THE SCHEDULING OF N TASKS WITH M OPERATIONS ON TWO PROCESSORS

BY

HENRY BAUER
HAROLD STONE

STAN-CS-70-165
JULY 1970

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

ABSTRACT

The job shop problem is one scheduling problem for which no efficient algorithm exists. That is, no algorithm is known in which the number of computational steps grow algebraically as the problem enlarges. This paper presents a discussion of the problem of scheduling N tasks on two processors when each task consists of three operations. The operations of each task must be performed in order and among the processors. We analyze this problem through four sub-problems. Johnson's scheduling algorithm is generalized to solve two of these sub-problems, and functional equation algorithms are used to solve the remaining two problems. Except for one case, the algorithms are efficient. The exceptional case has been labelled the "core" problem and the difficulties are described.

This research was supported by the U.S. Atomic Energy Commission under contract number AT (04-3)326 PA23 and NSF GJ 687.

The Scheduling of N Tasks with M Operations on Two Processors

by

Henry Bauer and Harold Stone

I. Introduction

The 'job shop problem is one scheduling problem for which no efficient algorithm exists [Conway 1967]. That is, no algorithm is known in which the number of computational steps grow algebraically as the problem enlarges. This paper presents a discussion of the problem of scheduling N tasks on two processors when each task consists of three operations. The operations of each task must be performed in order and among the processors. We analyze this problem through several sub-problems. Johnson's scheduling algorithm [Johnson 1955] is generalized to solve two of these sub-problems, and functional equation algorithms [Lawler 1969] are used to solve the remaining two problems. Except for one case, the algorithms are efficient. The exceptional case has been labelled the "core" problem and the difficulties are described.

This problem has been suggested by several examples in computer science.

1. N tasks exist which alternately require the use of a CPU and some peripheral processor and for which the time required by each processor is known within reasonable tolerance.

2. N tasks exist which are to be prepared (compiled) by one machine for execution by a second machine and the output is

to be processed by the first machine again. The time required for each processor is again known in advance.

The organization of the paper is as follows: Section II discusses relevant results of previous researchers. Section III states the problem of scheduling tasks with three operations on two processors and initiates the discussion of the problem's solution. Sections IV, V, and VI present three sub-problems for which efficient solutions have been found. The "core" problem is discussed in Section VII. Finally, the complete problem solution is outlined, and a summary of the results and an indication of future research directions are given.

II. Historical Results

The major results in the problem are due to S. M. Johnson [Johnson 1955]. Johnson considered the production schedule of N tasks each of which he assumed to have two operations. The first operation is performed on the first machine and the second operation is performed on the second machine. There are only two machines and the second operation may not begin before the first operation is completed. Johnson obtained the following two results.

1. The order of the production sequence on the two machines may be made the same without loss of time.
2. Let tasks i, $i = 1, \ldots, N$ consist of the pair of operations a_i, b_i where a_i, $i = 1, \ldots, N$, are the lengths of the operations to be processed on the first machine and b_i, $i = 1, \ldots, N$ are the lengths of the succeeding operations to be processed on the second machine. An optimal ordering is given by the following rule:

   Item j precedes item $j+1$ if

   $$\min(a_j, b_{j+1}) < \min(a_{j+1}, b_j) .$$

   This ordering is unique except for ties.

Equivalently, result 2 may be stated in other terms for which we require the following definitions.

Definition: The contribution of the i-th task is the difference $b_i - a_i$.

Definition: The Delay, , is the difference between the initiation times of the two operations of a given task.

Intuitively, the contribution of the i-th task represents the effect of the task's assignment on the value of the delay. A positive contribution tends toincreasethe delay for the next task assigned; a negative contribution tends to decrease the delay. Result 2 is equivalent to the following.

2'. Divide the tasks into two groups according to whether their contributions are negative or non-negative. Assign all the non-negative contributing tasks in order of increasing size of a_i's followed by the negative contributing tasks in order of decreasing size of b_i's .

The proof of this assertion can be seen as follows. Let the tasks with positive contribution be indexed for $j = 1,2,\ldots,m$. Then, if arranged by increasing a_k , these tasks satisfy

$$a_j \leq a_{j+1} \wedge a_j \leq b_j \text{ A } a_{j+1} \leq b_{j+1} \text{ A } a_j \leq b_{j+1} \qquad \text{for } 1 \leq j \leq m\text{-}1 .$$

Then

$$\min(a_j, b_{j+1}) = a_j \text{ and } \min(a_{j+1}, b_j) \geq a_j \qquad \text{for } m+1 \leq j \leq N\text{-}1 .$$

Similarly, we obtain

$$\min(a_j, b_{j+1}) = b_{j+1} \quad \text{and} \quad \min(a_{j+1}, b_j) \geq b_{j+1}$$

for the negative contributing tasks which again is Johnson's condition. At the dividing line between the negative and non-negative contributing tasks

$$a_m \leq b_m \wedge a_{m+1} > b_{m+1} .$$

Therefore, we obtain the Johnson condition

$$\min(a_m, b_{m+1}) \leq \min(b_m, b_{m+1}) \leq \min(b_m, a_{m+1}) \quad .$$

Johnson generalized his first result for N tasks, each with M operations, $M > 2$.

1'. Consider N tasks each with M operations to be processed respectively on M machines, $1,2,\ldots,M$. That is, the first operation of each task must be done on machine one, the second operation on machine two, and the k-th operation on machine k. To minimize the maximum flow time it is sufficient to, consider only schedules in which the production sequence is the same on machines one and two, and in which the production sequence is the same on machines M-l and M .

The third Johnson result is for a special case in the N task, 3 machine problem.

3. Consider N tasks each with 3 operations a_k, b_k, c_k , $k = 1,2,\ldots,N$ to be processed in order on machines 1, 2, and 3, respectively. Assume that

$$\min a_i \geq \max b_j \ .$$

Task i precedes task j if

$$\min(a_i + b_i, c_j + b_j) < \min(a_j + b_j, c_i + b_i) \ .$$

In the general job shop problem for M machines and N tasks, the only complete solution that is currently known for which the computational complexity is algebraic rather than exponential in N

is for $M = 2$. In an extension of Johnson% results, Jackson [Jackson 1956] showed that if

{A} is the set of jobs with only one operation to be performed on machine one,

{B} is the set of jobs with only one operation to be performed on machine two,

{AB} is the set of jobs which have two operations, the first to be performed on machine one and the second on machine two,

and {BA} is the set of jobs which have two operations, the first to be performed on machine two and the second on machine one,

then simply determine the sequence of tasks in {AB} and {BA} by Johnson's rule 2 and, using these orderings, assign the tasks to machine one and machine two as follows:

machine one: tasks in {AB}, followed by tasks in (A], followed by tasks in {BA}

machine two: tasks in {BA}, followed by tasks in (B), followed by tasks in {AB}

where the order of tasks in {A} and {B} does not matter.

