CHAIN IS REMOVED 
C CROSS=.FALSE.
C CALL BACKT(CROSS,LEVEL)
C GO TO 112
C
C CHAIN J1, OR, IF CROSS=TRUE, THE PAIR OF CROSSED CHAINS (J1,J2), FORM A 
C FEASIBLE CONTINUATION OF THE PARTIAL SCHEDULE 
C
C 210 IF (CROSS) GO TO 202
C PPREV (LEVEL)=J1
C XPREV (LEVEL)=0
C LEVEL=LEVEL+1
C GO TO 204
C 202 PPREV (LEVEL)=J1
C XPREV (LEVEL)=J2
C YPREV (LEVEL)=1
C XPREV (LEVEL+1)=1
C LEVEL=LEVEL+2
C CALL PSConv (J1,J2,CROSS,LEVEL)
C
C TEST IF OPTIMAL SOLUTION HAS BEEN FOUND 
C IF (LEVEL.GT.N) GO TO 500
C NCOUNT=NCOUNT+1
C IF (NCOUNT.GT.MAXMOD) GO TO 3000
C
C SEARCH IS NOT COMPLETE. BEFORE ATTEMPTING THE DETAILED CALCULATIONS OF 
C Searching FOR A FEASIBLE CONTINUATION CHAIN, CHECK WHETHER IN THE CURRENT 
C PARTIAL SCHEDULE, EACH MACHINE CAN BE REACHED BY AT LEAST ONE PART TYPE 
C (NOT YET ONLY SCHEDULED), FROM THE IMMEDIATELY PRECEDING MACHINE FOR THAT 
C PART, WITHOUT EXCEEDING THE REMAINING AVAILABLE IDLE TIME. IF NOT, THE 
C PARTIAL SCHEDULE CANNOT HAVE A SATURATED CONTINUATION; BACKTRACK IN SEARCH 
C TREE (STATE 110)
C
C DO 210 MCF=2,K
C IF ((CPART (MCN).EQ.MAXPBT(MCN)).OR.(TLAST (MCN).EQ.0)) GO TO 210
C DO 212 PPT=1,K
C IF (CJOBIM (PPT,MCN).EQ.0).OR.(REMJOB (PPT).EQ.0) GO TO 212
C MC=MCN-1
C 215 TMP=CJOBIM (PPT,MCN)
C IF (TMP.GT.0) GO TO 214
SUBROUTINE TEXT(J1,J2,TEST,CROSS,LEVEL)

C SHUFFLE THE CURRENT PARTIAL PERMUTATION (CPP):
C
C SHUFFLE TESTS WHICH THE CURRENT PARTIAL PERMUTATION (CPP):
C
C WHERE J1 AND J2 FORM A 'CROSSED PAIR', OR
C
C WHERE J1 IS ADDED TO CPP BY ITSELF, GIVEN J1, THE ROUTINE FIRST ATTEMPTS
C TO FIND J2 SUCH THAT (A) IS FEASIBLE, IF SUCCESSFUL, IT RETURNS TEST=.F.,
C CROSS=.T., J2 AND THE APPROPRIATE STARTING AND FINISHING TIMES OF CHAINS
C J1 AND J2 ON THE VARIOUS MACHINES IN ARRAYS STRT1,FIN1,STRT2,FIN2.
C IF NO J2 EXISTS FOR WHICH (A) IS FEASIBLE, THE ROUTINE TESTS WHETHER
C (B) IS FEASIBLE; IF SO, IT RETURNS TEST=.T., CROSS=.F. AND THE STARTING
C AND FINISHING TIMES OF CHAIN J1 IN STRT1,FIN1. IF NEITHER (A) NOR (B)
C ARE FEASIBLE, ROUTINE RETURNS TEST=.F.

C

IMPLICIT INTEGRAL (A-Z)
LOGICAL TEST,CROSS
COMMON/IF,H,JOB,(9),MCJOB(9),MPART(9),MAKPEA(9),SLACK(9),
1 MACH(9),CPREP(9),OTAKE(9,9),ETAKE(9,9),JIBEST(9,9),
2 JTOPEN(9,9),FIN1(9),FIN2(9),STRT1(9),STRT2(9),STRT(99,9),FINISH(99,9),
3 SRTF(99,9),SFIN(99,9),SLACK(99,9),
4 SSTART(99,9),SFIN(99,9),SFIN(99,9),SFIN(99,9),
5 STRT(9),SLACK(9)
DOWNS=FOR GAP(9)

