1970

An investigation of an implicit enumeration technique using imbedded linear programming to solve the zero one scheduling problem

Merrick H. Cooperman
Lehigh University

AN INVESTIGATION OF AN IMPLICIT ENUMERATION TECHNIQUE USING IMBEDDED LINEAR PROGRAMMING TO SOLVE THE ZERO-ONE SCHEDULING PROBLEM

by

Merrick H. Cooperman

ABSTRACT

A computational procedure for obtaining the minimum makespan solution for scheduling N jobs through M machines with job start constraints, due date constraints, and job passing allowed is investigated concerning the method of bringing jobs into solution. The procedure uses the branch - and - bound technique with an imbedded linear program. Eight methods of job ordering are investigated using three types of problems. The eight methods are: (1) Largest Dual Variable, (2) Shortest Processing Time, (3) Earliest Due Date, (4) Minimum Slack Time, (5) Shortest Processing Time Considering Due Dates, (6) Largest Number of Operations, (7) Largest Number of Operations With Longest Processing Time, and (8) Random.

The CPU time necessary to find the feasible zero - one solution was the basis of comparison for the eight methods. Of the eight methods, the Largest Dual Variable method yielded the best results, and upon further comparison with the Random method was found to be significantly better.

AN INVESTIGATION OF AN IMPLICIT ENUMERATION TECHNIQUE USING IMBEDDED LINEAR PROGRAMMING TO SOLVE THE ZERO-ONE SCHEDULING PROBLEM

by

Merrick H. Cooperman

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1970

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 4, 1970

Date

John W. Adams

Professor in Charge

Chairman of the Department
of Industrial Engineering

ACKNOWLEDGEMENTS

The author wishes to express appreciation to:

Professor John W. Adams, who served as my faculty advisor, for his advice and guidance during the preparation and writing of this thesis.

Mr. George M. Schultz, Western Electric Engineering Research Center, for his technical guidance and encouragement in this endeavor.

Mrs. Hazel C. Rogaczewski, Western Electric Engineering Research Center, for her unending help in the programming of the algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (cont.)

ABSTRACT

A computational procedure for obtaining the minimum makespan solution for scheduling N jobs through M machines with job start constraints, due date constraints, and job passing allowed is investigated concerning the method of bringing jobs into solution. The procedure uses the branch - and - bound technique with an imbedded linear program. Eight methods of job ordering are investigated using three types of problems. The eight methods are: (1) Largest Dual Variable, (2) Shortest Processing Time, (3) Earliest Due Date, (4) Minimum Slack Time, (5) Shortest Processing Time Considering Due Dates, (6) Largest Number of Operations, (7) Largest Number of Operations With Longest Processing Time, and (8) Random.

The CPU time necessary to find the feasible zero - one solution was the basis of comparison for the eight methods. Of the eight methods, the Largest Dual Variable method yielded the best results, and upon further comparison with the Random method was found to be significantly better.

I. INTRODUCTION

The following general linear programming form has been encountered in many applications:

$$\text{Minimize} \qquad Z = \sum_{i=1}^{k} \sum_{j=1}^{M} c_{ij} X_{ij}$$

subject to the restrictions;

$$(1) \qquad \sum_{j=1}^{m} x_{ij} = 1 \qquad \text{for } i = 1, 2, \ldots, k$$

$$(2) \qquad A X \geq b$$

$$(3) \qquad \text{all } x_{ij} = 0 \text{ or } 1.$$

The third restriction makes the problem a form of integer program. It is usually referred to as a zero-one programming problem because the x_{ij} can only assume the values of zero or one.

The constraints of restriction set (1) are called multiple choice constraints.[17] This set is not usually found in the general zero-one integer programming problem. These constraints impose the restriction that any feasible solution will have one and only one non-zero variable in each constraint of restriction set (1).

Chapter II of this thesis will present the detailed structure of the scheduling problem. Ensuing chapters will describe a computational procedure for solving scheduling problems and evaluate methods for ordering the jobs to be brought into solution. The balance of this chapter is a brief review of developments in the field of integer programming with particular emphasis on the zero-one integer program. The reader who is already familiar with integer programming techniques may proceed directly to Chapter II.

The idea that first comes to mind when trying to get integer solutions is to use the simplex method and then round off the answers to the nearest integer values. Although, this may be adequate for some problems, it is inappropriate for the zero-one problem. There are two main causes for this inappropriateness. First, a solution that is optimal as a non-integer solution, is not necessarily feasible after it is rounded off, and it is often difficult to see which way the answer should be rounded to maintain feasibility. Second, if rounding can be done to maintain feasibility, there is no guarantee that this rounded-off solution will be the optimal integer solution.

The best known, and probably most often used, algorithms for the solution to the pure integer programming problem are the cutting plane techniques of Gomory. In 1958, R. E. Gomory[14] introduced a method of solution that could be used when all of the variables are constrained to be integers. This algorithm begins with an already optimal, non-integer solution, and then proceeds to add constraints which eliminate some non-integer solutions, but which does not eliminate any feasible integer solutions. The optimal solution to this new problem is now found by the use of the dual simplex method. Constraints are added until the optimal integer solution is found. Geometrically, this method is equivalent to constructing hyperplanes through the solution space in such a manner as to exclude from the solution space the non-integer solutions and yet not eliminate any feasible integer solutions. Because the new constraints are all retained, the problem size grows as the iterations proceed and the problem gets too large to be conveniently solved. While it was shown that the procedure converged, computer experience has indicated that the method is somewhat unpredictable and often requires numerous calculations. Furthermore, due to the many multiplications involved, rounding errors can occur making exact solutions difficult to find.

Gomory described an improved all-integer integer programming algorithm in 1960.[13] This new algorithm discarded the old generated constraint after each iteration in favor of the next generated constraint. While this algorithm has also been shown to terminate in a finite number of steps, the actual number required has been found to be highly variable and unpredictable. This "new" algorithm, and the previous one as well, suffer from the problem of using the dual simplex method; that is, the current solutions are always infeasible. This does not allow for stopping before optimality while retaining a meaningful solution. Young,[24] in 1965, developed a primal all-integer method in which an improving feasible solution is always maintained. This technique was extremely complicated and Young[25] subsequently found a simplified version. Balinski and Spielberg[4] feel that the importance of Gomory's work is that he showed that such methods do lead to finite iterative algorithms.

The branch and bound method is a second major approach to the solution of integer programming problems. Land and Doig[9] have proposed a branch and bound method that proceeds from an optimal non-integer solution. From the optimal non-integer solution two new problems are created. A non-integer variable is restricted to the next lower integer value for one problem and to the next higher integer value for the other. Both problems are then solved using the simplex method. The process is repeated on each of the new problems. This approach is particularly appropriate for mixed integer problems. Various elaborations and refinements have been proposed and tested by Dakin,[8] and Driebeck.[10] Computational results reported have been favorable in comparison to cutting-plane techniques.

Greenberg[15] has proposed a branch and bound solution similar to Land and Doig's.[9] The difference lies in the technique used for branching. The basic approach is to solve a normal linear programming problem giving an optimal objective function. Then a 0-1 variable is set, first to zero, and then to one with the optimal objective function being determined and labeled in each case. Then two branches are made from the current minimum objective values; reflecting the integerizing of another variable and the finding of two new optimal values. The branching and labeling continue until a node is found that is minimal with all variables integerized that should be integers. This is the optimal solution.

One of the earliest algorithms developed to solve the zero-one problem contained the concept of near-optimal solutions. This algorithm developed by Healy,[17] was specifically designed for the multiple choice programming problem. The technique starts from a simplex solution to the linear programming problem. A profit constraint which is basically the same as Gomory's cutting plane constraint is added to the problem and the new problem is solved. The process is continued until the decision variables become arbitrarily close to zero or one. A proof of this technique does not exist and only near-optimal solutions can be obtained.

All of the algorithms mentioned so far have made use of a regular linear programming solution at some stage in the process. Balas[2] and Glover[12] have demonstrated that an optimal solution to the zero-one integer programming problem can be obtained with considerably less computation than that required by methods using ordinary linear program techniques. They attack the zero-one problem directly, without the use of linear programming techniques.

One of the most significant contributions to integer programming was made by Balas[2] in 1965 when he published his additive algorithm. He demonstrated that zero-one problems can be solved with less computation than that required by methods based on regular linear programming solutions. The algorithm is essentially enumerative and employs the technique known as implicit enumeration, which is essentially a branch and bound approach. The method consists of a systematic search of the combinatorial tree of feasible solutions. Once a feasible solution is found, a bound is established on the objective function. Using this bound, and certain infeasibility tests, each branch of the tree is thoroughly investigated (implicitly) and determined to be either a source for an improved solution or useless.

The search continues along all branches until all possible solutions have been implicitly enumerated. The major feature of this algorithm is its additive nature, thus avoiding the round-off errors found in other algorithms. Another feature is that a feasible solution (not necessarily optimal) is usually found quickly and therefore an answer is available if the process is terminated before optimality.

Glover[12] has extended the ideas of Balas by including a backtracking procedure for exhaustive searching. This procedure has improved the efficiency of the algorithm and helped to reduce the necessary storage requirements. Glover also employed a "surrogate" constraint which is defined as a positive linear combination of the constraints in the original problem. Using this new constraint a "surrogate" problem is defined. Glover shows that when an optimal solution is found for this new problem which is also feasible for the original problem, then the solution is also optimal for the original problem.

Geoffrion[11] has combined Balas' original additive algorithm with the backtracking ideas of Glover and added some stronger tests. The result is a more efficient algorithm requiring much less storage of computations. The Geoffrion algorithm is unique in that it repeatedly generates the "best" surrogate constraint.

Lemke and Spielberg[20] modified Balas' additive algorithm in the hopes of increasing computational efficiency and minimizing storage requirements. They defined a state vector that records which variables are free to be assigned a value of zero or one. Maintenance of this state vector is basically a bookkeeping operation indicating which solutions are unexamined in the implicit enumeration scheme. A subset of the free variables called a preferred set is defined by identifying those variables which, when set equal to one, would cause the constraint equations to tend toward feasibility. A preferred set of variables is defined for each constraint not currently feasible. These and other modifications have greatly increased the efficiency of the algorithm. The experience of Lemke and Spielberg[20] seems to indicate that for problems with less than 44 variables their algorithm is somewhat faster than the Balas algorithm.

Lawler and Bell[18] have developed an algorithm for solving the general zero-one programming problem. However, this algorithm does not compare favorably with any of the other algorithms after the number of variables exceeds ten.

Recently Pritsker[22] has taken another look at the scheduling problem from a mathematical programming point of view. He develops equations that ensure a schedule will meet the constraints of limited resources, precedence relations between jobs, job splitting, due dates, substitution of resources, and concurrent and nonconcurrent job performance requirements. This formulation can accommodate a wide range of conditions. Pritsker makes the general comment on the size of the formulation that "it is favorably affected by an increased amount of sequencing, by relatively long jobs, and by close proximity of the scheduling horizon (or absolute due date) to the optimal project completion date." He further asserts in his conclusion that: "this research coupled with the immense research on zero-one programming codes should yield practical procedures for obtaining optimal solutions to certain types of scheduling problems."

The preceding was a very brief review of the current status of integer zero-one programming. An attempt was made to present the major techniques of current interest. For a more comprehensive review of this subject, the reader is referred to the survey papers of M. L. Balinski,[3] E. M. L. Beale,[5] Lawler and Woods,[19] and Gue, Leggett, and Cain.[16]

The procedure that will be examined in this thesis is of the implicit enumeration type. It is essentially a branch and bound algorithm that uses the simplex technique to calculate the value of the bound at each node. Before presenting the computational procedure it will be best to discuss the detailed structure of the scheduling problem.

II. STATEMENT OF THE PROBLEM

Definitions

In a job scheduling problem tasks or operations are performed by "processors" on a group of items. These processors are called "machines," and the items are referred to as "jobs." In the job-shop problem to be studied here, the machines are all different and each must process all jobs, but zero processing time is allowed. In that case the machines with zero time are placed at the end of the sequence for that job.

The "processing time" is the time needed to process a given job at a given machine. The "total processing time" is the time for a given job on all the machines. The sequence of machines through which a particular job is processed is the "technological order" for the job. The "earliest start time" is the earliest time that a particular job can be started on the first machine for some uncontrollable reason such as material or manpower availability. The "due date" is the latest possible time that a job can be finished.

A "schedule" is the sequence of jobs to be processed at a given machine and a "solution" is a set of schedules for all machines. "Makespan" is the elapsed time from the instant machines begin processing jobs to that instant that the last machine running completes its last job. An "optimum solution" is the feasible set of schedules that has the shortest possible makespan.