III. The 3-stage Scheduling Problem

In computer scheduling, it is sometimes advantageous to queue a group of tasks (programs) which use a common facility (compiler) which is serially reuseable (core resident). In this case, intermixing the job queue with dissimilar tasks would cause set up delays of disproportionate length. Similarly, the processing (execution) of these tasks on a second machine may also require special facilities (run time administration) which are also serially reuseable. In addition, the completion of the tasks may be processed by the first machine with certain advantages of grouping.

The results of this paper concern a special case of the two machine job shop problem for N tasks with exactly three operations which reflects the situation stated above. The general problem is restricted in the following three ways:

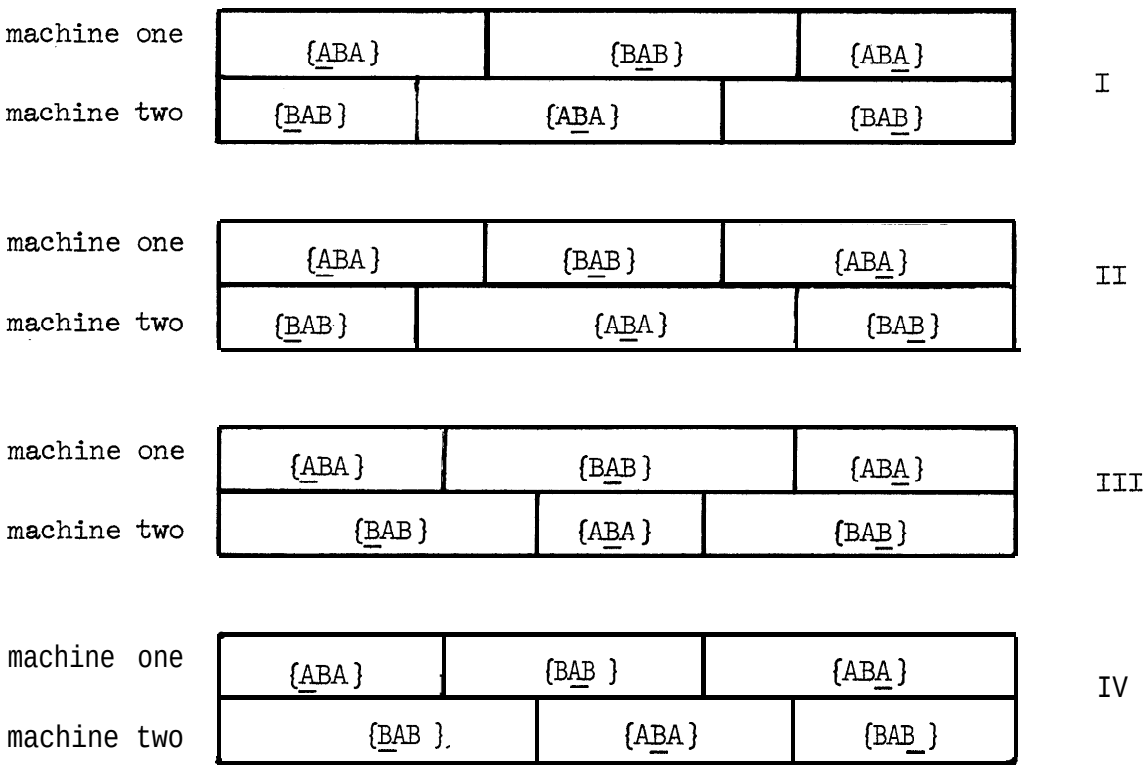
1. The first and third operations of each task must be performed on one machine and the second operation must be performed on the other machine. Hence in the notation of the previous section the tasks may be divided into two sets: {ABA} and {BAB} . (Note that when fewer than three operations exist in the cases {A] , {B} , {AB} , {BA} , an arbitrary extension to three operation tasks may be made. However, the choice of the extension may change the resulting assignment.)
2. The form of the solution is restricted as follows for machine one and machine two.

machine one: The initial operations of the set (ABA] , followed by the second operations of the set {BAB} , and followed by the third operation of the set {ABA} .

machine two: The initial operations of the set {BAB} , followed by the second operations of the set {ABA} , and followed by the third operation of the set {BAB} .

3. No idle time is allowed.

These three conditions restrict the solution to one of the four forms illustrated in the Gantt charts below. In these charts each segment is labeled by the set of tasks which may be assigned in the segment. The underline indicates the operation which is to be performed. For example, {ABA} indicates that the third operations of the tasks in the set {ABA} are processed.



By the symmetry of machine one with respect to machine two, Gantt Charts II and III are similar, and Gantt Charts I and IV are similar. The discussion will be limited to forms I and II.

At the beginning of this section it was noted that the form of our problem was chosen to reflect certain restraints found in some computer scheduling problems. It should also be noted that the solution to the problem as restricted by these conditions does not necessarily give an optimal solution for the general scheduling problem. Below are examples in which no solution of the form described in condition 1 may be found which will also satisfy conditions 2 and 3.

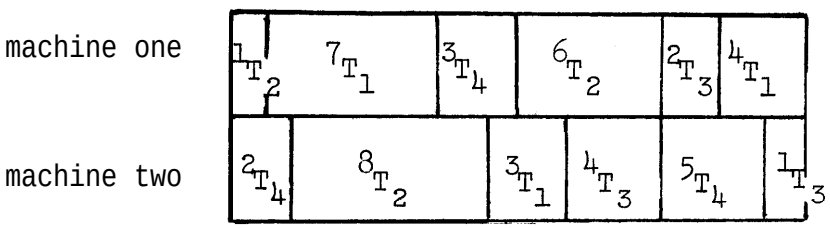
Example 111.1.

Given the four tasks

$$\{ABA\} = \{T_1 = (7,3,4)\ ,\ T_2 = (1,8,6)\}$$

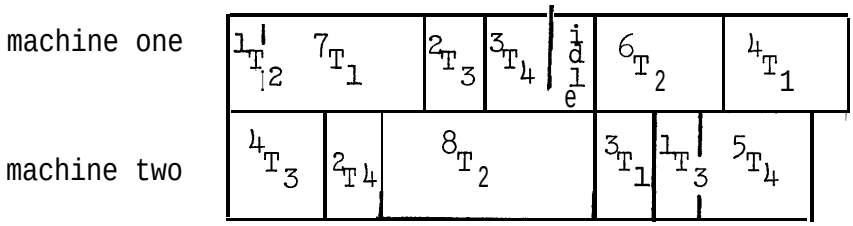
$$\{BAB\} = \{T_3 = (4,2,1)\ ,\ T_4 = (2,3,5)\}$$

an assignment may be found which contains no idle time and is completed in 23 time units.