C

C DETERMINE IF THERE IS A CHAIN J2 WHICH OVERTAKES J1
C
C IF(CREP(J1,9),EQ.*)GO TO 130
C J1=CPREP(J1)
100 DO 120 JX=1,JX
C J2=OTAKE(J1,JX)
C IF(SORT(LEV1+1,J2),EQ.*,(-1))GO TO 120
C IF(SRTF(J2),GT.0)GO TO 200
C IF(SORT(LEV1+1,J2)=(-1))C
120 CONTINUE
C
C CHAIN J1 CANNOT BE PAIRED WITH AN OVERTAKING CHAIN; TEST WHETHER J1 CAN BE
C JOINED TO PARTIAL SCHEDULE BY ITSELF
C
C
C 130 J2=*
C
C 200 DO 101 T=1,K
C STRT1(I)=0
C FIN1(I)=0
C FIN2(I)=0
C
C 101 GAP(T)=?
C STRT1(I)=SSTR1(I),JOBST(J1,I)
C IF(J2,9,9)GO TO 197
GO TO 210
212 CONTINUE
   GO TO 111
213 CONTINUE
   GO TO 112
C
C OUTPUT ROUTINE
C
1000 WRITE (NWRITE, 919) NCOUNT
919 FORMAT (1HO, 'NO SATURATED SCHEDULE EXISTS',/1H, 'NODERS WERE GEN-
   EERATED')
   
1001 WRITE (NWRITE, 920) BIGLEV, (SPER (J), J=1, BIGLEV)
920 FORMAT (1HO, 'LONGEST SATURATED PARTIAL SCHEDULE HAS',/1H, 'I5', ' ITEMS:',/1H, 1012)
   N1=BIGLEV
   DO 1002 J=1, N1
   SPER*(J)=SPER(J)
   DO 1002 I=1, K
   STCAP(J, I)=STCAP(J, I)
   NCHOF(J, I)=NCHOF(J, I)
   1002  CONTINUE

C
C 3000 WRITE (NWRITE, 949) NCOUNT
949 FORMAT (1HO, ' SEARCH EXCEEDS MAXIMUM NODE COUNT OF',/1H, 'I5',)
   LEVEL=LEVEL-1
   IF (BIGLEV.GT.LEVEL) GO TO 1001
   WRITE (NWRITE, 920) LEVEL, (SPER*(J), J=1, LEVEL)
   N1=LEVEL
   GO TO 2010
C
C 2000 N=NN
   WRITE (NWRITE, 922) NCOUNT, (SPER*(J), J=1, N)
922 FORMAT (1HO, 'FIRST OPTIMAL SCHEDULE FOUND AFTER GENERATING',/1H, 'N',
   ' NODES:','/1H0, 4013)
C
2010 N=LAST=NFINISH(N1, K)
   DO 2012 MC=1, K
   NPAFT (TYPE)=0
   DO 2012 I=1, NLAST
   NSCHOF (M, MC)=0
   2012 CONTINUE

C
2014 N=NFINISH (M, K)
   LIM=NCPART(M)
   DO 2016 PPT=1, LIM
   TYPE=NCHOF (PPT, MC)
   NPAFT (TYPE)=NPAFT (TYPE)+1
   2016 CONTINUE

**LISTING 2.2**

```
X = 1.0
PRINT "TYPE": PACT ("TYPE")
MIN = MAX (PT, MC) + 1
MAX = FINISH (PP, MC)
DO 19 X = MIN, MAX
2016 NSCHED (TIM, MC) = X
IF (MC.EQ.1) GO TO 2016
IF (BUFF (MC, MC) .NE. 0) GO TO 2016
MAX = START (PT, MC)
WX = MAX - BUFF (PT, MC) + 1
DO 2019 TIM = MIN, MAX
2019 CONTINUE
2016 CONTINUE
```

*DO 2030 I = 1, K*

```
RAND (I) = 1
WRITE (WRITE, 930) (RAN (I), I = 1, K)
930 FORMAT (100, 12X, "TIME", 8 (7X, "MC", I = 1, K))
WRITE (WRITE, 932)
932 FORMAT (100)
DO 2032 I = 1, NLAST
2032 WRITE (WRITE, 934) I, (NBUFF (I, J), NSCHED (I, J), J = 1, K)
934 FORMAT (100, 12X, 3, 3X, 8 (3X, I = 1, I = 1))
GO TO 1
END
```
11 DELUSER 2.0  NEXT  DATE = 79017  15/05/17