Restrictions

Scheduling problems usually have certain restrictions associated with them. The specific restrictions that the problem discussed here will obey are as follows:

1. All jobs are processed once on each machine. However, zero job times are allowed.
2. Each machine can process only one job at a time.
3. A job must be completed after starting on a machine (i.e. No jobsplitting).
4. The processing time for each job at each machine is known and includes any set-up time or transport delay.
5. Each job must be processed in some known technological order, but the order need not be the same for all jobs.
6. The earliest start times are known.
7. Due dates need not be the same for all jobs.
8. In-process inventory is allowed.
9. The resource cost of a solution set of schedules depends only upon the makespan.
10. The job processing schedule may vary between machines.

The Job Shop Problem

The problem this thesis is concerned with is the processing of N jobs on M machines. For this problem a Processing Time Matrix, a Technological Order Matrix, an Earliest Start Time Vector, and a Due Date Vector can be formulated. This formulation is shown in Figures II-1 and II-2.

For the general job-shop problem with a finite number of jobs, there is a finite number of possible solutions. Therefore, it is not necessary to prove that an optimum solution exists because all solutions can be completely defined as particular permutations of the job list at each machine. This corresponds to (N!) possible schedules at each machine. Since any schedule at a machine can be combined with any of the (N!) schedules at another machine, there are $(N!)^M$ total possible sets of schedules when the technological order is specified. Of these $(N!)^M$ possible schedules only between (N!) and $(N!)^M$ are feasible solutions. All $(N!)^M$ possible solutions are not feasible since certain sets of schedules may not be possible within the constraints of the known technological orderings. For example, consider the problem of scheduling two jobs on two machines. If the required technological order is the same for both jobs, see Figure II-3, all possible solutions are feasible as shown in that figure. However, if the technological order for Job 2 is reversed, there are only three feasible solutions. Solution 3 becomes infeasible since it is impossible to schedule Job 2 ahead of Job 1 on Machine 1 while at the same time schedule Job 1 ahead of Job 2 on Machine 2.

The scheduling problem to be investigated can be stated mathematically as follows:

Minimize $Z = \sum_{i=1}^{N} \sum_{j=1}^{L_i} c_{ij} x_{ij}$

Subject to the constraints;

1. $\sum_{j=1}^{L_i} x_{ij} = 1$ for $i = 1, 2, \ldots, N$

2. $AX \leq B$

3. all $x_{ij} = 0$ or 1.

The variables are defined as follows:

N = the total number of jobs available for scheduling

L_i = the total number of schedules possible for job i

$x_{ij} = \begin{cases} 1 \text{ if job } i \text{ is scheduled to be started in schedule } j \\ 0 \text{ otherwise} \end{cases}$

c_{ij} = Cost associated with starting job i in schedule j

b_k = amount of resource available for scheduling on any machine m for time period k

$a_{ij}^{km} = \begin{cases} 1 \text{ if job } i \text{ can be processed using schedule } j \text{ on} \\ \text{machine } m \text{ in time period } k \\ 0 \text{ otherwise} \end{cases}$

Figure II-4 shows the formulation of this problem in tableau form.

Byrne[6] indicates that in order to be able to use branch-and-bound techniques in the solution of job shop problems, two requirements must be met:

1. It must be possible to express all possible strategies conceptually in a spanning tree structure.

2. It must be possible to calculate a lower bound for cost at any node of the tree, where this lower bound is the cost of any strategy which passes through the node.

For computational efficiency a third requirement could be added:

3. The lower bounds calculated should be reasonably close to the actual values of the cost.

Lawler and Wood[19] give an excellent description of the branch-and-bound technique and state that "branching-and-bounding is an intelligently structured search of the space of all feasible solutions. Most commonly, the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets, and a lower bound (in the case of minimization) is calculated for the cost of the solutions within each subset. After each partitioning, those subsets with a bound that exceeds the cost of a known feasible solution are excluded from all further partitionings. The partitioning continues until a feasible solution is found such that its cost is no greater than the bound for any subset."

The branch-and-bound technique is concerned with the solution of a combinatorial problem. A combinatorial problem is one of assigning discrete numerical values to some finite set of variables X, in such a way as to satisfy a set of constraints and minimize some objective function, Z(X). Agin[1] has defined the algorithm employing the branch-and-bound method. However, a statement of the problem and certain preliminary definitions are required as a basis for his general definition. The problem to be discussed, as in most descriptions in the literature, is the "traveling salesman problem" of Little, et al.[21]

Consider the assignment of the values 0 or 1 to variables $X = \{x_{ij}\}$ for the objective function

$$\text{Minimize } Z(X) = \sum_{i} \sum_{j} c_{ij}\ x_{ij} \quad (i,\ j = 1,\ 2,\ \ldots,\ N),$$

where x_{ij} is one if the salesman travels from city i to city j and zero otherwise and c_{ij} is the cost associated with traveling from city i to city j.

Now consider the following definitions from Agin:[1]

Feasible solution: An assignment of numerical values to X that satisfies all the constraints.

s: A solution to the problem.

Ω: $\Omega = \{s\}$, the complete solution space for the combinatorial problem.

S: A subset of Ω

Partition of Ω : The exhaustive division of Ω into disjoint subsets $S_1, S_2, \ldots, S_n$.

Branching: The process of partitioning a subset S into m disjoint subsets $S_1, S_2, \ldots, S_m$, where $S_1 \cup S_2 \cup \ldots \cup S_m = S$, and $S_i \cup S_j = \Phi$, $i \neq j$.

S(N): The subset $S \in \Omega$ represented by the node N.

Intermediate node: A node N from which no branching has yet taken place.

Final node: An intermediate node which consists of a single solution, s.

L(N): A lower bound on the value of the objective function for all solutions associated with node N.

Using these definitions Agin[1] defines the branch-and-bound algorithm, in general, as "A set of rules for

(1) branching from nodes to new nodes,

(2) determining lower bounds for the new nodes,

(3) choosing an intermediate node from which to branch next,

(4) recognizing when a node contains only infeasible or non-optimal solutions and

(5) recognizing when a final node contains an optimal solution."

From these rules it is noted that the branching characteristic provides for the complete enumeration of all possible solutions and the bounding characteristic provides the means for recognizing an optimum solution prior to complete enumeration. Therefore, the algorithm must solve the combinatorial problem since the solution space is finite.

PROCESSING TIME MATRIX

$$\begin{array}{c|c|ccccccc}
 & & \multicolumn{7}{c}{\text{Machine}} \\
 & & 1 & 2 & \cdot & \cdot & \cdot & m \\
\hline
 & 1 & x_{11} & x_{12} & \cdot & \cdot & \cdot & x_{1m} \\
 & 2 & x_{21} & x_{22} & \cdot & \cdot & \cdot & x_{2m} \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\text{Job} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 & n & x_{n1} & x_{n2} & \cdot & \cdot & \cdot & x_{nm}
\end{array}$$

Where x_{ij} = time to process job i on machine j

TECHNOLOGICAL ORDER MATRIX

$$\begin{array}{c|c|ccccccc}
 & & \multicolumn{7}{c}{\text{Process}} \\
 & & 1 & 2 & \cdot & \cdot & \cdot & m \\
\hline
 & 1 & 1 & 6 & \cdot & \cdot & \cdot & 2 \\
 & 2 & 2 & m & \cdot & \cdot & \cdot & 4 \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\text{Job} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 & n & 5 & 7 & \cdot & \cdot & \cdot & 3
\end{array}$$

Where job i must be processed in the machine order specified.

FIGURE II-1

EARLIEST START TIME VECTOR

Job	1	2	.	.	.	n
Time	Y_1	Y_2	.	.	.	Y_n

Where Y_i is the earliest start time permitted for job i to be processed.

DUE DATE VECTOR

Job	1	2	.	.	.	n
Time	Z_1	Z_2	.	.	.	Z_n

Where Z_i is the latest time permitted for job i to be processed.

FIGURE II-2

TECHNOLOGICAL ORDER

Process Job	1	2
1	1	2
2	1	2

Solution 1: Machine 1 Schedule = 1, 2
Machine 2 Schedule = 1, 2

Solution 2: Machine 1 Schedule = 1, 2
Machine 2 Schedule = 2, 1

Solution 3: Machine 1 Schedule = 2, 1
Machine 2 Schedule = 1, 2

Solution 4: Machine 1 Schedule = 2, 1
Machine 2 Schedule = 2, 1

FIGURE II-3

$$
\begin{array}{lll|ccccccccccccccc|c|l}
\text{Variables} & & & x_{11} & x_{12} & \cdots & x_{1L_1} & x_{21} & x_{22} & \cdots & x_{2L_2} & \cdots & x_{N1} & x_{N2} & \cdots & x_{NL_N} & & \text{B} \\
\text{Objective} & & \text{MIN } Z = & c_{11} & c_{12} & \cdots & c_{1L_1} & c_{21} & c_{22} & \cdots & c_{2L_2} & \cdots & c_{N1} & c_{N2} & \cdots & c_{NL_N} & & \\
\hline
 & & \text{Job } 1 & 1 & 1 & \cdots & 1 & & & & & & & & & & = & 1 \\
 & & \vdots & & & & & & & & & & & & & & & \vdots \\
 & & \text{Job } N & & & & & & & & & & 1 & 1 & \cdots & 1 & = & 1 \\
 & & 1 & a^{11}_{11} & a^{11}_{12} & \cdots & a^{11}_{1L_1} & a^{11}_{21} & a^{11}_{22} & \cdots & a^{11}_{2L_2} & \cdots & a^{11}_{N1} & a^{11}_{N2} & \cdots & a^{11}_{NL_N} & \le & b_1 \\
 & & \vdots & & & & & & & & & & & & & & & \\
\text{MACH } 1 & \text{Time Periods} & k & a^{k1}_{11} & a^{k1}_{12} & \cdots & a^{k1}_{1L_1} & a^{k1}_{21} & a^{k1}_{22} & \cdots & a^{k1}_{2L_2} & \cdots & a^{k1}_{N1} & a^{k1}_{N2} & & a^{k1}_{NL_N} & \le & b_k \\
 & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\
 & & 1 & & & & & & & & & & & & & & \le & b_1 \\
 & & \vdots & & & & & & & & & & & & & & & \vdots \\
\text{MACH } 2 & \text{Time Periods} & k & & & & & & & & & & & & & & \le & b_k \\
 & & \vdots & & & & & & & & & & & & & & & \vdots \\
 & & 1 & & & & & & & & & & & & & & \le & b_1 \\
 & & \vdots & & & & & & & & & & & & & & & \vdots \\
\text{MACH } M & \text{Time Periods} & k & a^{kM}_{11} & a^{kM}_{12} & \cdots & a^{kM}_{1L_1} & a^{kM}_{21} & a^{kM}_{22} & \cdots & a^{kM}_{2L_2} & \cdots & a^{kM}_{N1} & a^{kM}_{N2} & \cdots & a^{kM}_{NL_N} & \le & b_k \\
 & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots & & \vdots
\end{array}
$$

FIGURE II-4

III. DESCRIPTION OF THE COMPUTATIONAL PROCEDURE

The algorithm that is being investigated in this thesis to solve the scheduling problem was suggested by G. M. Schultz[23]. The algorithm consists of the following steps (see Figure III-3).

(1) Solve the linear program using the regular simplex method. If all the variables are 0 or 1, the solution is optimal and no further work is needed. If some variables are fractional, then call this level zero (b=0), and proceed to step (2) after setting b = 1.

(2) Select a job which is not in solution at the b level. In general a job i will be selected for consideration. Go to step (3).

(3) Set variable $x_{i1} = 1$. Solve the resulting linear programming problem as a regular problem and record the value of the objective function. Go to step (4).

(4) Set variable $x_{i2} = 1$. Repeat the same process as in step (3). Go to step (5).

(5) Repeat the same process as in step (3) for all variables associated with this job, recording the value of the objective function for each variable. If for the variables x_{ij} which were set equal to one, the solution contains any $b_k < \sum_j a_{ij}^{km}$ in the original problem, the solution is infeasible and the objective function is set equal to infinity. Proceed to step (6).

(6) If some solutions Z_{bj} are [0,1] , label these nodes as feasible solutions. Go to step (7).

(7) Branch from that node with the lowest objective function value. If this node is a [0,1] solution, then an optimal solution has been found. If not, set the value of b for the lowest Z_{bj} to b_m. Go to step (8).

(8) If there are L jobs at this level which are slated to start in the same time period for all nodes, then let $b = b_m + L + 1$, if not, let $b = b_m+1$. Go to step (9).

(9) If $L>0$, proceed to level $b = b_m + L + 1$ and slot all jobs contained in L in their respective levels in the branching process. Go to step (2).

Figure III-1 is a flow chart of the above algorithm.

The algorithm is a branch and bound technique which uses an imbedded linear program to find the bounds and implicit enumeration to evaluate the possible solutions. This thesis is concerned with the problem of the selection of the job to be brought into solution at each level. It has been noted[7] that the order in which the jobs are brought into solution affects the amount of time needed to find an optimal solution. A discussion of the techniques to be investigated will be presented in Chapter IV.