III.A

However, condition 2 may not be maintained without violating condition 3. The best solution that satisfies condition 2 is an assignment of length 24 as shown below.



III.B

Example 111.2:

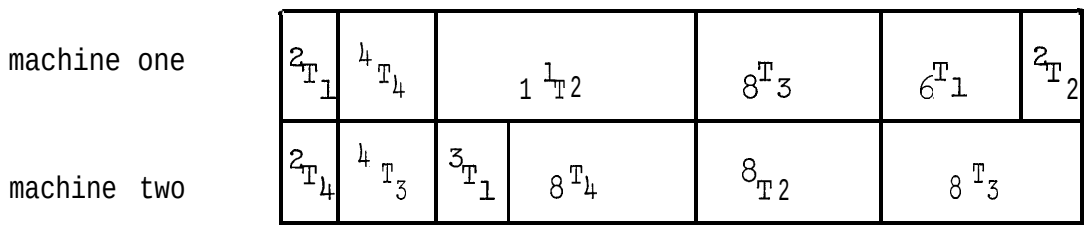
An example of a problem with the form of Gantt Chart I follows.

Given the four tasks

$$(ABA\} = \{T_1 = (2,3,6)\ ,T_2 = (11,8,2)\ \}$$

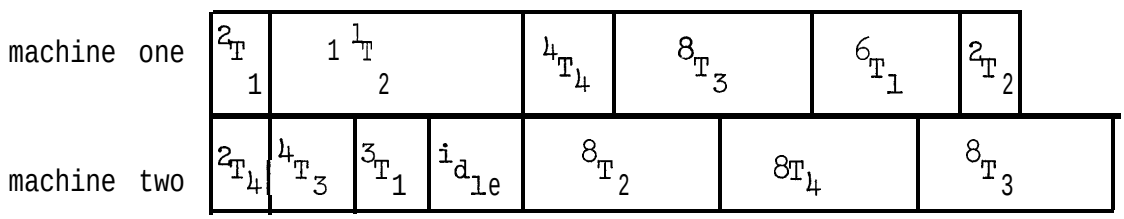
$$\{BAB\} = \{T_3 = (4,8,8)\ ,\ T_4 = (2,4,8)\ \}$$

we obtain the minimal solution of length 33.



III.C

However, again condition 2 may not be maintained without violating condition 3. The best solution that satisfies condition 2 is of length 37.



III.D

Let us now consider problems which have the form of Gantt Chart I. This form of the problem has a very simple solution. The reason for the ease of solution is that the operations are decoupled.

Definition: Two successive operations in a set of tasks in a job shop are decoupled if all of the first operations of all the tasks in the set can be completed before any of the successor operations of any of the tasks in the set-may be initiated.

In an assignment of the type of Gantt Chart I, two pairs of operations are decoupled: the first and second operations of the set {BAB} and the second and third operations of the set {ABA) . The order in which the first operations of the set {BAB} are performed, therefore, is arbitrary. Likewise, the order in which the third operations of the set {ABA} are performed is also arbitrary. The remaining operations may be assigned using Johnson's method if a feasible assignment is at all possible with this form.

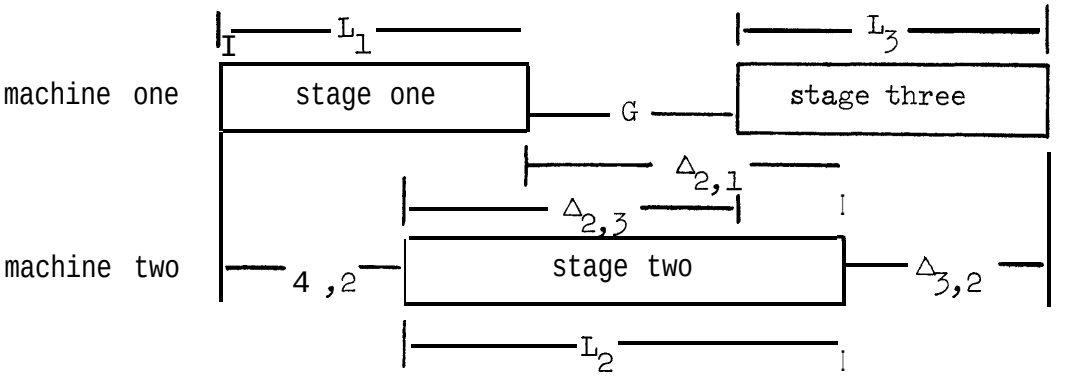
The form of the problem characterized by Gantt Chart II provides a more challenging problem, It is clear that the operations of {BAB} may be performed without any regard to their relative order since both pairs of successive operations are decoupled. We are then concerned only with the assignment order of the operations of tasks in set {ABA} .

Figure III.E depicts the form of Gantt Chart II. To discuss this sub-problem we make the following definitions.

Definition: A stage i of a given machine is a segment of time in which the i-th operations, and only the i-th operations, of all tasks are scheduled.

Definition: A delay, Δ_{ij} , is the difference between the time a task's j-th operation is initiated and its i-th operation is initiated.

Definition: The gap is the segment of time after the first stage terminates and the third stage initiates.



III.E

Figure III.E pictures the initial situation of a typical problem to schedule tasks in the set {ABA} . Each task T_k consists of three operations a_k, b_k, c_k , $k = 1, 2, \ldots, N$ corresponding to the operations to be scheduled in stages 1, 2, and 3, respectively. The length of each stage is defined by L_i as follows:

$$L_1 = \sum_{k=1}^{N} a_k \qquad L_2 = \sum_{k=1}^{N} b_k \qquad L_3 = \sum_{k=1}^{N} c_k \ .$$

The length of the gap is designated by G . $A_{1,2}$, $A_{2,3}$, $A_{2,1}$, and $\Delta_{3,2}$ designate the initial delay values.

An important concept in this assignment problem is again the contribution of a task.

Definition: Let x_i, y_i be a pair of successive operations of a task T_i. Then the contribution $C(x_i, y_i)$ of task T_i is the difference $y_i - x_i$.