C

DO 999 MCM=2,K
   NMC="CPART"(MCM)
   TOB1=JOB1(MCM)
   IF(J2,GT,0)GO TO 201
   IF(TJCB1)999,999,203
201 TJ032=JOB1(J2,MCM)
   IF(TJ031,GT,0)GO TO 202
   IF(TJ032,EQ,0)GO TO 999
C
C ONLY CHAIN J2 VISITS MACHINE MCM
C
   IF(TLAST(MCM),EQ,0)GO TO 204
   GAP(MCM)=FIN2(M2L)+TIME(M2L,MCM)-TLAST(MCM)
   IF(GAP(MCM),GT,0)GO TO 206
   DIFF=LAST(GAP(MCM))-TLAST(MCM)
   IF(DIFF,LT,0)GO TO 208
   IF(GAP(MCM,MCM),E0,0)GO TO 210
   IF(DIFF,EQ,0)GO TO 208
212 STT2(1)=STT2(1)+DIFF
   GAP(1)=GAP(1)+DIFF
   GO TO 214
215 STT2(1)=STT2(1)+DIFF
   GAP(1)=GAP(1)+DIFF
214 IF(GAP(1)-SLACK(1))199,199,290
C
226 IF (GAP(MCM).EQ.,SLACK(MCM)) GO TO 290
228 STT2(MCM)=FIN2(M2L)+TIME(M2L,MCM)
   GO TO 216
229 STT2(MCM)=TLAST(MCM)
216 FIN2(MCM)=STT2(MCM)+TJ032
   M2L=MCM
   GO TO 999
C
232 IF(TJ032,GT,0)GO TO 230
C
C ONLY CHAIN J1 VISITS MCM
C
233 IF(TLAST(MCM),EQ,0)GO TO 224
   GAP(MCM)=FIN1(M2L)+TIME(M2L,MCM)-TLAST(MCM)
   IF(GAP(MCM),GT,0)GO TO 226
   DIFF=LAST(GAP(MCM))-TLAST(MCM)
   IF(DIFF,LT,0)GO TO 228
   IF(DIFF,EQ,0)GO TO 230
   IF(TJ031,EQ,0)GO TO 229
232 STT1(1)=STT1(1)+DIFF
   GAP(1)=GAP(1)+DIFF
GO TO 234

231 \text{STEP1}(1) = \text{STEP1}(1) + \text{DIFF} \text{H}
\text{GAP}(1) = \text{GAP}(1) + \text{DIFF}

C

234 \text{IF} (\text{GAP}(1) - \text{SLACK}(1)) = 190, 191, 200 \text{GO TO} 290

C

226 \text{IF} (\text{GAP}(1) - \text{SLACK}(1)) = 190, 191, 200 \text{GO TO} 290

224 \text{STEP1}(1) = \text{FIN1}(1) \text{H} + \text{TIME}(1, \text{MCN}) \text{GO TO} 236

236 \text{STEP1}(1) = \text{LAST}(\text{MCN}) \text{H} + \text{JOB1} \text{GO TO} 290

C

C \text{CHAINS J1 AND J2 ARE BOTH PROCESSED ON MACHINE MCN}

C

300 \text{STEP1} (\text{MCN}) = \text{FIN1} (\text{MCN}) + \text{TIME} (\text{MCN}) \text{H}
\text{FIN1} (\text{MCN}) = \text{STEP1} (\text{MCN}) + \text{JOB1}
\text{STEP2} (\text{MCN}) = \text{FIN2} (\text{MCN}) + \text{TIME} (\text{MCN}) \text{H}
\text{FIN2} (\text{MCN}) = \text{STEP2} (\text{MCN}) + \text{JOB2}

C

\text{IF} (\text{STEP2} (\text{MCN}) \neq \text{STEP1} (\text{MCN})) \text{GO TO} 400

C

C \text{CHAIN J2 RECEIVED MACHINES MCN BEFORE J1}

C

\text{FIN2} (\text{MCN}) = \text{LAST} (\text{MCN}) \text{H}
\text{FIN2} (\text{MCN}) = \text{TIME} (\text{MCN}) \text{H}
\text{FIN2} (\text{MCN}) = \text{TIME} (\text{MCN}) \text{H}
\text{FIN2} (\text{MCN}) = \text{TIME} (\text{MCN}) \text{H}

C

310 \text{IF} (\text{STEP2} (\text{MCN}) \neq \text{STEP1} (\text{MCN})) \text{GO TO} 302

C

C \text{JOBS J2 AND J1 ARRIVE AT MCN BEFORE TLAST(MCN). SINCE ONLY ONE BUFFER}

C \text{POSITION IS AVAILABLE, LOADING OF J1 MUST BE DELAYED.}

C

\text{SHIFT} = \text{LAST} (\text{MCN}) - \text{STEP1} (\text{MCN})
\text{STEP1} (1) = \text{STEP1} (1) + \text{SHIFT}
\text{GAP}(1) = \text{GAP}(1) + \text{SHIFT}
\text{IF} (\text{GAP}(1) - \text{SLACK}(1)) = 190, 191, 290 \text{GO TO} 314