An example of the type of scheduling problems that will be examined can be derived from the information shown in Figure III-2. Figure III-2 is an example for two jobs on three machine classes, where each class has a specified number of machines available for scheduling. The matrix that is derived using this information is shown in Figure III-3. It may not be immediately apparent how Figure III-3 was derived from the information in Figure III-2, so a brief explanation will be given. The rows of the matrix can be divided into two groups. The first group consists of the equality constraints. These constraints allow one and only one possible schedule to be used for any job. Thus, there is a one in every column for job i that shows a possible schedule for job i. Since there are four possible ways to schedule job one the first row has a one placed in the first four columns, which correspond to job one, and zero (blank) in the remaining columns. Similarly for row two which corresponds to job two.

The second group of rows consists of the possible schedules for each job. Each machine class is assigned the number of time periods available for scheduling and listed as shown in Figure III-3. The generation of this second group then proceeds by columns. There is no restriction for either job concerning starting time. If there were, just start the procedure from the appropriate time period. Check the Technological Order Matrix for the machine class order for job one. In the example, the order is 1, 2, 3. Next, note the number of machines needed for job one in each class. In the example, one machine is needed in each class for job one. A possible schedule for job one is to use the first two time periods for machine class one (since the processing time needed is two time periods), the third time period for class two, (processing time needed is one time period), and the fourth, fifth, and sixth time periods for class three (processing time needed is three time periods). Another possible schedule is to keep the same time periods for classes one and two and allow a unit of in-process inventory between classes two and three, therefore scheduling job one on class three in time periods five, six, and seven. Having reached the due date for job one, time period seven, we cannot introduce any further in-process inventory between classes two and three. Next, consider placing a unit of in-process inventory between classes one and two and generating the next schedule with no in-process inventory between classes two and three. Now increase the in-process inventory between classes two and three until time period seven is reached on the last class. Then go back to class one and increase the in-process inventory between classes one and two. Continue as previously with class three. Finally, consider delaying the start of job one on the first class and repeat the generation considering the possibility of in-process inventory between each of the classes. The number placed in each possible schedule is the number of machines needed for that job in each class. Since all classes in job one require one machine, all ones are placed in the four possible schedules for job one. Thus, the schedules for job one are found.

Job two's schedules are found in the same manner as job one's. The technological order for job two is 3, 1, 2. This means that the job starts on machine class three and finishes on machine class two. The due date for job two is the sixth time period, therefore the schedules cannot go beyond time period six on machine class two. Starting with machine class three job two is scheduled in time periods one, two, and three (processing time is three units). Proceeding to machine class one job two is scheduled in time period four and finally job two is scheduled in time period five for machine class two. Since the due date hasn't been reached yet, a unit of in-process inventory is introduced between machine classes one and two and another schedule results with job two being processed on machine class two in time period six. Since the due date has now been reached, proceed to introduce other in-process inventory and delays as for job one. Doing so results in the four schedules shown in Figure III-3. Next, looking at the number of machines needed in each class for job two it is seen that the need for machine classes one, two and three is one, two and two respectively. Therefore in each schedule for job two the number placed opposite machine class one is a one, opposite machine class two is a two, and opposite machine class three is a two. This can be seen in Figure III-3.

The right hand side is derived from the number of machines available in each machine class. Since there is only one machine available in machine class one for each of the time periods, a one is placed in the right hand side for each time period of machine class one. Similarly, a two is placed opposite each time period in machine class two and a three opposite each time period in machine class three. It is coincidental that the machine class number is the same as the number of machines available in that machine class.

The objective function to be minimized is found in the following manner. Assign a one to the schedule that finishes first for a given job. Assign a two to those schedules which finish one time period later. Assign a three to those schedules which finish two time periods later, etc. until all schedules have been given a value. This is done for each job. The problem will then find the fastest schedule since it will minimize the sum of the values in the objective function. A program has been written in Fortran IV for the IBM 360/50 to solve this type of problem, and a brief discussion of the main programs used for this thesis to solve the problem will be discussed next. An example problem is solved in Appendix A using the algorithm.

There are two distinct main programs, one of which has two subroutines. One of the main programs is used to generate the problem and the other one is the actual algorithm. This second one has the two subroutines. One subroutine is called CHOICE and it is used to determine the order in which the jobs are scheduled. CHOICE is described in Chapter IV. The other subroutine is called SIMPLE and it is used to solve the linear program. This subroutine was obtained from the SHARE library and a description of it is available from IBM. A listing is provided in Appendix B for completeness.

The program for generating the matrix from the information in Figure III-2 is called PROGEN, and is listed in Appendix B. PROGEN is currently designed to generate a problem for ten jobs on ten machines and uses approximately 100,000 bytes of core memory on an IBM 360/50 computer. The problem size may be increased to fill the available memory, and by the realignment of the number of machines and the number of jobs away from a square matrix. The problem, as generated for the algorithm, is stored on a disk pack in a packed form to reduce the necessary memory space. The generation of the problem with PROGEN is virtually the same method that was described earlier in this chapter.

The program for the actual solution to the problem is called SOLUTE and is also listed in Appendix B. SOLUTE is currently designed for any combination of jobs and machines which will yield less than nine hundred possible schedules. The problem to be solved must fit into a 53 x 900 array. SOLUTE was programmed in FORTRAN IV for the IBM 360/50 and currently requires approximately 390,000 memory bytes including the two subroutines. The program follows the algorithm that was described at the beginning of this chapter. The listing in Appendix B has appropriate comments to indicate when the program corresponds to the algorithm.

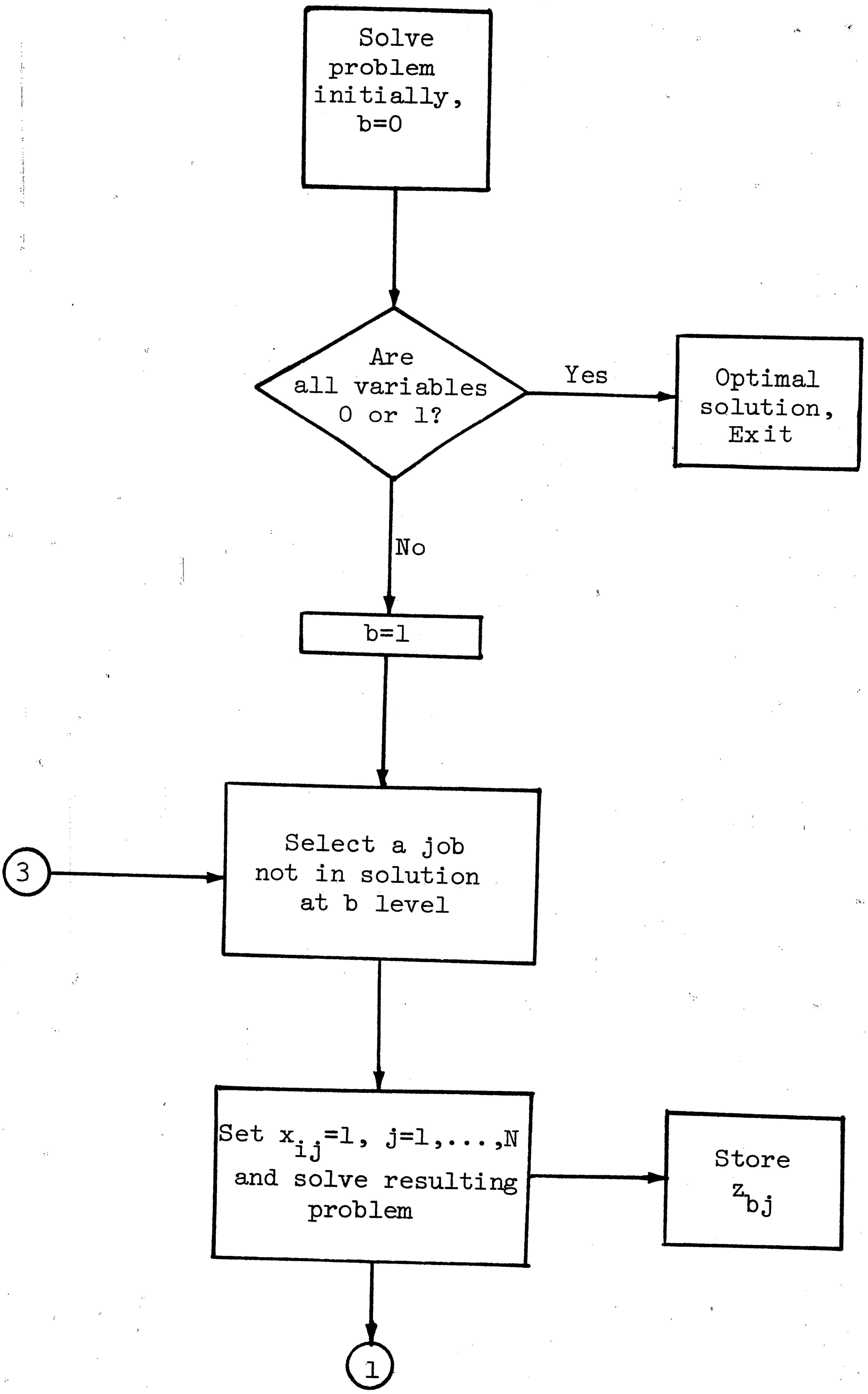

Solve problem initially, b=0
Are all variables 0 or 1?
Yes
Optimal solution, Exit
No
b=1
3
Select a job not in solution at b level
Set x_{ij}=1, j=1,...,N and solve resulting problem
Store z_{bj}
1