The x_i, y_i of the definition may, for example, be a_i, b_i or b_i, c_i in the description of our problem. Special note should be made of the properties of the contribution. A contribution $C(x_i, y_i)$ is called positive (+) if its value is greater than or equal to zero. Likewise, $C(x_i, y_i)$ is called negative (-) if its value is less than zero. A positive contribution $C(x_i, y_i)$ increases or leaves unchanged the corresponding delay while a negative value of $C(x_i, y_i)$ decreases the same delay.

Corresponding to Johnson's first result, the order of the operations during each stage may be the same as at any other stage. The immediate advantage is that although there are n! operation. assignment orders at each stage and therefore $(n!)^3$ assignment orders for the complete problem, this resultlimitsthe solution space to n! assignment orders. The statement and proof of this result follow.

Theorem [Johnson]: Consider N tasks each with 3 operations to be processed on the first machine, the second machine, and the third machine, respectively. To construct a minimal-time solution it is sufficient to consider only schedules with the property that the operations at each stage are sequencedidentically by task number.

Proof

Given any minimal solution assignment, it is shown that the operations in the first and third stages may be reordered without extending the completion time so that operations in each of the three stages are scheduled in the same order, by task number.

1. Inspect the first assigned operation of the first stage. If it belongs to the same task as the first assigned operation of the second stage, then go to step 3.
2. If it does not, find the first stage operation that has the same task number as the first assigned operation of the second stage. Place this operation first in stage one, delaying all previously assigned operations by the length of this operation. Since the initial ordering was a solution and since no displaced operation in stage one completes before the first operation in stage two begins, the new order is still a solution.
3. Inspect the first assigned operation of the third stage. If its task number is the same as that of the first assigned operations in the first and second stages, then go to step 5.
4. If it is not, find the third stage operation. which has the same task number and place it first in stage three. All other operations of stage 3 either begin later than or at the same time they did in the initial solution. The new assignment order is then a solution also.
5. At this point, the first assigned operations at each stage of the assignment solution belongs to the same task. Remove the first assigned operation from each stage and consider the new problem resulting by repeating steps 1 to 5 on the reduced problem until no tasks remain.

We can now assume, without loss of generality, that the operations in each stage are in the same order by task number.

Let us now construct a table which allows us to determine whether or not a given assignment order is feasible between two stages.

Definition: An assignment order is feasible if no operation begins before its preceding operation is completed and no processor is idle during any stage.

For each pair of successive stages we shall construct a table that we call a feasibility table as shown in Figure III.F. Each table consists of four columns with each row corresponding to the operations of a specific task to be performed during the two stages in the order of the proposed schedule. The first column is the length of the operation performed in the first of the two stages. The second column is the contribution of the pair of operations. The third column is the sum of the contributions of all rows above plus the initial value of the delay between the two stages. The fourth column is the difference of the value in the third column minus the first column value. Since the third column represents the delay before the given operation is assigned, column four represents the excess delay time when the operation is assigned. The pair of operations may be assigned without causing idle time on the second processor only if the fourth column value is non-negative. Consequently, an assignment order between the two stages is feasible if and only if all the values in the fourth column are non-negative.

Feasibility Table

operation length	contribution	total previous contribution	excess delay
x_1	$C(x_1,y_1)$	$C(x_0,y_0)$	$C(x_0,y_0)-x_1$
x_2	$C(x_2,y_2)$	$\sum_{k=0}^{1} C(x_k,y_k)$	$\sum_{k=0}^{1} C(x_k,y_k)-x_2$
⋮	⋮	⋮	⋮
x_N	$C(x_N,y_N)$	$\sum_{k=0}^{N-1} C(x_k,y_k)$	$\sum_{k=0}^{N-1} C(x_k,y_k)-x_N$

III.F

where $C(x_0,y_0)$ is the initial value of the delay between the stages being considered and x_i,y_i refer to two successive operations of task i .

The feasibility table has a direct relation to the concept of immediate assignability.

Definition: Let a partial assignment exist after some set of tasks (possibly empty) has been assigned to the processors. A task is immediately assignable after a partial assignment if at each pair of stages the length of the first operation does not exceed the value of the delay between the two stages.

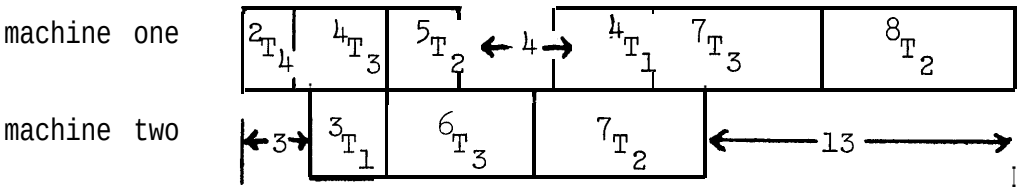
The existence of immediate assignability for each task in an assignment may be verified by the feasibility table. In the feasibility tables for each pair of successive operations no fourth column value may be negative if all tasks were immediately assignable since the fourth column represents the excess delay when a task is assigned.

The first goal is to find a canonical form of a solution of the {ABA} problem. Consider the contributions which the pairs of successive operations in each task make; that is, $C(a_k,b_k)$ and $C(b_k,c_k)$, k = 1,...,N .

IV. Case 1: Positive contributions at both stages.

Let all tasks be such that the second operation is not greater than the third operation and that the first operation is not greater than the second operation. In this case, for all $k = 1,\ldots,N$, $C(a_k,b_k) \geq 0$ and $C(b_k,c_k) \geq 0$. At any instant, whatever task is immediately assignable may be assigned. This is clear since with each new assignment, the delay at each stage may not decrease. Therefore, once a task becomes immediately assignable, it remains immediately assignable until it is assigned. Only if all tasks may be assigned, is the schedule a feasible solution. The solution is obtained by assigning the operations of any immediately assignable task at each stage.

Example IV.1:



IV.A

Figure IV.A describes a solution to the problem in which the tasks are

$T_1 = (2,3,4)$

$T_2 = (5,7,8)$

$T_3 = (4,6,7)$.

Initially the delays are $\Delta_{1,2} = 3$ and $\Delta_{2,3} = 10$. Task T_1 is the only task immediately assignable initially. After T_1 is assigned, $\Delta_{1,2} = 4$ and $A_{2,3} = 11$. Then T_3 may be assigned and $_{4,2} = 6$ and $A_{2,3} = 12$. Finally, T_2 may be assigned. At each step the delays $_{4,2}$ and $A_{2,3}$ were incremented by the respective contributions associated with each task.

V. Case 2: Negative contributions at both stages.

Let all tasks be such that the second operation is not greater than the first operation and that the third operation is not greater than the second operation. This case is the opposite of the preceding case and may be solved by "reversing time".