C

312 \text{IF} (\text{STEP2} (\text{MCN}) \neq \text{STEP1} (\text{MCN})) \text{GO TO} 312

C

C \text{J2 ARRIVES BEFORE, AND J1 AT OR AFTER TLAST(MCN)}

C

\text{DIFF} = \text{LAST} (\text{MCN}) - \text{STEP2} (\text{MCN}) - \text{LAST} (\text{MCN})
\text{IF} (\text{DIFF} \neq 0) \text{GO TO} 314
\text{IF} (\text{DIFF} \neq 0) \text{GO TO} 210
\text{IF} (\text{DIFF} \neq 0) \text{GO TO} 212
\text{DIFF} = \text{STEP2} (\text{MCN}) - \text{LAST} (\text{MCN})
\text{FIN2} (\text{MCN}) = \text{LAST} (\text{MCN}) + \text{JOB2}

C

C \text{BOTH J2 AND J1 ARRIVE AFTER TLAST}

C

312 \text{DIFF} = \text{STEP1} (\text{MCN}) - \text{FIN2} (\text{MCN})
\text{IF} (\text{DIFF} \neq 0) \text{GO TO} 314
\text{IF} (\text{DIFF} \neq 0) \text{GO TO} 314
\text{FIN1} (\text{MCN}) = \text{STEP1} (\text{MCN}) + \text{JOB1}
\text{FIN1} (\text{MCN}) = \text{STEP1} (\text{MCN}) + \text{JOB1}

314 \text{FIN} (\text{LAST} (\text{MCN}) + \text{ST.}) \text{GO TO} 318
L, 7

C: MXT DA .T 79017 5/06/17

3 r l

(U'I(1) = Dr

V 0 To

318 ; - P (. (N)

C

319 IF (GAP(MCN) + SLACK(MCN)) GO TO 290

M1 = MCN

M2 = MCN

GO TO 999

C

400 IF (STRT2(MCN) + SLACK(MCN)) GO TO 401

STRT2(1) = SHIFT

GAP(1) = GAP(1) + SHIFT

IF (GAP(1) - SLACK(1)) = 199, 199, 290

C CHAIN J1 REACHES MACHINE MCN BEFORE J2

C

401 IF (STRT2(MCN) + SLACK(MCN)) GO TO 402

C BOTH J1 AND J2 ARRIVE AT MACHINE MCN BEFORE TLAST(MCN). SINCE ONLY ONE BUFFER POSITION IS AVAILABLE, LOADING OF J2 MUST BE DELAYED.

C

SHIFT = TLAST(MCN) - STRT2(MCN)

STRT2(1) = SHIFT

GAP(1) = GAP(1) + SHIFT

IF (GAP(1) - SLACK(1)) = 199, 199, 290

C

432 IF (STRT1(MCN) + SLACK(MCN)) GO TO 412

C J1 ARRIVES BEORE, AND J2 AT OR AFTER TLAST(MCN)

C

DIFF = TLAST(MCN) - TLAST(MCN)

IF (DIFF < 0) GO TO 404

IF (DIFF > 0) GO TO 230

IF (DIFF = 0) GO TO 212

404 STRT1 = TLAST(MCN)

FIN1(MCN) = TLAST(MCN) + TJOB1

C BOTH J1 AND J2 ARRIVE AFTER TLAST(MCN)

C

412 DIFF = STRT2(MCN) - FIN1(MCN)

IF (DIFF > 0) GO TO 414

STRT2(MCN) = FIN1(MCN)