FIGURE III-1

Page 1 of 3

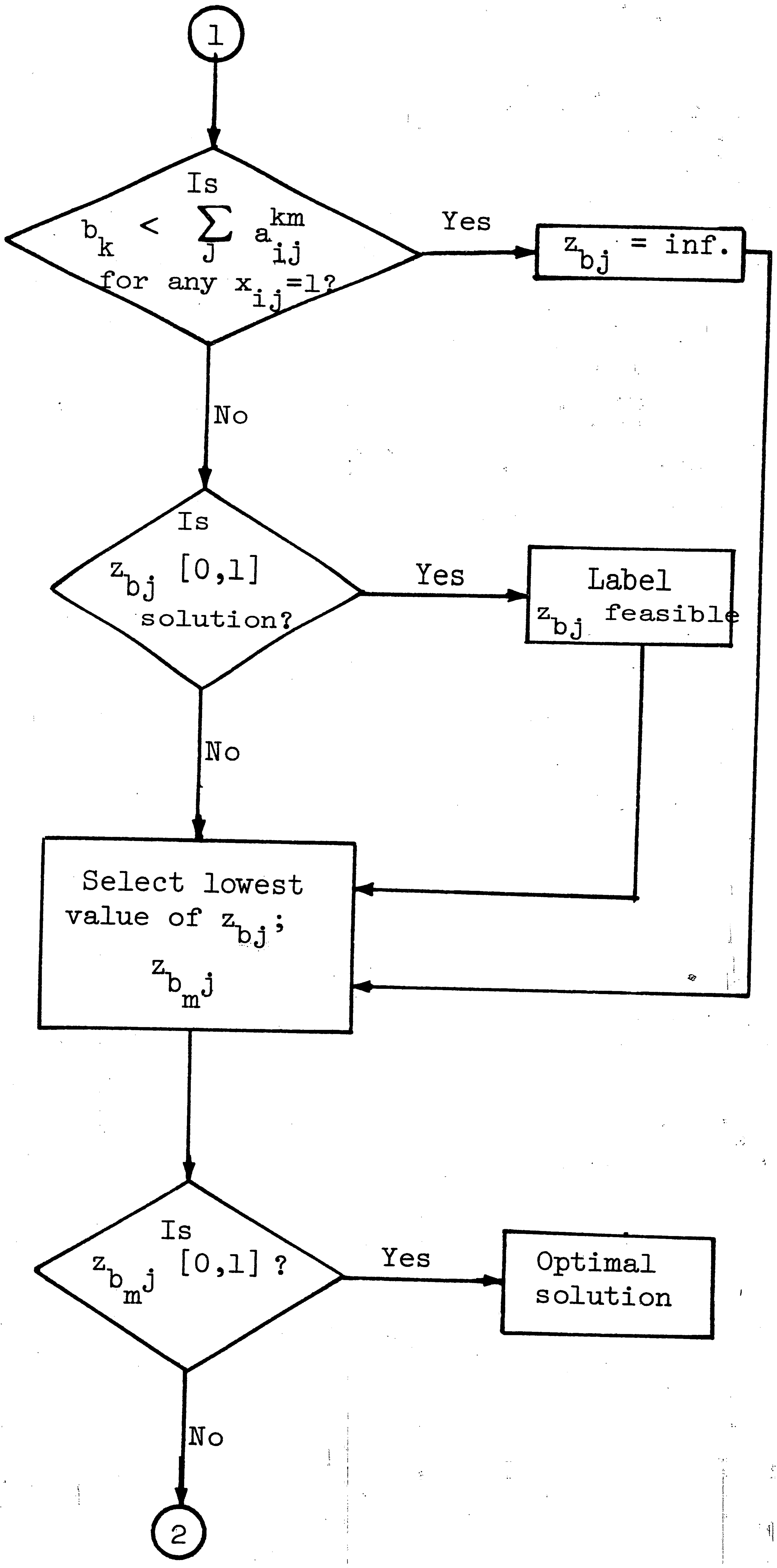


FIGURE III-1

Page 2 of 3

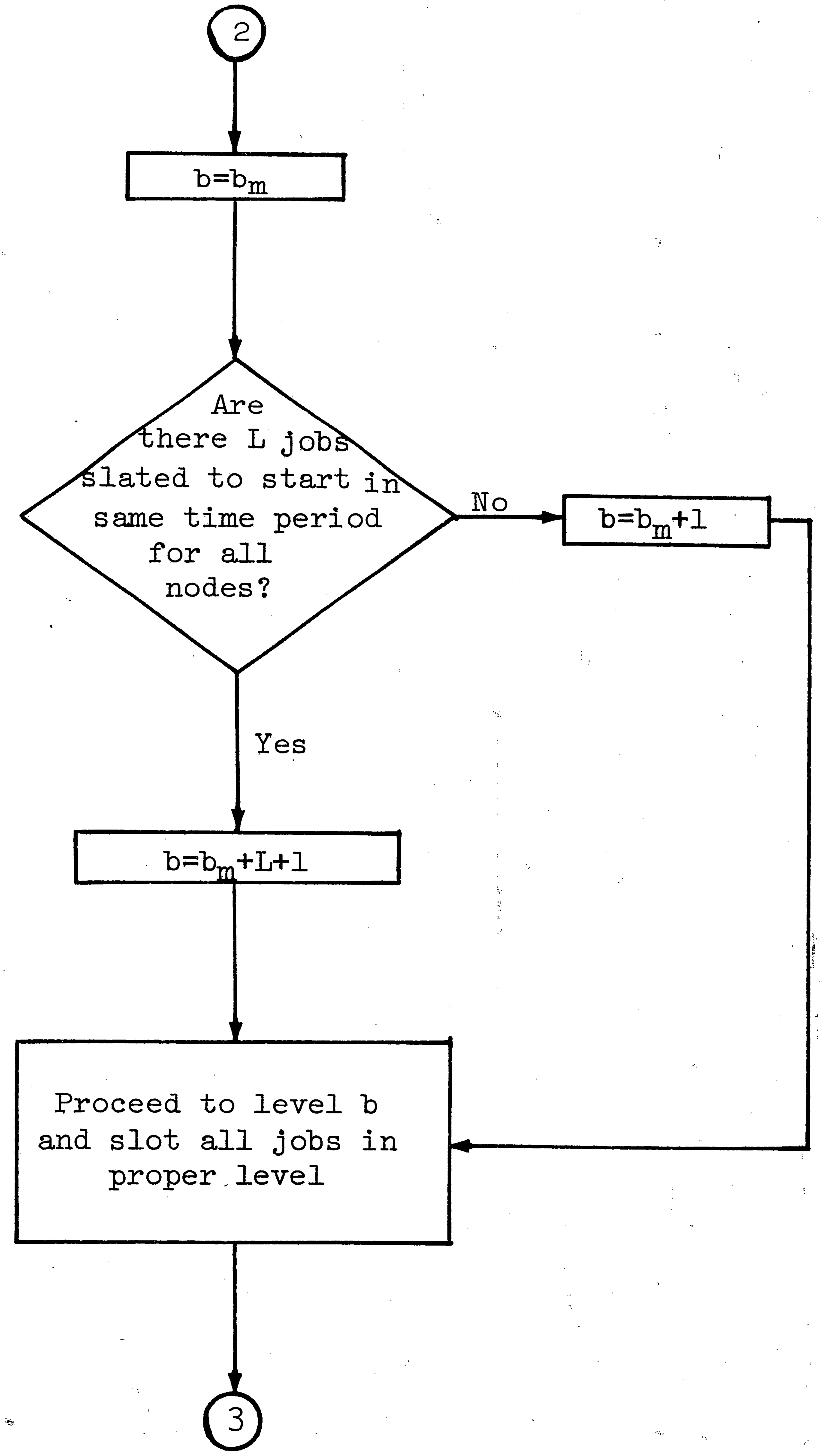

2
$b=b_m$
Are there L jobs slated to start in same time period for all nodes?
No
$b=b_m+1$
Yes
$b=b_m+L+1$
Proceed to level b and slot all jobs in proper level
3

FIGURE III-1

Page 3 of 3

PROCESSING TIME

Mach Job	1	2	3
1	2	1	3
2	1	1	3

TECHNOLOGICAL ORDER

Process Job	1	2	3
1	1	2	3
2	3	1	2

Job	Earliest Start Time	Due Date
1	0	7
2	0	6

Machines Available in each Machine Class

	Class		
	1	2	3
Avail	1	2	3

Machines Needed for each job in each class

Class Job	1	2	3
1	1	1	1
2	1	2	2

Data for scheduling 2 jobs on 3 machine classes in 7 time periods.

FIGURE III-2

		Schedules Job 1				Schedules Job 2				
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	
Job 1		1	1	1	1					=1
Job 2						1	1	1	1	=1
Machine Class 1	1	1	1	1						≤ 1
	2	1	1	1	1					≤ 1
	3				1					≤ 1
	4					1	1			≤ 1
	5							1	1	≤ 1
	6									≤ 1
	7									≤ 1
Machine Class 2	1									≤ 2
	2									≤ 2
	3	1	1							≤ 2
	4			1	1					≤ 2
	5					2				≤ 2
	6						2	2	2	≤ 2
	7									≤ 2
Machine Class 3	1					2	2	2		≤ 3
	2					2	2	2	2	≤ 3
	3					2	2	2	2	≤ 3
	4	1							2	≤ 3
	5	1	1	1	1					≤ 3
	6	1	1	1	1					≤ 3
	7		1	1	1					≤ 3
Objective Function		1	2	2	2	1	2	2	2	

FIGURE III-3

IV. Description of the Job-Ordering Methods Investigated

The subroutine CHOICE, as stated in Chapter III, is the section of the program that is used to determine the order that the jobs should be entered into solution. Eight different job-ordering methods were investigated. These eight methods are:

(1) Largest dual variable,

(2) Shortest processing time,

(3) Earliest due date,

(4) Minimum slack time,

(5) Shortest processing time considering due dates,

(6) Largest number of operations,

(7) Largest number of operations with longest processing time,

(8) Random.

The eight methods will now be discussed in the order in which they were presented.

(1) Largest Dual Variable

This subroutine is shown in Figure IV-1. Each job has a corresponding dual variable which is calculated by the subroutine SIMPLE. These dual variable values are ranked, largest first, for those jobs which are not yet in solution. This ranking is done at each level and the job with the largest dual variable, of those not yet in solution, is selected to be placed into solution next.

(2) Shortest Processing Time

This subroutine is shown in Figure IV-2. The jobs are ranked in order, from the job with the least total processing time first to the job with the most processing time last. Only those jobs which are not yet in solution are considered. The job with the least total processing time is placed into solution next.

(3) Earliest Due Date

This subroutine is shown in Figure IV-3. Each job has a date by which time that job must be finished. This date is the job's due date. Each job is ranked in order, from earliest to latest, by due date. Only those jobs which are not yet in solution are considered. The job with the earliest due date is placed into solution next.

(4) Minimum Slack Time

This subroutine is shown in Figure IV-4. The slack time for each job is obtained by subtracting the total processing time necessary for that job from the job's due date. This difference is the amount of time by which either the job can be delayed before it must be started or the amount of in-process delay allowed. The delay or slack time is then ranked in order, least first. Only those jobs not yet in solution are considered. The job with the least slack time is then placed into solution next.

(5) Shortest Processing Time Considering Due Dates

This subroutine is shown in Figure IV-5. First, the processing time for the jobs not in solution are added together. Next, considering only the jobs not yet in solution, those jobs whose due date is past the sum of the processing times found previously are ranked according to their processing times, with the largest processing time last. Now, considering only those jobs not in solution or ranked, the processing times for the remaining jobs are added together, and those jobs with due dates exceeding the new total are ranked on top of the previous jobs. This continues until all the jobs are ranked. The job with the shortest processing time, considering due dates, is now ranked in the first position and placed into solution next.

(6) Largest Number of Operations

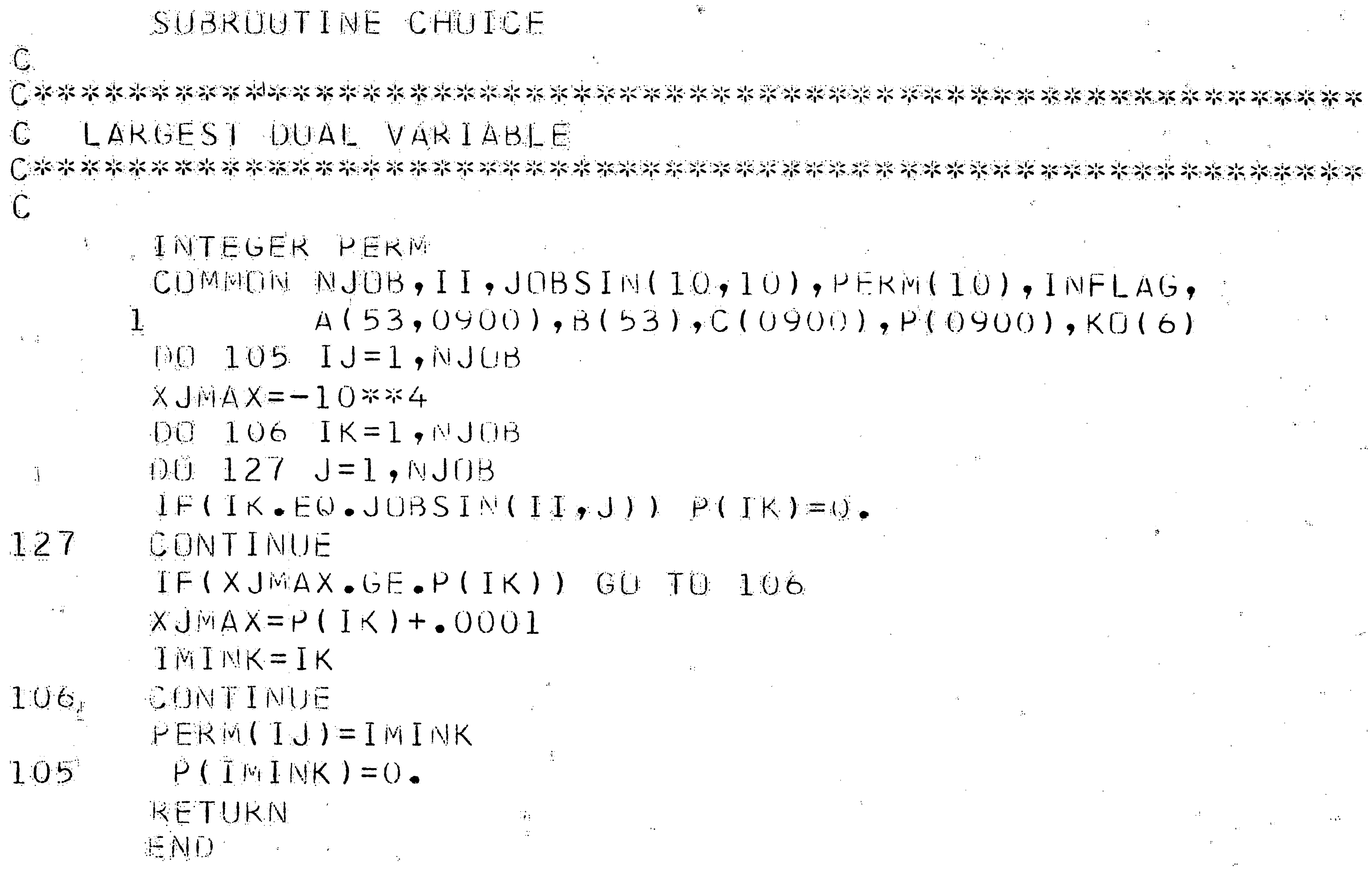
This subroutine is shown in Figure IV-6. Each job is ranked by the number of machine classes it must go through. The job with the most machine classes to be processed on is ranked first. Only those jobs not yet in solution are considered. The job that is ranked first is placed into solution next.

(7) Largest Number of Operations with Longest Processing Time

This subroutine is shown in Figure IV-7. For each job not yet scheduled a number s is calculated, where s is the sum of the reciprocals of the processing times for that job on each machine class. The jobs are then ranked according to the value of s for each job, with the lowest value of s ranked first. The job with the lowest value of s is then placed into solution next.