Definition: The mirror image problem is the problem obtained by the following two transformations.

a. The precedence among the three operations of each task is reversed. That is, if a_k precedes b_k precedes c_k in task k of the initial problem, then c_k precedes b_k precedes a_k in task k of the mirror image problem.

b. The initial delays $A_{1,2}$, $\Delta_{2,3}$, $\Delta_{3,2}$, and $\Delta_{2,1}$ of the original problem become the initial delays $A_{3,2}'$, $A_{2,1}'$, $\Delta_{1,2}$, and $A_{2,3}$, respectively.

In terms of the mirror image problem this case becomes one in which the second operation of each task is not less than the first operation and the third operation is not less than the second operation. But the mirror image problem is identical to case 1. A solution to the mirror image problem is found by applying the solution for case 1. Reversing the order of the tasks scheduled in the mirror image problem yields a minimum time solution that satisfies all precedence constraints in case 2.

Example V.1 :

In the original problem the tasks are defined as follows.

$$T_1 = (a_1, b_1, c_1) \qquad C(a_1, b_1) \leq 0 \qquad C(b_1, c_1) \leq 0$$

$$T_2 = (a_2, b_2, c_2) \qquad C(a_2, b_2) \leq 0 \qquad C(b_2, c_2) \leq 0$$

$$\cdots$$

$$T_N = (a_N, b_N, c_N) \qquad C(a_N, b_N) \leq 0 \qquad C(b_N, c_N) \leq 0$$

In the mirror image problem the tasks are:

$$T'_1 = (c_1, b_1, a_1) \qquad C(c_1, b_1) \geq 0 \qquad C(b_1, a_1) \geq 0$$

$$T'_2 = (c_2, b_2, a_2) \qquad C(c_2, b_2) \geq 0 \qquad C(b_2, a_2) \geq 0$$

$$\cdots$$

$$T'_N = (c_N, b_N, a_N) \qquad C(c_N, b_N) \geq 0 \qquad C(b_N, a_N) \geq 0 \quad .$$

VI. Case 3: A negative contribution followed by a positive contribution

Let all tasks be of the form in which both the first and third operations are greater than the second operations. Note that this problem is symmetric with respect to the first and third operations as is its mirror image problem.

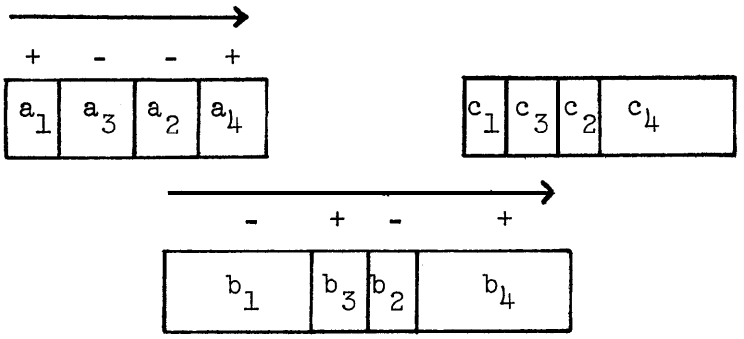
To facilitate the discussion let us consider that the general problem consists of tasks whose operations may lend either positive or negative contributions. A negative contribution in the original problem will be a positive contribution in the mirror image problem; likewise, a positive contribution in the problem will be a negative contribution in the mirror image problem. In Figure VI.A the signs above stage one indicate that

$$b_1 \geq a_1 \qquad b_3 < a_3 \qquad b_2 < a_2 \qquad b_4 \geq a_4$$

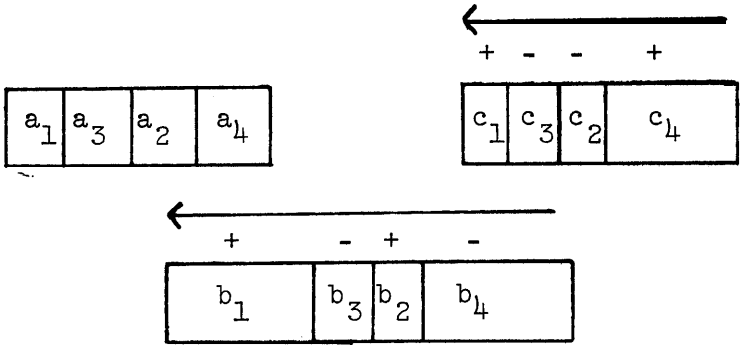
and the signs above stage two indicate that

$$c_1 < b_1 \qquad c3\ Lb3 \qquad c_2 < b_2 \qquad c_4 \geq b_4 .$$

That is, the sign above the operation of the j-th task at each stage represents the sign of the contribution of the task. The mirror image problem in Figure VI.A is diagrammed in Figure VI.B.

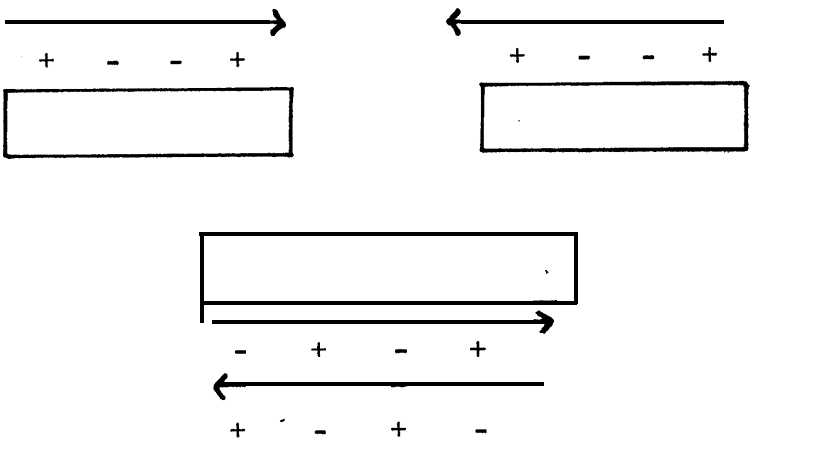


VI.A



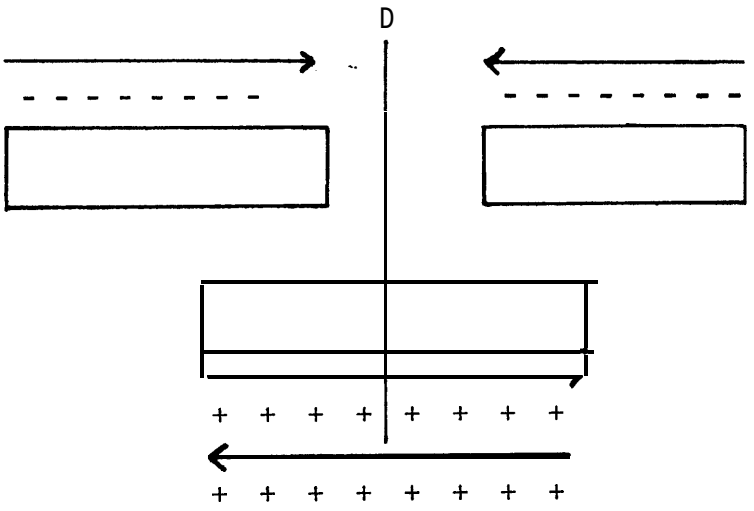
VI.B

Both Figure VI.A and Figure VI.B may be combined and abbreviated as in Figure VI.C. The arrow to the right above a row of signs indicates the contributions below are considered in the original while an arrow to the left above a row of signs indicates the contributions below are considered in the mirror image problem.