FIN2(MCN) = STRT2(MCN) + TJOB2

414 IF (TLAST(MCN) - STRT1) GO TO 418

GAP(MCN) = DIFF

GO TO 419

418 GAP(MCN) = STRT2(MCN) - TLAST(MCN) - TJOB1

C

419 IF (GAP(MCN) + SLACK(MCN)) GO TO 290

M1 = MCN

M2 = MCN

999 CONTINUE

C
TEST = TRUE.
RETURN
290 IF (LEVEL = 0) GO TO 292
"OP (LEVEL + 1, 12) = (-1)
GO TO 110
292 TEST = FALSE.
RETURN
END
INTEGER CROSS
LOGICAL CROSS
COMMON/FF,J,M,S,MACH(9),CPART(9),MAXPT(9),SLACK(9),
1 MACH(9),CPART(9),MAXPT(9),SLACK(9),JOBT1N(9,9),TPTIME(9,9),
2 JOBT1N(9,9),FIN1(9),FIN2(9),START(9,9),START(9,9),FINISH(9,9),
3 JOBPT(9,9),DBPT(9,9),DPPT(9,9),XPPT(9,9),NODE(9,9),LAST(9),
4 NCPD(9,9),SNFF(9,9),SSST(9,9),SFIN(9,9),SCPTD(9,9),
5 SBT(9),SFT(R99)
C
REMJOB(J1)=REMJOB(J1)-1
IF(CROSS) REMJOB(J2)=REMJOB(J2)-1
NO 2 J=1,2
IF(REMJOB(J1).GE.0)GOTO 4
NODE(LEVEL,J)=-1
NODE(LEVEL+1,J)=(-1)
GO TO 2
4 NODE(LEVEL,J)=0
NODE(LEVEL+1,J)=0
2 CONTINUE
C
GO TO 100
C
ONLY CHAIN J1 IS ADDED TO PARTIAL SCHEDULE
C
NT=MACH(J1)
MC=1
NO 6 I=2,NT
1 M=MC
MC=MCPRT(MC)
DIF=LAST(MC)-FIN1(M1L)-TPTIME(M1L,MC)
MC=MCPRT(MC)+1
IF(DIF.GT.0)GOTO 8
DBPF(NK,MC)=0
GO TO 6
8 DBPF(NK,MC)=DIF
2 CONTINUE
NO 17 I=1,NT
MC=MCPRT(MC)
MC=MCPRT(MC)+1
MC=MCPRT(MC)
MC=MACH(NK,MC)=J1
FINISH(NK,MC)=FIN1(MC)
C BOTH J1 AND J2 ARE ADDED TO PARTIAL SCHEDULE. LOOP ENDING AT STMT 110
C UPDATE PUFFER OCCUPANCY
C
100 MIL=1
M2L=1
DO 110 MC=2,K
MC=MCART (Mc) +1
IF (JOBT11(J1,MC).GT.0) GO TO 102
IF (JOBT11(J2,MC).EQ.0) GO TO 110
DIFF(TILAST(MC)-FIN1(M1L)-TPTIME(M1L,MC)
M1L=MC
GO TO 109
102 IF (JOBT11(J2,MC).GT.0) GO TO 104
DIFF=TILAST(MC)-FIN1(M1L)-TPTIME(M1L,MC)
M1L=MC
104 IF (STF(T1(MC),J.1,STT(J1(MC))) GO TO 106
DIFF=TILAST(MC)-FIN1(M1L)-TPTIME(M1L,MC)
DIFF=FIN1(MC)-FIN2(M2L)-TPTIME(M2L,MC)
GO TO 107
106 DIFF=TILAST(MC)-FIN2(M2L)-TPTIME(M2L,MC)
D1FF=FIN1(MC)-FIN2(M2L)-TPTIME(M2L,MC)
107 M1L=MC
M2L=MC
IF (DIFF1.GT.0) GO TO 109
DIFF(MC+1,MC)=0
GO TO 109
109 DIFF(MC+1,MC)=DIFF1
108 IF (DIFF.GT.0.0) GO TO 111
DIFF(MC,MC)=0
GO TO 111
111 DIFF(MC,MC)=DIFF
110 CONTINUE
C
DO 200 MC=1,K
MC=MCART (MC) +1
IF (JOBT11(J1,MC).GT.0) GO TO 210
IF (JOBT11(J2,MC).EQ.0) GO TO 200
C
JOBT11(J2,MC)
MCART(MC)=MC
MCRT11(MC,MC)=J2
START(MC,MC)=STT2(MC)
FIN1SH(MC,MC)=FIN2(MC)
SAP=ST1T2(MC)-TILAST(MC)
TLAST (MC) = FIN2 (MC) - TLAST (MC)
LAST (MC) = TJOB
IF (MC = NMC) GO TO 206
C
TJOB = JOBTIM (J2, MC)
SCHFF (MC, MC) = J1
SCHRD (NMC, MC) = J2
START (NMC, MC) = FIN1 (MC) - JOBTIM (J1, MC)
FINISH (NMC, MC) = FIN1 (MC)
START (NMC, MC) = SHFT (MC)
FINISH (NMC, MC) = FIN2 (MC)
GAP = FIN1 (MC) - TLAST (MC) - TJOB
LAST (MC) = FIN2 (MC)
LAST (MC) = TJOB
IF (MC = NMC) GO TO 206
C
SCHFF (MC, MC) = J1
SCHRD (NMC, MC) = J2
START (NMC, MC) = SHFT (MC)
FINISH (NMC, MC) = FIN2 (MC)
GAP = FIN1 (MC) - TLAST (MC) - TJOB
LAST (MC) = FIN1 (MC)
LAST (MC) = TJOB
IF (MC = NMC) GO TO 206
C
SLACK (MC) = SLACK (MC) - GAP
C
CONTINUE
RETURN
END
C SUBROUTINE BACKL (CROSS, LEVEL)...
C THIS ROUTINE IS CALLED WHEN SUBROUTINE 'NEXT' RETURNS TEST = .F., IS THEN
C C PP <PREF (1) ..., PPRN (LEVEL)> HAS NO FEASIBLE SATISFIED CONTINUATION.
C IF LEVEL = 0, CPP IS STORED IN ARRAYS SART, SPPF, SBUFF, SSRT, SFIN.
C SART AND LEVEL 1 REPLACES THE PREVIOUS VALUE OF BIGLEV.
C IF CROSS = .F., ROUTINE DELETES PPRN (LEVEL) FROM THE CPP AND UPDATES
C ARRAYS MCPART, SART, SLAST, SLACK. IF CROSS = .T., ROUTINE DELETES
C BOTH PPRN (LEVEL) AND PPRN (LEVEL - 1) AND UPDATES THE SAME ARRAYS.
C...