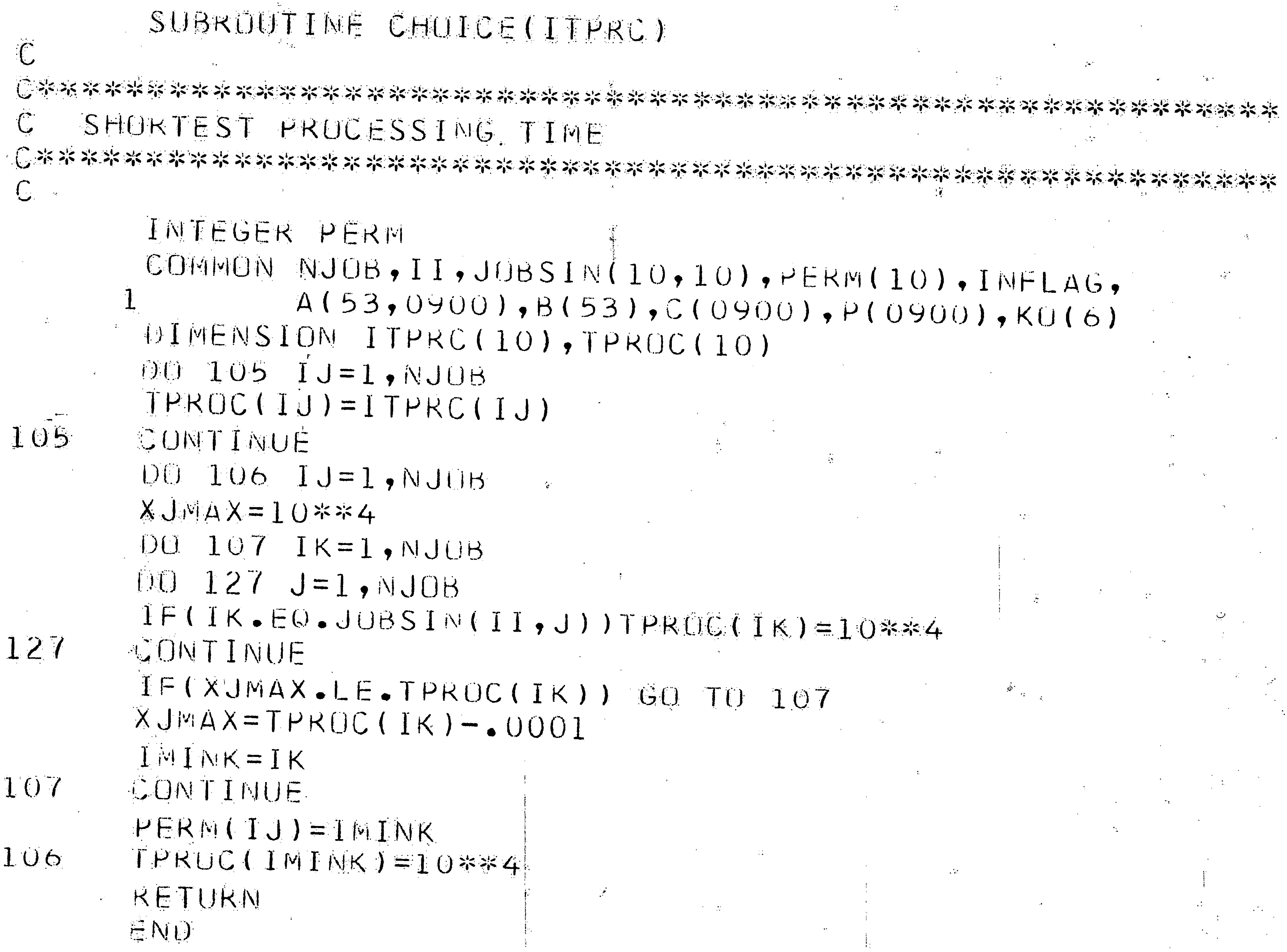
(8) Random

This subroutine is shown in Figure IV-8. A random number is generated for each job not in solution. The jobs are then ranked by descending random numbers. The job with the largest random number is placed into solution next. This subroutine is used as a basis of comparison for the other methods to see if it really matters if the jobs are brought into solution in an orderly manner.



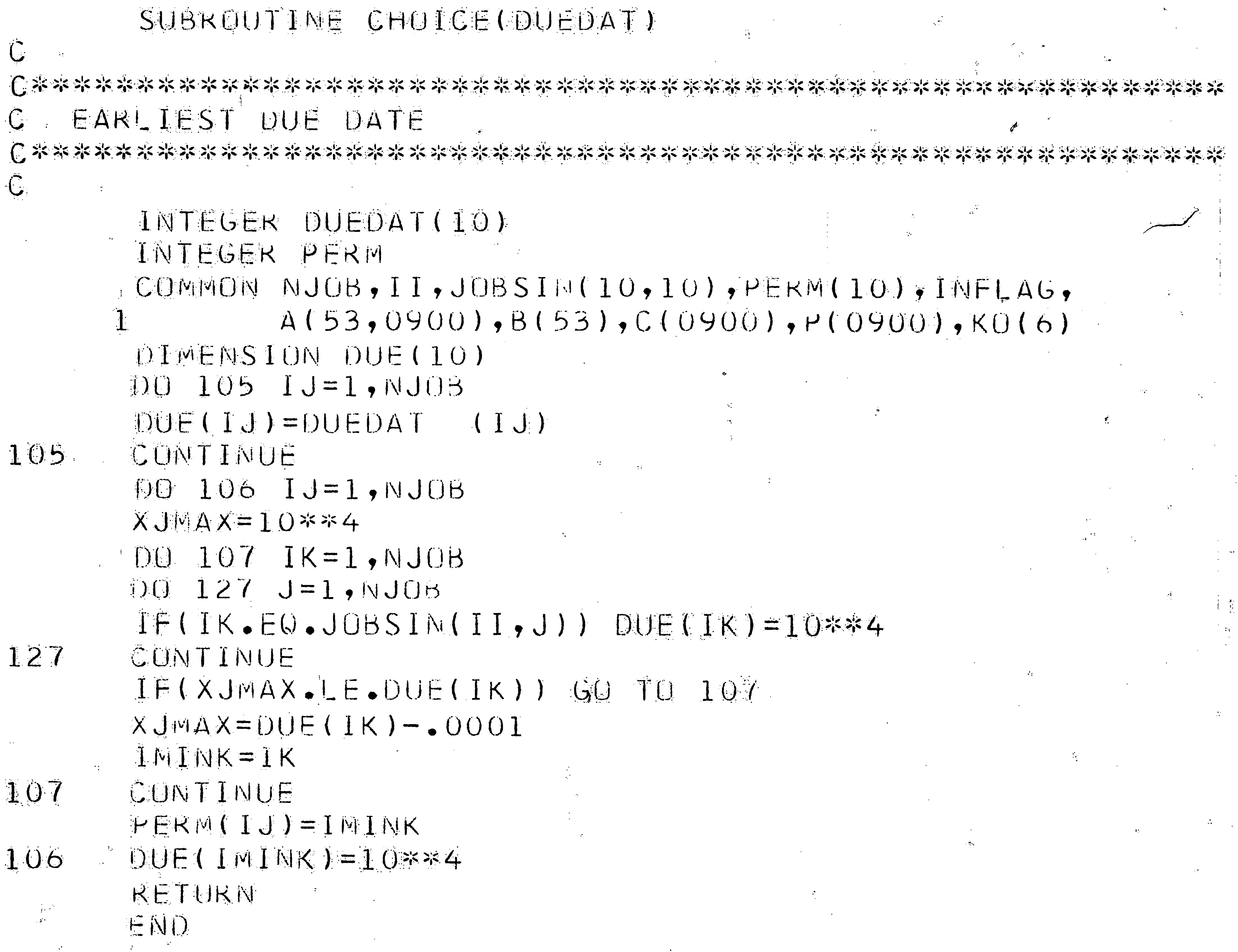
```
      SUBROUTINE CHOICE
C
C***********************************************************
C  LARGEST DUAL VARIABLE
C***********************************************************
C
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DO 105 IJ=1,NJOB
      XJMAX=-10**4
      DO 106 IK=1,NJOB
      DO 127 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J)) P(IK)=0.
127   CONTINUE
      IF(XJMAX.GE.P(IK)) GO TO 106
      XJMAX=P(IK)+.0001
      IMINK=IK
106   CONTINUE
      PERM(IJ)=IMINK
105    P(IMINK)=0.
      RETURN
      END
```

FIGURE IV-1



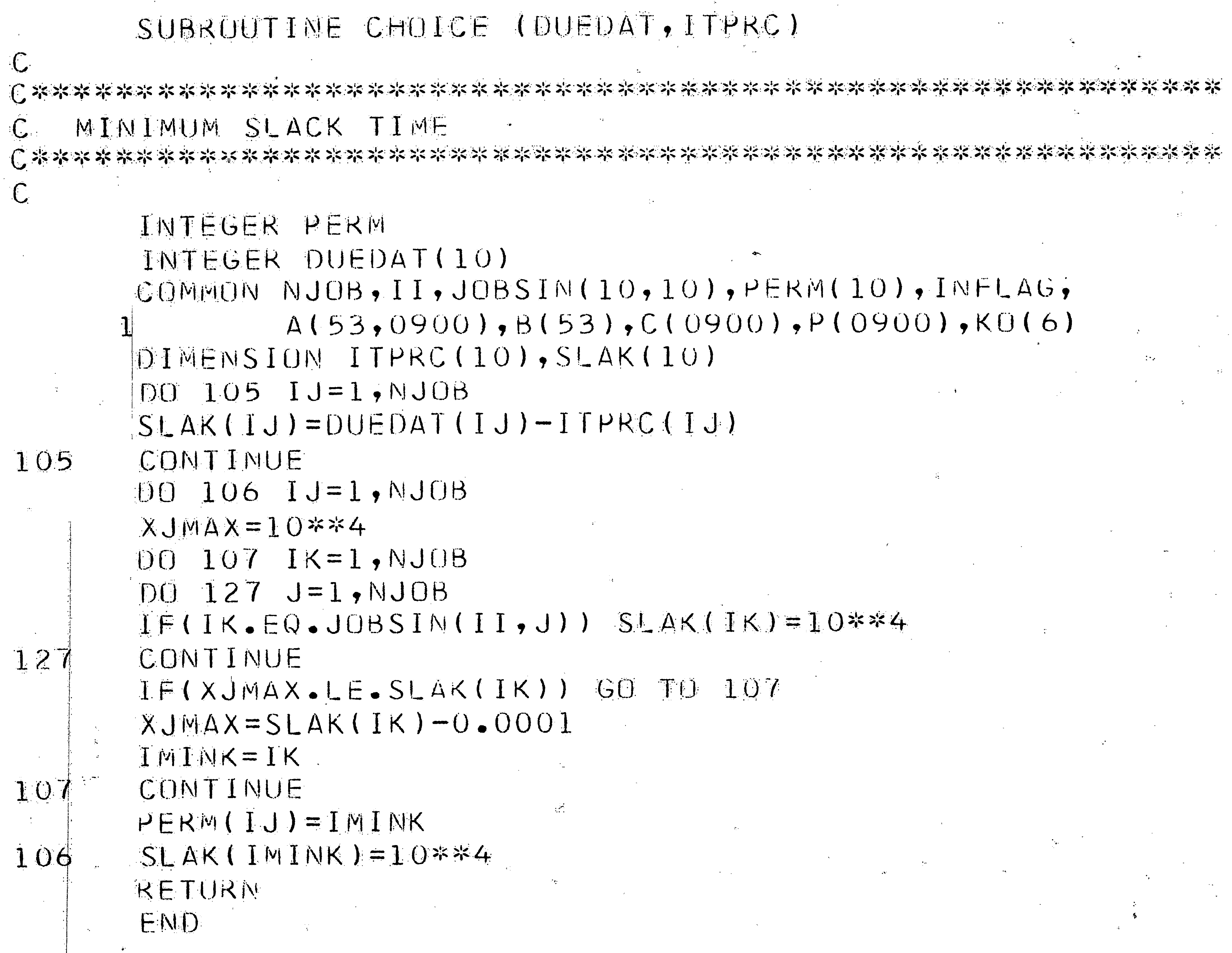
```
      SUBROUTINE CHOICE(ITPRC)
C
C************************************************************************
C  SHORTEST PROCESSING TIME
C************************************************************************
C
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DIMENSION ITPRC(10),TPROC(10)
      DO 105 IJ=1,NJOB
      TPROC(IJ)=ITPRC(IJ)
105   CONTINUE
      DO 106 IJ=1,NJOB
      XJMAX=10**4
      DO 107 IK=1,NJOB
      DO 127 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J))TPROC(IK)=10**4
127   CONTINUE
      IF(XJMAX.LE.TPROC(IK)) GO TO 107
      XJMAX=TPROC(IK)-.0001
      IMINK=IK
107   CONTINUE
      PERM(IJ)=IMINK
106   TPROC(IMINK)=10**4
      RETURN
      END
```