VI.C

In this new notation, Figure VI.D is a representation of the problem of this section.



VI.D

Consider that there exists a time D within the gap at which some second operation terminates and another begins in stage two. Time D occurs after stage one is complete but before stage three has been initiated. This condition clearly does not have to exist in an optimal solution of this form; a second operation may begin before stage one ends and terminate after stage three begins. The condition will be relaxed later. If such a D does exist, however, the problem is decoupled into two, two-stage problems in which the tasks for each problem have not been determined. With such a condition and Johnson's solution method, it is known that the second operations of stage two are arranged in increasing order of size in both directions from D . In other words, the operations of stage two are arranged in order of decreasing size from each end up to point D . If the second operations are arranged in a list in order of decreasing size, the task corresponding to the first

operation in the list must be either assigned first or last. Once this is decided, the problem (and the list) is reduced by one task and the solution continues in the same manner.

This solution may be expressed in terms of a functional equation. Four quantities distinguish a partial solution at any instant in the assignment process. These quantities are:

j -- the task to be assigned next, $1 \leq j \leq$.

ℓ_2 -- the total of the lengths of all the operations assigned initially in stage two

$C_{2,3}$ -- the total contribution of all operations of the tasks assigned initially in stage two

$C_{1,2}$ -- the total contribution of all operations of the tasks assigned initially in stage one

All other quantities pertaining to the assignment may be calculated from these quantities. The superscript ' indicates that the value is calculated in the mirror image problem.

$$\ell_1 = \ell_2 \, C_{1,2} \qquad \text{VI.E}$$

$$\ell_3 = C_{2,3} - \ell_2$$

$$C'_{3} = -\left(\sum_{k=1}^{j-1} C(a_k, b_k) - C_{1,2}\right)$$

$$C'_{1,2} = -\left(\sum_{k=1}^{j-1} C(b_k, c_k) \, C_{2,3}\right)$$

$$\ell'_2 = \sum_{k=1}^{j-1} b_k - \ell_2$$

$$\ell'_3 = \ell'_2 + C'_{2,3}$$

$$\ell'_1 = \ell'_2 - C'_{1,2}$$

The sums $\sum_{k=1}^{j-1} C(a_k,b_k)$, $\sum_{k=1}^{j-1} C(b_k,c_k)$, and $\sum_{k=1}^{j-1} b_k$ are properties of each task after they have been ordered in a list by decreasing size of their second operations. Hence, they need to be calculated only once.

The solution proceeds by determining if the task is immediately assignable in the original problem and if it is immediately assignable in the mirror image problem. If the task is immediately assignable in the original problem, it is tentatively assigned and the solution recurses by continuing with the next task in order. If a TRUE value is returned, a solution is found. If a FALSE value is returned or if the task is not immediately assignable in the original problem, then if the task is immediately assignable in the mirror image problem, it is tentatively assigned there and the solution recurses by continuing with the next task in order. If a value of TRUE is returned, a solution is found. Otherwise, the value FALSE is returned. In such a situation 2^N possible solution orders exist. However, the tasks are selected in a predetermined order and the value of the 4-tuple $(j,\ell_2,C_{1,2},C_{2,3})$ describes the total length and contribution assigned using the j-l tasks. As the iteration continues through the 2^N possible solutions, if a 4-tuple identical to one previously encountered occurs, it is not necessary to continue since the result will be the same as when the 4-tuple was encountered previously. In other words, the problem is reduced to a sub-problem previously attempted, This algorithm eliminates many solution possibilities from consideration.

The following algorithm determines if a solution exists.

1. Place the tasks in order of decreasing size of their second operations and renumber tasks so that $b_1 \geq b_2 \geq b_3 \geq \cdots \geq b_N$.

2. A solution exists if $f(1,0,0,0)$ is TRUE where

$f(j, \ell_2, C_{1,2}, C_{2,3}) =$

FALSE if $\ell_2 + \Delta_{1,2} > L_1 + G$

$(a_j \leq \Delta_{1,2} + C_{1,2} \wedge b_j \leq \Delta_{2,3} + C_{2,3} \wedge$

$f(j+1, \ell_2 + b_j, C_{1,2} + C(a_j, b_j), C_{2,3} + C(b_j, c_j))) \vee$

$(c_j \leq \Delta_{1,2} + C'_{1,2} \wedge b_j \leq \Delta_{2,1} + C'_{2,3} \wedge$

$f(j+1, \ell_2, C_{1,2}, C_{2,3}))$ if $L \leq j \leq N$

$f(N+1, \ell_2, C_{1,2}, C_{2,3}) =$ TRUE if $L_1 \leq \ell_2 + \Delta_{1,2} \leq L_1 + G$

= FALSE otherwise.

For simplicity, the solution presented here does not yield the explicit assignment order. This order may be easily obtained by modifying f to have a result of an ordered pair of values. The first value being TRUE or FALSE as described above. The second value is null if the first value is FALSE. Otherwise, when the first value is TRUE, the second value is a list of tasks assigned in the original problem. At each iteration a task is appended to the list if it is assigned in the original problem.

The number of calculations of f for a solution given the tasks and sizes of operations initially is bounded by

$N \cdot (L_1 + G - \Delta_{1,2}) \cdot r(C_{12}) \cdot$ [illegible] $, 3)$

where $r(x)$ indicates the number of values in the range of x plus 5N additions to calculate the contributions and sums indicated in Figure VI.E.

To relax the restriction placed on the solution by the point D, consider all tasks with a second operation of size greater than G+1; say, there are P such tasks. The above solution method must be repeated P times for each such task k with $\ell_2 + \Delta_{1,2}$ not to exceed L_1-1 at any step and with $\ell_2 + \Delta_{1,2} + b_k > L_1 + G$ at termination. P is bounded by N-1.