C IMPLICIT INTEGER (A-Z)
LOGICAL CROS
COMMON /KXX,BIGLEV, REMJOB (9), MCPART (9), MAXPET (9), SLACK (9),
1 MACH (9), CRSP (5), OTAKE (9, 9), ISACH (9, 9), JOBFIN (9, 9), TTIME (9, 9),
2 JOBTE (9, 9), FIN1 (9), TIN2 (9), START (9), START2 (9), START (99, 9), FINISH (99, 9),
3 SCHED (99, 9), STUFF (99, 9), SART (99, 9), SFIN (99, 9), SCHED (99, 9),
4 SART (99, 9), SPPR (99)
5 SPART (9), SNUM (99)
C
IF (LEVEL .LE. BIGLEV) GO TO 10
BIGLEV=LEVEL
DO 1 MC=1, K
1 SPART (MC) = MCPART (MC)
DO 2 I=1, LEVEL
SPPF (I) = PPEPM (I)
DO 2 J = 1, K
SUFF (I, J) = SUFF (I, J)
SSTP (I, J) = SART (I, J)
STRT (I, J) = FINISH (I, J)
SCHED (I, J) = SCHED (I, J)
2 CONTINUE
10 J2=PPRM (LEVEL)
REMJOB (12) = REMJOB (12) + 1
IF (CROSS) GO TO 20
C ONLY CHAIN J2 IS DELETED FROM SCHEDULE
C
MACH (J12)
DO 12 J1 = 1, NT
MC=MACH (J12, J1)
MCP=MCPART (MC)
MNC=MNC-1
MCPRT (MC) = MNC
IF (MNC .LT. 0) GO TO 14
SLAST (MC) =.
PLAST (MC) =.
GO TO 12
14 SLACK (MC) = SLACK (MC) + START (MNC, MC) - FINISH (MNC, MC)
PLAST (MC) = PLAST (MC) + START (MNC, MC)
12 CONTINUE
CALL
C PAIR OF CROSSED CHAINS ARE DELETED FROM SCHEDULE
C
20 J1=PPEP.PLEVEL-1
20 PRED (J1) = PRED (J1) + 1
C
DO 22 MC=1,K
22 IF (JOBTM (J1, MC) .GT. 7) GO TO 24
IF (JOBTM (J2, MC)) 22, 24, 26
24 IF (JOCTLY (J2, MC)) 26, 26, 28
26 NMC=CPART (MC)
NMC = NMC - 1
MCPAR (MC) = NMC
IF (NMC .GT. 0) GO TO 30
TLAST (MC) = 0
WLAST (MC) = 0
GO TO 22
30 SLACK (MC) = SLACK (MC) + START (NMC, MC) - FINISH (NMCL, MC)
TLAST (MC) = FINISH (NMCL, MC)
WLAST (MC) = TLAST (MC) - START (NMCL, MC)
GO TO 22
C
26 NMC = MCPAR (MC)
NMC = NMC - 2
MCPAR (MC) = NMC
IF (NMC .GT. 0) GO TO 32
SLACK (MC) = SLACK (MC) + START (NMC, MC) - FINISH (NMCL, MC)
TLAST (MC) = FINISH (NMCL, MC)
WLAST (MC) = TLAST (MC) - START (NMCL, MC)
GO TO 22
32 SLACK (MC) = SLACK (MC) + START (NMC, MC) - FINISH (NMCL, MC)
TLAST (MC) = 0
WLAST (MC) = 0
GO TO 22
22 CONTINUE
RETURN
END
A3: Sample Problems and Printouts of Solutions
**Example 1**