FIGURE IV-2



```
      SUBROUTINE CHOICE(DUEDAT)
C
C*****************************************************************
C  EARLIEST DUE DATE
C*****************************************************************
C
      INTEGER DUEDAT(10)
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DIMENSION DUE(10)
      DO 105 IJ=1,NJOB
      DUE(IJ)=DUEDAT  (IJ)
105   CONTINUE
      DO 106 IJ=1,NJOB
      XJMAX=10**4
      DO 107 IK=1,NJOB
      DO 127 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J)) DUE(IK)=10**4
127   CONTINUE
      IF(XJMAX.LE.DUE(IK)) GO TO 107
      XJMAX=DUE(IK)-.0001
      IMINK=IK
107   CONTINUE
      PERM(IJ)=IMINK
106   DUE(IMINK)=10**4
      RETURN
      END
```

FIGURE IV-3



```
      SUBROUTINE CHOICE (DUEDAT,ITPRC)
C
C*****************************************************************
C  MINIMUM SLACK TIME
C*****************************************************************
C
      INTEGER PERM
      INTEGER DUEDAT(10)
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DIMENSION ITPRC(10),SLAK(10)
      DO 105 IJ=1,NJOB
      SLAK(IJ)=DUEDAT(IJ)-ITPRC(IJ)
105   CONTINUE
      DO 106 IJ=1,NJOB
      XJMAX=10**4
      DO 107 IK=1,NJOB
      DO 127 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J)) SLAK(IK)=10**4
127   CONTINUE
      IF(XJMAX.LE.SLAK(IK)) GO TO 107
      XJMAX=SLAK(IK)-0.0001
      IMINK=IK
107   CONTINUE
      PERM(IJ)=IMINK
106   SLAK(IMINK)=10**4
      RETURN
      END
```

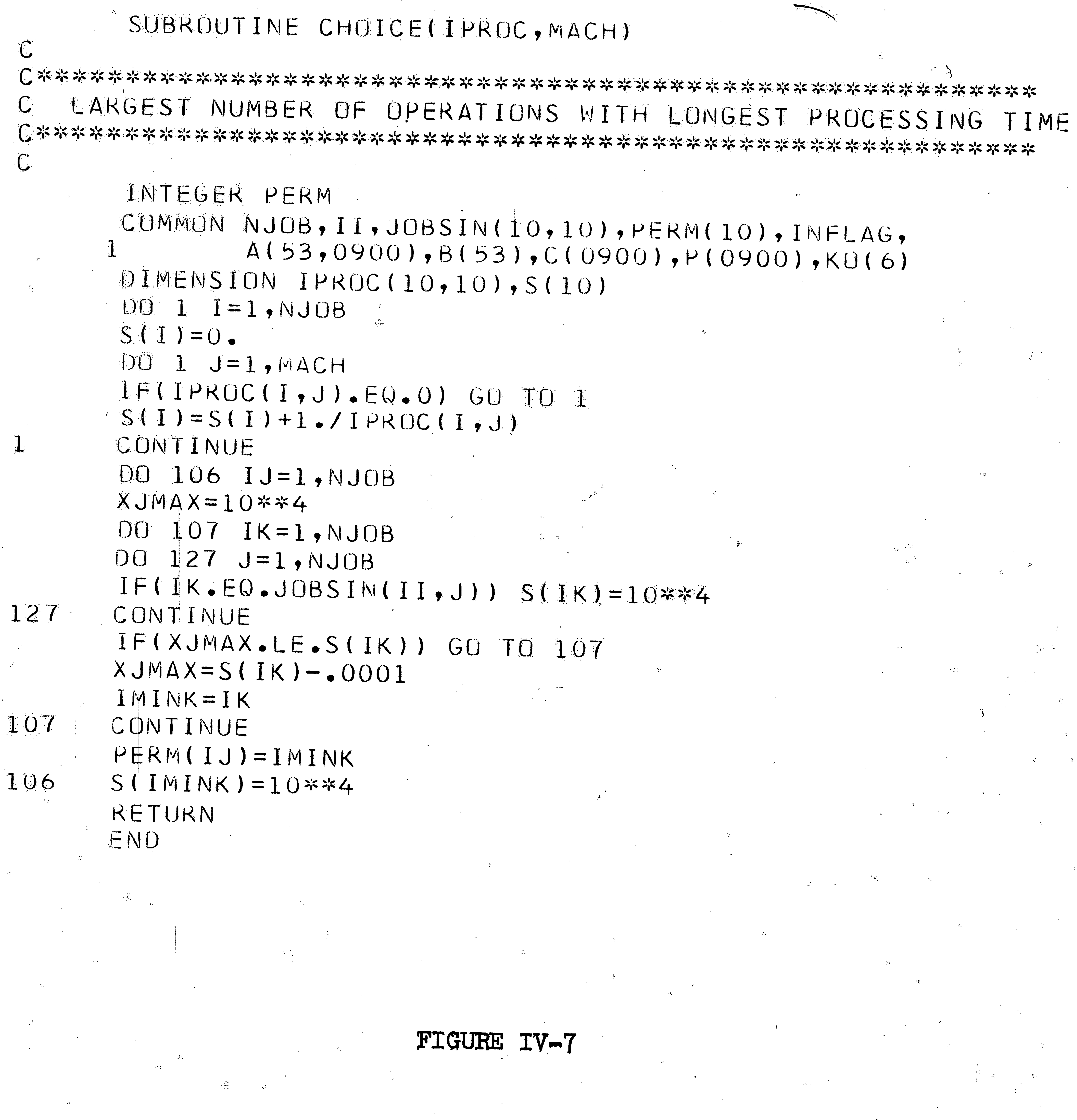
FIGURE IV-4

```
      SUBROUTINE CHOICE(DUEDAT,ITPRC)
C
C*************************************************************
C   SHORTEST PROC TIME CONSIDERING DUE DATES
C*************************************************************
C
      INTEGER PERM,DUEDAT(10)
      COMMON NJOB,II,JOBSIN(10,10),PERM(10)
      COMMON INFLAG,A(53,900),B(53),C(900),P(900),KU(6)
      DIMENSION ITPRC(10),SHORT(10), ITEM(10)
      NP=0
      DO 127 IJ=1,NJOB
      SHORT(IJ)=DUEDAT(IJ)
      IF(IJ.EQ.JOBSIN(II,IJ)) GO TO 127
      NP=NP+ITPRC(IJ)
  127 CONTINUE
      DO 105 IJ=1,NJOB
      YJMAX=-10**4
      XJMAX=-10**4
      KMINK=0
      DO 106 IK=1,NJOB
      DO 107 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J)) SHORT(IK)=0.
  107 CONTINUE
      IF(SHORT(IK).EQ.0.) GO TO 106
      DO 108 J=1,NJOB
      ITEM(J)=ITPRC(J)
      IF(ITPRC(J).LE.NP) ITEM(J)=0
  108  CONTINUE
      IF(ITEM(IK).NE.0) GO TO 110
      TPRC=ITPRC(IK)
      IF(XJMAX.GE.TPRC) GO TO 106
      XJMAX=TPRC+.0001
      IMINK=IK
      GO TO 106
  110 TPRC=ITPRC(IK)
      IF(YJMAX.GE.TPRC) GO TO 106
      YJMAX=TPRC+.0001
      KMIMK=IK
  106 CONTINUE
      IF(KMINK.EQ.0) GO TO 111
      PERM(NJOB+1-IJ)=KMINK
      SHORT(KMINK)=0.
      NP=NP-ITPRC(KMINK)
      GO TO 105
  111 PERM(NJOB+1-IJ)=IMINK
      SHORT(IMINK)=0.
      NP=NP-ITPRC(IMINK)
  105 CONTINUE
      RETURN
      END
```

FIGURE IV-5

```
      SUBROUTINE CHOICE (MACH,IPROC)
C
C*********************************************************
C  LARGEST NUMBER OF OPERATIONS
C*********************************************************
C
      DIMENSION IPROC(10,10) ,OPER(10)
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DO 127 I=1,NJOB
      OPER(I)=0.
      DO 127 J=1,MACH
      IF(IPROC(I,J).GT.0) OPER(I)=OPER(I)+1.
127   CONTINUE
      DO 105 IJ=1,NJOB
      XJMAX=-10**4
      DO 106 IK=1,NJOB
      DO 107 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J))OPER(IK)=0.
107   CONTINUE
      IF(XJMAX.GE.OPER(IK)) GO TO 106
      XJMAX=OPER(IK)+.0001
      IMINK=IK
106   CONTINUE
      PERM(IJ)=IMINK
105   OPER(IMINK)=0.
      RETURN
      END
```

FIGURE IV-6



```
      SUBROUTINE CHOICE(IPROC,MACH)
C
C*****************************************************************
C  LARGEST NUMBER OF OPERATIONS WITH LONGEST PROCESSING TIME
C*****************************************************************
C
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DIMENSION IPROC(10,10),S(10)
      DO 1 I=1,NJOB
      S(I)=0.
      DO 1 J=1,MACH
      IF(IPROC(I,J).EQ.0) GO TO 1
      S(I)=S(I)+1./IPROC(I,J)
1     CONTINUE
      DO 106 IJ=1,NJOB
      XJMAX=10**4
      DO 107 IK=1,NJOB
      DO 127 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J)) S(IK)=10**4
127   CONTINUE
      IF(XJMAX.LE.S(IK)) GO TO 107
      XJMAX=S(IK)-.0001
      IMINK=IK
107   CONTINUE
      PERM(IJ)=IMINK
106   S(IMINK)=10**4
      RETURN
      END
```

FIGURE IV-7

```
      SUBROUTINE CHOICE(IY)
C
C**********************************************************
C  RANDOM
C**********************************************************
C
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
       DIMENSION RAN(10)
      DO 1 I=1,NJOB
      IY=IY*65539
      IF(IY)5,6,6
5     IY=IY+2147483647+1
6     YFL=IY
      YFL=YFL*0.4656613E-9
      RAN(I)=YFL
1     CONTINUE
      DO 105 IJ=1,NJOB
      XJMAX=-10**4
      DO 106 IK=1,NJOB
      DO 127 J=1,NJOB
      IF(IK.EQ.JOBSIN(II,J)) RAN(IK)=0.
127   CONTINUE
      IF(XJMAX.GE.RAN(IK)) GO TO 106
      XJMAX=RAN(IK)+.0001
      IMINK=IK
106   CONTINUE
      PERM(IJ)=IMINK
105   RAN(IMINK)=0.0
      RETURN
      END
```

FIGURE IV-8

V METHOD OF EVALUATION

The input that is used to generate a problem is representative of the characteristics of that problem. The inputs for each job are:

(1) Processing time per machine

(2) Total processing time

(3) Technological order

(4) Earliest start time

(5) Due date

(6) Machines needed in each machine class

(7) Machines available in each machine class

Among these inputs the one that affects the scheduling of jobs most is the total processing time. This is because the number of schedules per job are directly related to the total processing time needed by that job. This is shown by the following formula:

$$\text{NSPJ} = \frac{(\text{IT-TPROC} + \text{NMACH})!}{(\text{NMACH})!\ (\text{IT} - \text{TPROC})!}$$

where

NSPJ = number of schedules per job

TPROC = total processing time needed on the job

IT = number of time periods the job is scheduled over

NMACH = number of machine classes used on the job

The formula also shows that the number of schedules per job is dependent upon the number of time periods the job is scheduled over and the number of machine classes used on the job. However, these two factors are kept constant for all problems at fifteen time periods and three machine classes.

The actual scheduling of the job, and the ease with which it is achieved, is dependent upon the possible number of schedules for that job. The more possible schedules, the easier it is to fit a job into a schedule to reduce the idle time for all jobs in the schedule. For that reason the problems were constructed on the basis of total processing time per job.

Four problems were constructed to illustrate the three main possible job combinations. These job combinations are: (1) all jobs of short duration; (2) all jobs of long duration; and (3) a mixture of long and short duration jobs. The four problems are shown in Figures V-1 through V-4. Problem one has four jobs all of relatively short duration. Problem two has six jobs all of relatively long duration. Problems three and four have a mixture of long and short duration jobs.

Each method will be compared on the basis of Central Processing Unit (CPU) time. This is a measure, in hundredths of a minute, of the amount of time that the program is being worked on in the central processor of the computer. A maximum of thirty minutes is allowed, and if no solution has been found in that time period the program terminates.

While the programs were being "debugged" and a sample problem run to check the validity of the program, it was noted that no optimal integer solution could often be found within the allowed thirty minutes of CPU time. This raised the problem of how to compare the efficiency of the job ordering methods. It was decided that zero-one feasibility would be used. That is, the problem would be considered solved when a feasible zero-one solution was found. This meant that it was now impossible to know whether or not the optimal integer answer had been found. However, it could still be stated as to within what per cent the feasible solution was of the best possible integer solution. Therefore each method was evaluated on the basis of CPU time and within what per cent its feasible solution was of the best possible integer solution. The best possible integer solution would be obtained from rounding up the optimal non-integer solution obtained from step one of the solution procedure.