Example VI.1:

Using the above algorithm we find the solution to a simple four task problem

$T_1 = (7,5,6)$ $T_2 = (8,2,5)$

$T_3 = (2,1,2)$ $T_4 = (2,1,2)$

Initially, we find that

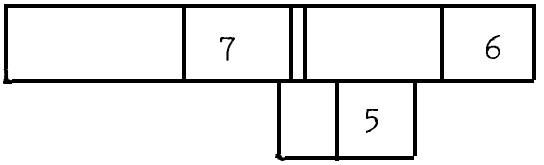
$\Delta_{1,2} = 18$ $\Delta_{2,3} = 2$

$\Delta_{2,1} = 8$ $\Delta_{3,2} = 8$

$L_1 = 19$ $G = 1$

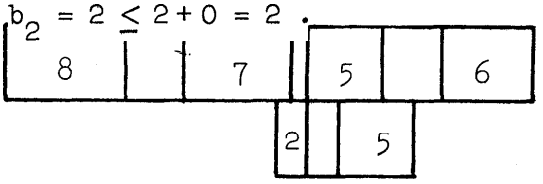
$L_1 + G = 20$

1. $f(1,0,0,0) = f(2,0,0,0)$ since $b_1 = 5 \geq 2+0 = 2$ but $c_1 = 6 \leq 8+0 = 8$ and $b_1 = 5 \leq 8+0 = 8$.

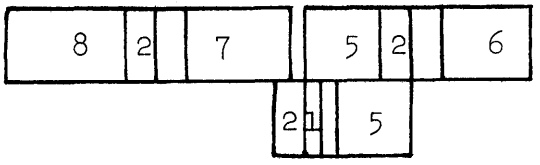


Now $C'_{1,2} = -1$ and $C'_{2,3} = 2$.

2. $f(2,0,0,0) = f(3,2,-6,3)$ since $a_2 = 8 \leq 18+0 = 18$ and $b_2 = 2 \leq 2+0 = 2$.



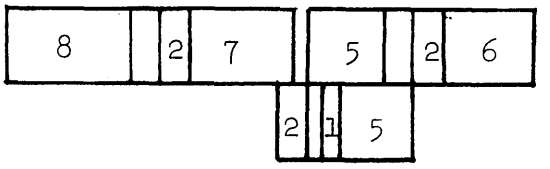
3. $f(3,2,-6,3) = f(4,3,-7,4)$ since $a_3 = 2 \leq 18-6 = 12$ and $b_3 = 1 \leq 2+3 = 5$.



4. $f(4,3,-7,4) =$ FALSE since $\ell_2 + \Delta_{1,2} = 3+18 = 21 \geq L_1 + G = 20$.

5. Return to 3 and

$f(3,2,-6,3) = f(4,2,-6,3)$ since $c_3 = 2 \leq 8 - 1 = 7$ and $b_3 = 1 \leq 8 + 2 = 10$.



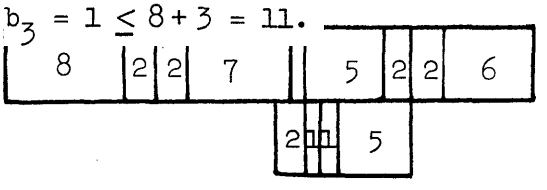
NOW $c'_{1,2} = -2$ and $c'_{2,3} = 3$.

6. $f(4,2,-6,3) = f(4,3,-7,4) = \text{FALSE}$ We have the same argument as in step 3 but we have already found that $f(4,3,-7,4)$ is FALSE in step 4.

7. Return to 6 and

since $c_3 = 2 \leq 8 - 2 - 6,3)$ sinc6 = and

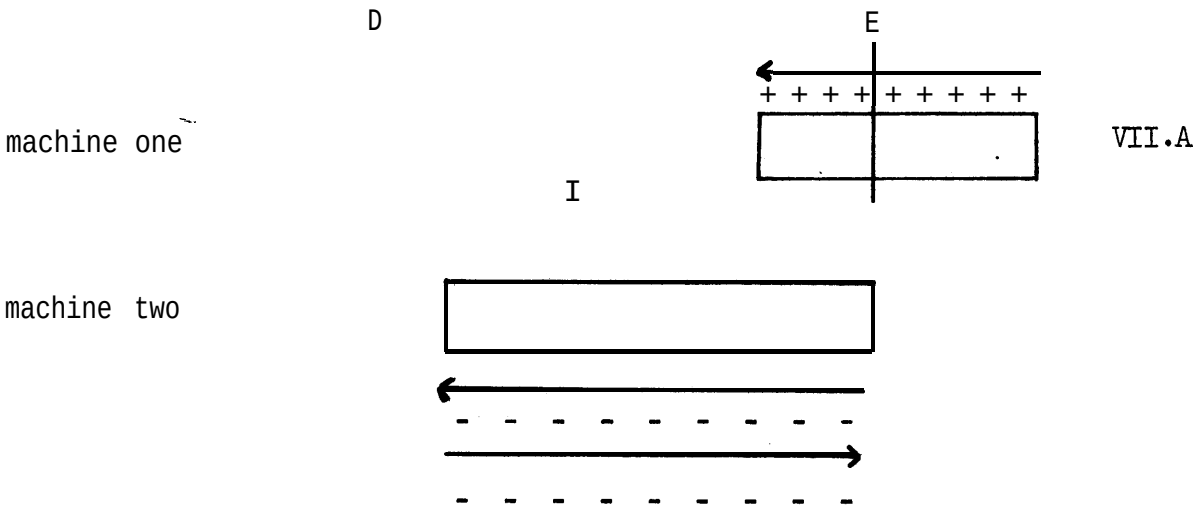
$b_3 = 1 \leq 8 + 3 = 11.$



8. $f(5,2,-6,3) = \text{TRUE}$ since $L_1 = 19 \leq \ell_2 + \Delta_{1,2} = 2 + 18 = 20 \leq L_1 + G = 20$.

VII. Case 4: The "core" problem

Let all tasks be of the form in which the first operation is less than the second operation and the second operation is greater than the third operation. Note that as in the previous sub-problem, this sub-problem has the same characteristics as its corresponding mirror image problem. In Figure VII.A the signs of the contributions are indicated near each stage.



VII.A

No efficient solution has been found to this sub-problem, and hence it represents the "core" of our stated problem. Certain efficiencies may be gained on this special problem which do not readily lend themselves well to incorporation into the general problem.