**Problem Data:**

**No. of Machines:** 6

**Travel Times Between Machines:**

<table>
<thead>
<tr>
<th>FROM</th>
<th>TO 1</th>
<th>TO 2</th>
<th>TO 3</th>
<th>TO 4</th>
<th>TO 5</th>
<th>TO 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>17</td>
<td>12</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**No. of Item Types:** 5

**Item Type** | **No. Req'd** | **Proc. Times on MC**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4 6 0 0 10 2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4 7 5 4 0 2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2 0 10 0 1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2 4 7 0 4 1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2 0 5 7 6 1</td>
</tr>
</tbody>
</table>

**Period = 42**  
**Slack (J) = 16 12 0 4 0 29**

**Item Type, CRSPR, OTAKE:**

<table>
<thead>
<tr>
<th>Item Type</th>
<th>CRSPR</th>
<th>OTAKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3 5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1 3 5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Reach(MC, MCN):**

<table>
<thead>
<tr>
<th>7 12 24 32 45</th>
<th>10 25 23 38</th>
<th>12 19 26</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 14</td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

**First Optimal Schedule Found After Generating 20 Nodes:**

1 5 3 1 4 5 2 5 2

**Time** | **MC1** | **MC2** | **MC3** | **MC4** | **MC5** | **MC6**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>3</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>4</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 501</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>6</td>
<td>0 501</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>7</td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>8</td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>9</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>10</td>
<td>0 0</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>11</td>
<td>0 102</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>12</td>
<td>0 102</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>13</td>
<td>0 102</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>14</td>
<td>0 102</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>15</td>
<td>0 401</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>16</td>
<td>0 401</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>17</td>
<td>0 502</td>
<td>0 0</td>
<td>0 501</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>18</td>
<td>0 502</td>
<td>0 0</td>
<td>0 501</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>19</td>
<td>0 0</td>
<td>0 0</td>
<td>0 301</td>
<td>501 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>
### Problem Data: Example 2

**No of Machines:** 6  
**Travel Times Between Machines:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>17</td>
<td>12</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

**No of Item Types:** 5

<table>
<thead>
<tr>
<th>Item Type</th>
<th>No Req</th>
<th>Proc. Times on MC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

**Period = 119**  
**Slack(J) = 49 46 0 2 1 84**

<table>
<thead>
<tr>
<th>Item Type</th>
<th>CRSPR</th>
<th>OTAKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3 5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1 3 5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Reach(MC, JMC):**

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>12</th>
<th>24</th>
<th>32</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td>25</td>
<td>23</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**First Optimal Schedule Found After Generating 370 Nodes:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>5</th>
<th>2</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>3</th>
<th>1</th>
<th>5</th>
<th>1</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td></td>
<td>MC1</td>
<td>MC2</td>
<td>MC3</td>
<td>MC4</td>
<td>MC5</td>
<td>MC6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>102</td>
<td>101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0 401</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0 401</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0 501</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>0 501</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>0 201</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0 201</td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>0 201</td>
<td>0 102</td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>0 201</td>
<td>0 102</td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>0 201</td>
<td>0 102</td>
<td>0 301</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>275</td>
<td>0</td>
<td>0</td>
<td>0 105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example 3

**Problem Data:**

- **No of Machines:** 6
- **Travel Times Between Machines:**
<table>
<thead>
<tr>
<th>From</th>
<th>To 1</th>
<th>To 2</th>
<th>To 3</th>
<th>To 4</th>
<th>To 5</th>
<th>To 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>17</td>
<td>12</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>12</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

- **No of Item Types:** 5

<table>
<thead>
<tr>
<th>Item Type</th>
<th>Proc. Times on MC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Period = 117**

**Slack (J):** 57 44 48 0 5 82

**Item Type Cycles Order:**

- Item Type 1: 0 0
- Item Type 2: 3 5
- Item Type 3: 2 4 5
- Item Type 4: 3 1 3 5
- Item Type 5: 0 0

**Reach (MC, MNC):**

- 7 12 17 33 41
- 10 25 23 35
- 12 14 25
- 13 14
- 6

**First Optimal Schedule Found After Generating 26961 Nodes:**

<table>
<thead>
<tr>
<th>Time</th>
<th>MC1</th>
<th>MC2</th>
<th>MC3</th>
<th>MC4</th>
<th>MC5</th>
<th>MC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>3</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>4</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 101</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>6</td>
<td>0 3 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>7</td>
<td>0 4 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>8</td>
<td>0 4 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>9</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>10</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>11</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>12</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>13</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>14</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>15</td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>16</td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>17</td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>18</td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>19</td>
<td>0 102</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td>205</td>
<td>0</td>
<td>0</td>
<td>204</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>196</td>
<td>205</td>
<td>0</td>
<td>0</td>
<td>204</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>197</td>
<td>205</td>
<td>0</td>
<td>0</td>
<td>204</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>198</td>
<td>205</td>
<td>0</td>
<td>0</td>
<td>204</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>199</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td>202</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td>203</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td>204</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td>205</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>510</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>152</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>153</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>155</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>156</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>159</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>160</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>161</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
PROBLEM DATA:  EXAMPLE 4