Problem 1

Schedule 4 jobs on 3 machine classes in 15 time periods

Processing Time

Mach				
Job	1	2	3	Total
1	1	1	1	3
2	0	2	2	4
3	3	2	2	7
4	2	3	2	7

Technological Order

Process			
Job	1	2	3
1	1	2	3
2	3	2	1
3	2	3	1
4	2	1	3

Start Time-Due Date

Job	Earliest Start Time	Due Date
1	0	13
2	0	15
3	0	15
4	0	14

Machines Needed For Each Job in Each Class

Class			
Job	1	2	3
1	1	1	1
2	0	2	1
3	3	1	2
4	1	2	1

Machines Available in Each Machine Class

Class	Avail.
1	3
2	2
3	2

FIGURE V-1

Problem 2

Schedule 6 jobs on 3 machine classes in 15 time periods

Processing Time

Mach Job	1	2	3	Total
1	3	3	3	9
2	4	5	0	9
3	3	4	4	11
4	5	0	6	11
5	4	5	4	13
6	5	4	5	14

Technological Order

Process Job	1	2	3
1	3	2	1
2	2	1	3
3	1	2	3
4	3	1	2
5	2	3	1
6	3	2	1

Start Time-Due Date

Job	Earliest Start Time	Due Date
1	0	14
2	0	15
3	0	13
4	0	15
5	0	14
6	0	15

Machines Needed For Each Job in Each Class

Class Job	1	2	3
1	1	2	1
2	1	1	0
3	2	1	1
4	1	0	2
5	2	1	1
6	1	1	1

Machines Available in Each Machine Class

Class	Avail.
1	4
2	3
3	4

FIGURE V-2

Problem 3

Schedule 5 jobs on 3 machine classes in 15 time periods

Processing Time

Mach Job	1	2	3	Total
1	1	2	0	3
2	2	2	1	5
3	5	6	0	11
4	5	3	4	12
5	4	5	4	13

Technological Order

Process Job	1	2	3
1	2	1	3
2	3	1	2
3	1	2	3
4	2	3	1
5	3	2	1

Start Time-Due Date

Job	Earliest Start Time	Due Date
1	0	15
2	0	13
3	0	15
4	0	14
5	0	15

Machines Needed For Each Job in Each Class

Class Job	1	2	3
1	1	1	0
2	1	1	1
3	2	1	0
4	1	2	1
5	1	1	1

Machines Available in Each Machine Class

Class	Avail.
1	2
2	3
3	3

FIGURE V-3

Problem 4

Schedule 6 jobs on 3 machine classes in 15 time periods

Processing Time

Job \ Mach	1	2	3	Total
1	2	2	2	6
2	0	3	2	5
3	3	3	3	9
4	4	3	3	10
5	4	3	5	12
6	5	4	4	13

Technological Order

Job \ Process	1	2	3
1	2	1	3
2	3	2	1
3	3	1	2
4	2	3	1
5	3	2	1
6	1	2	3

Start Time-Due Date

Job	Earliest Start Time	Due Date
1	0	15
2	0	15
3	0	15
4	0	14
5	0	15
6	0	15

Machines Needed For Each Job in Each Class

Job \ Class	1	2	3
1	1	2	1
2	0	2	1
3	1	1	1
4	2	1	2
5	1	1	2
6	1	1	1

Machines Available in Each Machine Class

Class	Avail.
1	3
2	3
3	3

FIGURE V-4

VI RESULTS

The four test problems were run using each of the eight job ordering methods discussed in Chapter IV. The results are presented in Figure VI-1. This figure gives the optimal non-integer solution to each of the four problems and the Central Processing Unit (CPU) time required for each method to find a feasible zero-one solution. This feasible zero-one solution, and what per cent of the best possible optimal solution it is, are also presented in Figure VI-1. The per cent of the best possible optimal solution is defined as

$$\left(1 - \frac{Z_{feas.} - Z_{opt.}}{Z_{opt.}}\right) \times 100.$$

Problem one, which only had jobs with short processing times, was either solved to a feasible zero-one solution in approximately five minutes, or no feasible zero-one solution was found in thirty minutes. Problem two, which only had jobs with long processing times, was solved to a feasible zero-one solution by each of the methods in less than twelve minutes. The Minimum Slack Time method found the best possible optimal zero-one solution in 8.9 minutes.

Problems three and four, the ones with a mixture of long and short processing time jobs, are of most interest, since they are the more typical problems found in everyday operations. Examining the methods used to solve these two problems shows that the Largest Number of Operations method failed to find a solution in either case. Also, the Random method, the Shortest Processing Time method, and the Shortest Processing Time Considering Due Dates method all failed to find a solution to problem three. While the other methods found solutions to both problems the Earliest Due Date method took the longest time to find a solution to problem four, while the Minimum Slack Time method was next longest, and the Largest Number of Operations With The Longest Processing Time method was fifth longest. Comparing the results, the Largest Dual Variable method appears to have found a feasible zero-one solution as close to the best possible solution as any other method, and, on the average, its solution time was better than any other method.

Since the Largest Dual Variable method had the best overall results on problems three and four it was decided to formulate additional problems to be solved using this method and compare them with the Random method. The additional problem to be solved are variations of problems three and four. The technological order for these two problems was varied to produce seven additional problems from the originals. These additional problems will be denoted 3A, 3B, 3C, 3D, 4A, 4B, and 4C. The changed technological orders are shown in Figure VI-2. The results of these additional runs are shown in Figure VI-3. The results are expressed in the same form as in Figure VI-1.

It should be noted in Figure VI-3 that there were four instances when ~~the~~ Random method was unable to find a feasible zero-one solution in thirty minutes of CPU time. Also, the Largest Dual

Variable method took from 2.71 minutes to 30.0 minutes to find a feasible zero-one solution. It should be further noted that for three problems the Largest Dual Variable method found the best possible optimal zero-one solution.

Examining the results for the Largest Dual Variable method from problems three and four, and their seven variations, some statistical statements can be made. Using the Wilks' distribution-free α - tolerance limits with probability level $1-\epsilon$, it can be said that for a sample size of nine

$$P(2.71 \leq x \leq 30.0) \geq .60$$

can be asserted with probability .90. In other words, it can be stated that a feasible zero-one solution will be found 90% of the time in 60% of the range between 2.71 minutes and 30.0 minutes. For the Random method it can be asserted with probability .50 that

$$P(4.51 \leq x \leq 19.37) \geq .60$$

for a sample size of four, since the remaining five problems did not yield any feasible zero-one solution in the alotted thirty minutes of CPU time. This shows that the Largest Dual Variable method has a much higher probability of covering the same per cent of a slightly larger range than the Random method.

Further nonparametric tests could be used to statistically test the hypothesis that the solution time using the Largest Dual Variable method is significantly less than the Random method, but such tests are not necessary because the solution time of the Largest

Dual Variable method was less for every problem and in some problems the difference was large. Problems 3, 3B, and 3D offer convincing evidence of the effectiveness of the Largest Dual Variable method over the Random method.

While the Largest Dual Variable method found feasible zero-one solutions in less time than the Random method, its feasible zero-one solution was also the same per cent, or higher, of the best possible optimal solution as the Random method. With either method, however, it is not possible to predict the results beforehand. The unpredictability of the results is not unique to these methods. All algorithms which are enumerative in nature seem to have the same unpredictability. It is known that in general the time required to solve a problem increases as the number of possible combinations increases. The time also greatly depends upon whether or not the optimal solution lies close to the minimum of the objective function. Since this is seldom known beforehand, the results will always be somewhat unpredictable.

	Non-Integer Optimal Answer	Largest Dual Variable	Shortest Processing Time (SPT)	Earliest Due Date	Minimum Slack Time	SPT Considering Due Dates	Largest Number of Operations (LNO)	LNO With Longest Processing Time	Random
Problem 1	8.5								
Feasible Solution		*	10	*	*	10	10	*	10
CPU Time		30	5.15	30	30	4.55	5.11	30	5.18
% of Optimal			89			89	89		89
Problem 2	10.5								
Feasible Solution		15	15	13	11	15	15	15	13
CPU Time		5.35	1.92	7.94	8.9	11.94	1.93	10.64	5.71
% of Optimal		64	64	82	100	64	64	64	82
Problem 3	10.0								
Feasible Solution		13	*	13	17	*	*	14	*
CPU Time		13.60	30	13.38	19.43	30	30	3.29	30
% of Optimal		70		70	30			60	
Problem 4	9.5								
Feasible Solution		11	18.0	11.0	11.0	18.0	*	11.0	11.0
CPU Time(min.)		4.25	13.31	20.94	20.75	13.48	30	20.68	4.51
% of Optimal		90	20	90	90	20		90	90

* No feasible solution found in 30 minutes of CPU time.

FIGURE VI-1

3A

Process			
Job	1	2	3
1	2	1	3
2	1	2	3
3	1	2	3
4	2	3	1
5	3	2	1

4A

Process			
Job	1	2	3
1	2	1	3
2	3	2	1
3	1	3	2
4	2	3	1
5	3	2	1
6	1	2	3

<u>3B</u>

Process			
Job	1	2	3
1	2	1	3
2	2	1	3
3	1	2	3
4	2	3	1
5	3	2	1

<u>4B</u>

Process			
Job	1	2	3
1	2	1	3
2	3	2	1
3	2	3	1
4	2	3	1
5	3	2	1
6	1	2	3

<u>3C</u>

Process			
Job	1	2	3
1	2	1	3
2	2	3	1
3	1	2	3
4	2	3	1
5	3	2	1

<u>4C</u>

Process			
Job	1	2	3
1	2	1	3
2	3	2	1
3	3	2	1
4	2	3	1
5	3	2	1
6	1	2	3

<u>3D</u>

Process			
Job	1	2	3
1	2	1	3
2	3	2	1
3	1	2	3
4	2	3	1
5	3	2	1

Modified Technological Orders for Problems Three and Four

FIGURE VI-2

	Non-Integer Optimal Answer	Largest Dual Variable	Random
Problem 3A	9.8		
Feasible Solution		15	*
CPU Time		25.74	30
% of Optimal		50	
Problem 3B	11.5		
Feasible Solution		12	*
CPU Time		2.71	30
% of Optimal		100	
Problem 3C	11.5		
Feasible Solution		14	15
CPU Time		4.01	10.64
% of Optimal		83	75
Problem 3D	10.5		
Feasible Solution		11	*
CPU Time		2.94	30
% of Optimal		100	
Problem 4A	9.5		
Feasible Solution		10	*
CPU Time		30.0	30
% of Optimal		100	
Problem 4B	9.75		
Feasible Solution		12	12
CPU Time		19.24	19.37
% of Optimal		80	80
Problem 4C	9.5		
Feasible Solution		14	14
CPU Time		6.83	6.94
% of Optimal		60	60

* No Feasible Solution found in 30 minutes of CPU time.

FIGURE VI-3

VII CONCLUSIONS

The purpose of the "job shop" problem is to schedule a set of jobs through a number of machines in an optimum way. The job schedule may vary between machines and the required machine technological ordering may vary between jobs. Thus a job-passing situation is described as opposed to the no-passing situation for an assembly line or flow shop. An optimum solution for a problem is defined as the solution which minimizes the elapsed time between that instant the first machine begins to process its schedule and that instant the last machine running completes its schedule.

In this paper a computational procedure employing the branch-and-bound technique with an imbedded linear program has been investigated concerning which method should be used to bring jobs into solution. These eight methods of job ordering were tested for four sample problems and two of the methods were tested with seven additional problems. The algorithm used to find the solution has been explained and a FORTRAN computer program has been written to perform the operations. The procedure used has allowed for a solution that is either optimum, within any desired fraction of optimum, or only zero-one feasible.

The techniques employed have allowed for certain practical contingencies. The possibility that some of the jobs skip some of the machines is allowable by assigning zero processing times at the appropriate machines. This in turn allows routine machine maintenance to be scheduled as a dummy job with the processing times set to zero at all machines. If all machines are due for maintenance, a single dummy job may be used with the required maintenance time entered as processing time. In addition, constraint inputs are provided to delay the start of a particular job or to account for an earlier due date for some job. Thus, it is not necessary for all jobs to be ready simultaneously to begin processing or to require the same number of time periods for all jobs.

Examination of the results of the eight job ordering methods for the first four problems indicated that the time necessary to find an optimal zero-one solution would, in many cases, be prohibitive. It was therefore decided to search only for zero-one feasibility. The times required to find zero-one feasibility varied greatly from a shortest time of 1.92 minutes to no feasible solution found in 30.0 minutes. The best overall performance with the four problems was shown by the Largest Dual Variable method. It found a zero-one feasible solution in a reasonable amount of time for three of the four problems, but then no method found feasible solutions for all four problems. Further comparison of the Largest Dual Variable method with the Random method indicated that it had a much higher probability of covering the same per cent of a slightly larger range than did the Random method.