In reference to Figure VII.A, there exists some point D in stage one when all remaining tasks are immediately assignable. At this instant the problem is decoupled into a two stage problem consisting of stages two and three with the remaining tasks. Similarly, at E the third stage is decoupled from stages one and two reducing the problem to a different two stage problem consisting of stage one and stage two with the remaining tasks. When both D and E have been reached, the problem is completely decoupled, and the tasks may be assigned in any order.

A solution may theoretically be found in a computation using a variation on the usually efficient functional equation method. In this solution the number of computational steps is dominated by 2^m where m is at most N-l .

In the following algorithm, the value B is an ordered array of N binary valued elements corresponding to each of the N tasks in order. A given element B_k of the B array is 1 if the k-th task has been assigned and 0 if it has not been assigned. The notation $B \vee B_k = 1$ means that the value B is unchanged except that the B_k element is set to 1 . The value ϕ means that all elements of B have the value 0 . The algorithm is similar to the algorithm for case III. However, here the tasks previously assigned are explicitly recorded in the B array.

1. Arrange the N tasks in an arbitrary order $T_1, T_2, \ldots, T_N$.

2. A solution exists if $f(1,\emptyset,0,0,0)$ is TRUE where

$f(j,B,\ell_2,C_{1,2},C_{2,3})$ =

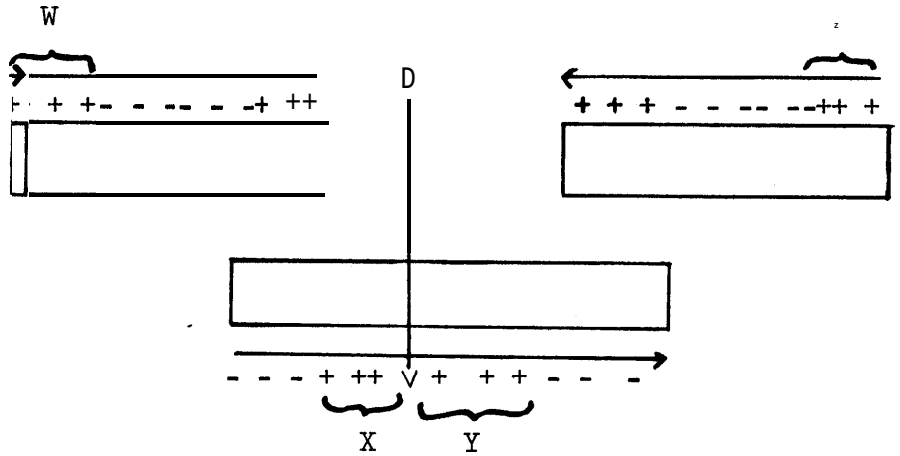
FALSE	if $j = N+1$
TRUE	if $\ell_1 \geq D$ and $\ell_3 \geq E$
$f(j, B, \ell_2, C_{1,2})$	if $\ell_3 \geq E$
$f(j, B, \ell_2, C_{2,3})$	if $\ell_1 \geq D$
$f(j+1, B, \ell 2, C_{1,2}, C_{2,3})$	if $B_j = 1$
$(a_j \leq \Delta_{1,2} + C_{1,2} \wedge b_j \leq \Delta_{2,3} \wedge$	
$f(1, B \vee B_j = 1, \ell_2 + b_j, C_{1,2} + C(a_j, b_j), C_{2,3} + C(b_j, c_j)) \vee$	
$f(j+1, B, \ell_2, C_{1,2}, C_{2,3})$	if $B_j = 0$

As in the previous functional equation solution, the exact order of assignment may be found by pairing this ordering with the TRUE logical values. The maximum number of computations is found by taking the number of computations in the previous functional equation example and multiplying it by 2^N .

VIII. The Complete Problem

The general problem may have tasks of each of the forms described in the four sub-problems. However, since it is clear that tasks of the first sub-problem and the tasks of the second sub-problem may be assigned as soon as they become immediately assignable, Figure VIII.A exhibits the canonical form of the solution for the problem which only involves tasks of the types described in sub-problems three and four and which has a decoupling point D . In this solution form, the tasks in stage one and stage three are arranged so that some of the positively contributing operations are grouped first, followed by all the negatively contributing operations, followed by the remaining positively contributing operations. In stage two, groups of the negatively contributing operations both precede and succeed the positively contributing operations. The reason for this canonical form is based on Johnson's result since the point D decouples the problem into two, two-stage problems. Hence in Figure VIII.A, the tasks in groups W and Y are arranged, left to right, in order of increasing size of the respective operations. In groups X and , the tasks are arranged, left to right, in order of the decreasing size of the respective operations.

A solution to this problem may be found by combining the solutions presented to the four sub-problems into one algorithm.



VIII.A

Ix. Conclusions

We have discussed the problem of scheduling N 3-stage tasks on two processors, a problem that has historically resisted efficient solution. When schedules are restricted so that operations are scheduled by stages, Johnson's scheduling technique and a functional equation scheduling technique introduced here can be applied to obtain feasible schedules with high computational efficiency. The problem divides into several cases, all but one of which can be solved with algorithms that grow algebraically with the size of problem. One case, which now may be considered the "core" problem still does not have a solution that grows algebraically although it is solved here by an algorithm that grows exponentially with the size of the problem.

The analysis of the decoupling effect described here may be extended to the similar problem with an arbitrary number of operations which may be executed in stages. In addition, we continue to research the problem of more than two processors.

Bibliography

[Conway 1967] R. Conway, W. Maxwell, L. Miller, Theory of Scheduling. Reading, Massachusetts: Addison-Wesley Publishing Co., 1967.

[Dudek 1964] R. A. Dudek, 0. F. Teuton, "Development of M-State Decision Rule for Scheduling n Jobs through M Machines." Operations Research, Vol. 12, No. 3, May 1964.

[Jackson 1956] J. R. Jackson, "An Extension of Johnson's Results on Job-Lot Scheduling." Naval Research Logistics Quarterly, Vol. 3, No. 3, September 1956.

[Johnson 1955] S. M. Johnson, "Optimal Two-and-Three-Stage Production Schedules with Setup Times Included." Naval Research Logistics Quarterly, Vol. 1, No. 1, March 1954.

[Johnson 1959] S. M. Johnson, "Discussion: Sequencing n Jobs on Two Machines with Arbitrary Time Lags." Management Science, Vol. 5, No. 3, April 1959.

[Lawler 1969] E. L. Lawler, J. M. Moore, "A Functional Equation and its Application to Resource Allocation and Sequencing Problems.,' Management Science, Vol. 16, No. 1, September 1969.

[Mitten 1959] L. G. Mitten, "Sequencing n Jobs on Two Machines with Arbitrary Time Lap." Management Science, Vol. 5, No. 3, April 1959.