NO OF MACHINES: 6
TRAVEL TIMES BETWEEN MACHINES:

<table>
<thead>
<tr>
<th>FROM TO 1 2 3 4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 5</td>
</tr>
<tr>
<td>3 10 6</td>
</tr>
<tr>
<td>4 16 12 7</td>
</tr>
<tr>
<td>5 21 17 12 6</td>
</tr>
<tr>
<td>6 25 21 16 10 5</td>
</tr>
</tbody>
</table>

NO OF ITEM TYPES: 5

<table>
<thead>
<tr>
<th>ITEM TYPE</th>
<th>NO REQD</th>
<th>PROC. TIMES ON MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4 6 0 0 10 2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4 20 5 10 0 2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2 0 10 15 0 2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2 10 0 4 1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3 0 5 9 10 1</td>
</tr>
</tbody>
</table>

PERIOD = 143  SLACK (J)  75  5  38  0  25  109

ITEM TYPE CRSP3 OTAKE

<table>
<thead>
<tr>
<th>ITEM TYPE</th>
<th>CRSP3</th>
<th>OTAKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3 5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1 4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1 3 5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

REACH (MC, MCN):

<table>
<thead>
<tr>
<th>REACH</th>
<th>7 12 25 32 47</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>18 23 38</td>
</tr>
<tr>
<td>12</td>
<td>22 31</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
</tr>
</tbody>
</table>

FIRST OPTIMAL SCHEDULE FOUND AFTER GENERATING 8812 NODES:

2 3 1 1 5 4 3 2 5 2 5 2 5 1 5 4 3 2 1 1 5

TIME MC1 MC2 MC3 MC4 MC5 MC6

<table>
<thead>
<tr>
<th>TIME</th>
<th>MC1</th>
<th>MC2</th>
<th>MC3</th>
<th>MC4</th>
<th>MC5</th>
<th>MC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>301</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>301</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>152</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>153</td>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>155</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>156</td>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>506</td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>159</td>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>160</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>161</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>162</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>163</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>164</td>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>165</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>166</td>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>167</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>168</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>169</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>170</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>171</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>172</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>173</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>174</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>175</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>176</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>177</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>178</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>179</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>180</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>181</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>182</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>183</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>184</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>186</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>187</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>188</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>189</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>190</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>191</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>193</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>194</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>195</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>196</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>197</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>198</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>199</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

-66-
PROBLEM DATA:  

EXAMPLE 5

NO OF MACHINES: 6  
NO OF ITEM TYPES: 5

TRAVEL TIMES BETWEEN MACHINES:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>16</td>
<td>12</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>21</td>
<td>17</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>10</td>
</tr>
</tbody>
</table>

ITEM TYPE  NO REQD PROC. TIMES ON MC

<table>
<thead>
<tr>
<th>ITEM TYPE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>5</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

PERIOD = 143  
SLACK (J) = 75 5 38 0 1 109

ITEM TYPE  CRSPR  OTAKE

<table>
<thead>
<tr>
<th>ITEM TYPE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REACH (MC, MCN):  
7 12 25 32 47  
10 38 23 38  
12 22 31  
14 20  
9

NO SATURATED SCHEDULE EXISTS  
1667 NODES WERE GENERATED

LONGEST SATURATED PARTIAL SCHEDULE HAS 18 ITEMS:  
2 1 1 5 5 3 4 2 5 2 1 1 3 1 3 2 5 4

<table>
<thead>
<tr>
<th>TIME</th>
<th>MC1</th>
<th>MC2</th>
<th>MC3</th>
<th>MC4</th>
<th>MC5</th>
<th>MC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
REFERENCES

Interim Reports: ESL-IR-740, ESL-IR-771 (1977)
Laboratory for Information and Decision Systems, M.I.T.

Laboratory for Information and Decision Systems, M.I.T. (1978)


Laboratory for Information and Decision Systems, M.I.T. (1978)

8. R. Hildebrand (C.S. Draper Laboratory, Cambridge, Mass.) Private Communications.


Laboratory for Information and Decision Systems, M.I.T. (1978)