It is important to realize that, while the computational procedure investigated here with the eight job ordering methods was applied to three types of problems within the scheduling area, its true usefulness can only be determined through application to actual problems. It should be possible to introduce special variations to the procedure which will greatly improve its efficiency when applied to specific problems of a particular structure and size. Some further areas of investigation are discussed in the next chapter.

VIII RECOMMENDATIONS FOR FURTHER STUDY

Several aspects of the computational procedure are open for further investigation. First it may be possible to bring in more than one job at a time. Presently only one job is forced to solution at a time. It may be possible to keep track of the variables in such a manner as to enable more than one job to be forced into solution at one time.

Furthermore, it is currently required to solve a linear program at each node. Is it possible, by examination of a previous node, or nodes, to determine what the solution is at a node without solving a linear program? Can some of the bounds be estimated instead of calculated at each node? A reduction in the number of linear programming problems solved at each level can cause a significant reduction in the solution time.

Another area for investigation is how to work with a packed matrix in a linear program solver. A lot of computer core space is required to store the unpacked matrix form of the problem along with the packed form. The major portion of the core requirements for this procedure is devoted to the unpacked matrix. This forces the program to store much of its results on disks, outside of the computer core. The constant use of these disks adds greatly to the amount of time needed to solve a problem.

Other investigation might be in the area of the decomposition of large problems into smaller subsets that would be solved in tandem using the procedure. One last suggestion is concerned with the preconditioning of the jobs of a particular problem. However, this presupposes that certain parameters have predictable computing time characteristics. Therefore, the more generalized area of further investigation would be in the area of identifying those parameters with predictable characteristics.

APPENDIX A

An Illustrative Example Solved Using The Computational Procedure

AN ILLUSTRATIVE SAMPLE PROBLEM

Consider the following hypothetical problem which is shown in tableau form:

	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}	x_{31}	x_{32}	x_{33}	
MIN Z =	1	2	3	2	4	5	1	3	6	
Subject to;										
	1	1	1							=1
				1	1	1				=1
							1	1	1	=1
	2			3			4			5
		2			3			4		5
			2			3			4	5
	2			1			1			2
		2			1			1		2
			2			1			1	2

Step 1:

Initial solution is 6; some x_{ij} non-integer (Node 1).

Step $2^{(1)}$:

Select job 1 to be brought into solution.

Step $3^{(1)}$:

Set $x_{11} = 1$. Solution is 8.66; some x_{ij} non-integer (Node 2).

Step $4^{(1)}$:

Set $x_{12} = 1$. Solution is 7; some x_{ij} non-integer (Node 3).

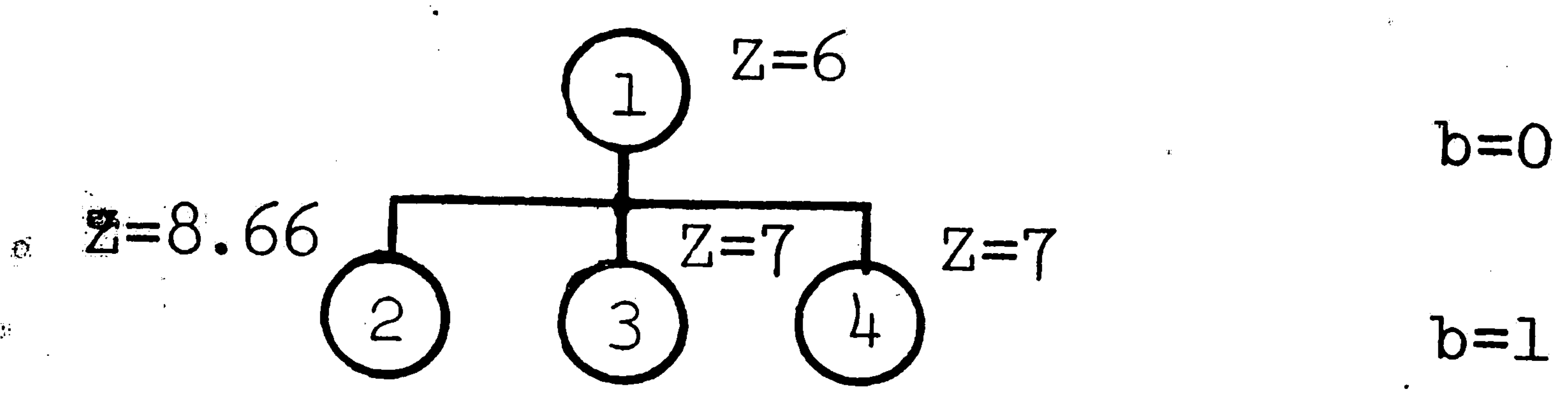
Step $5^{(1)}$:

Set $x_{13} = 1$. Solution is 7; some x_{ij} non-integer (Node 4).

Step $6^{(1)}$:

No $[0,1]$ solutions.

Tree structure at this point is as follows:



Step $7^{(1)}$:

$Z = 7$ is lowest value. Tie between nodes 3 and 4. Since neither is a $[0,1]$ solution, an optimal solution has not been found. Tie will be broken arbitrarily and will branch to the next level from node 3. Set $b_m=1$.

Step $8^{(1)}$:

$L=0$. Set $b = b_m+1 = 2$.

Step $9^{(1)}$:

$L=0$, go to step 2.

Step $2^{(2)}$:

Select job 2 to be brought into solution.

Step $3^{(2)}$:

Set $x_{21}=1$. Solution is 7.5; some x_{ij} non-integer (Node 5).

Step $4^{(2)}$:

Set $x_{22}=1$. Solution is infeasible, $Z = \infty$ (Node 6).

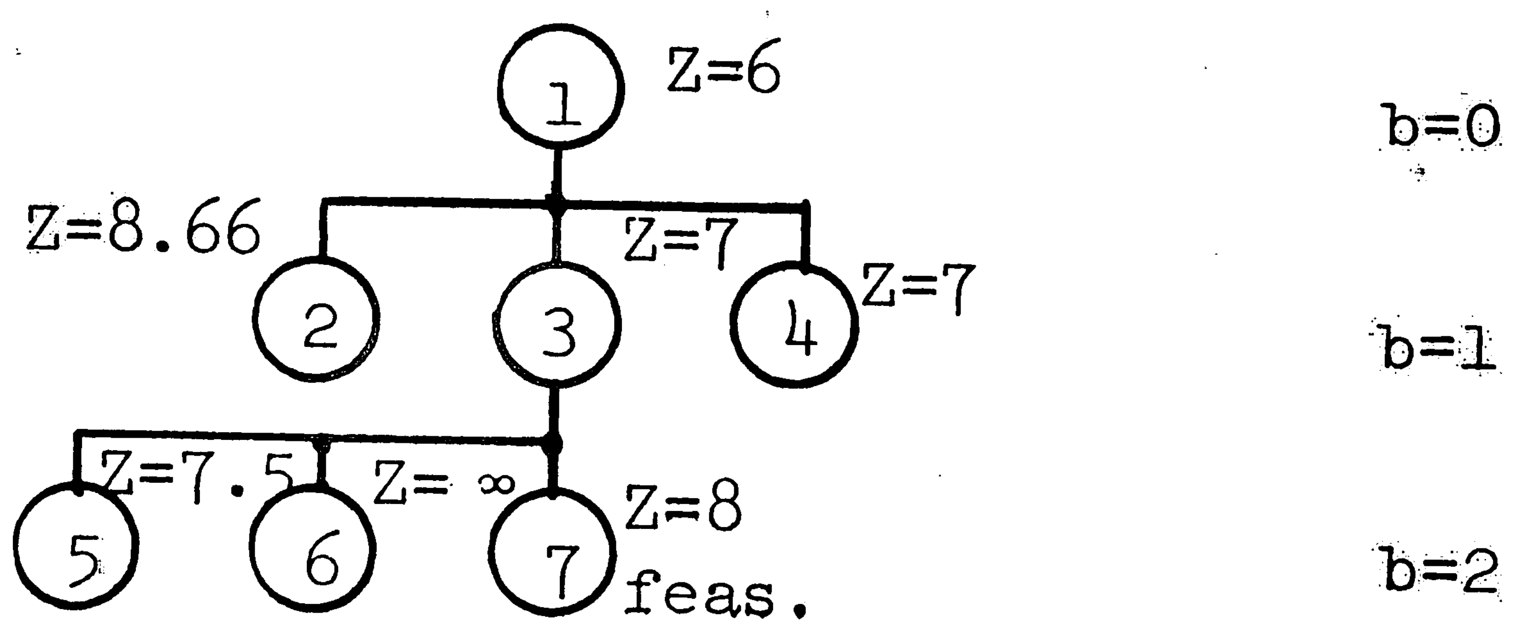
Step $5^{(2)}$:

Set $x_{23}=1$. Solution is 8.0, all x_{ij} integers (Node 7).

Step $6^{(2)}$:

Label node 7 as a feasible solution.

Tree structure at this point is as follows:



Step $7^{(2)}$:

Z = 7 is lowest value. This value occurs at node 4. Since the solution at node 4 is not [0,1] , an optimal solution has not been found. Branch to the next level from node 4. Set $b_m=1$.

Step $8^{(2)}$:

L=0. Set $b = b_m+1 = 2$.

Step $9^{(2)}$:

L=0, go to step 2

Step $2^{(3)}$:

Select job 2 to be brought into solution.

Step $3^{(3)}$:

Set $x_{21}=1$. Solution is 7.0, some x_{ij} non-integer (Node 8).

Step $4^{(3)}$:

Set $x_{22}=1$. Solution is 8.0, all x_{ij} integer (Node 9).

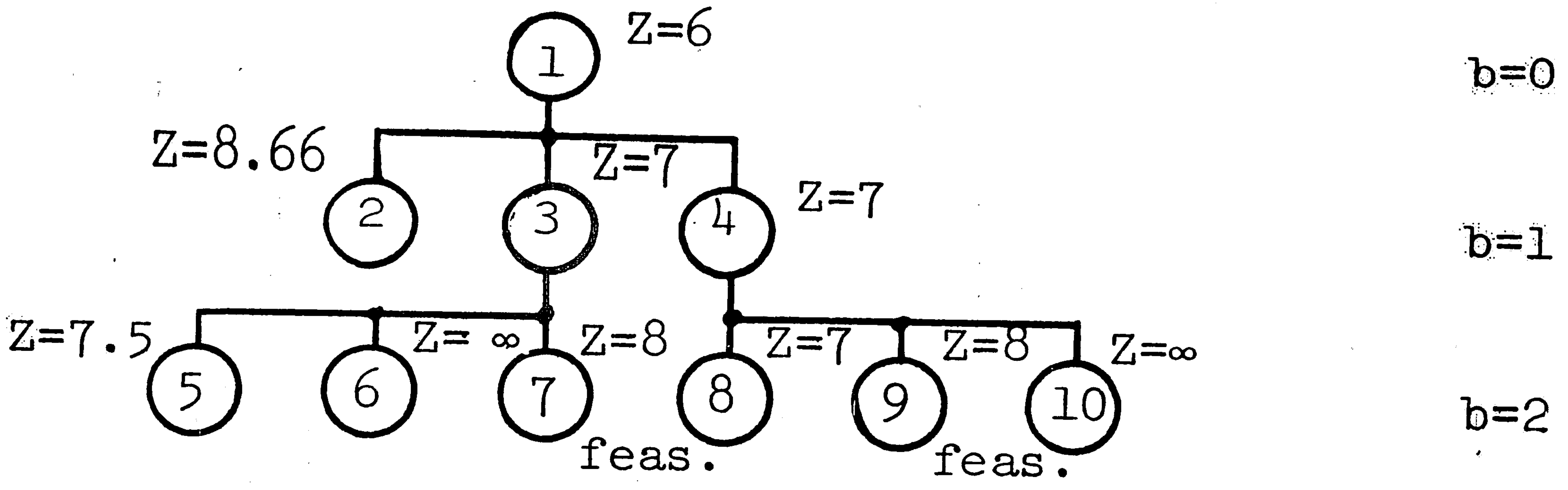
Step $5^{(3)}$:

Set $x_{23}=1$. Solution is infeasible, Z = ∞ (Node 10).

Step $6^{(3)}$:

Label node 9 as feasible solution.

Tree structure at this point is as follows:



Step $7^{(3)}$:

Z = 7 is lowest value. This value occurs at node 8. Since the solution at node 8 is not $[0,1]$, an optimal solution has not been found. Branch to the next level from node 8. Set $b_m=2$.

Step $8^{(3)}$:

L=0. Set $b = b_m+1 = 3$.

Step $9^{(3)}$:

L=0, go to step 2

Step $2^{(4)}$:

Select job 3 to be brought into solution.

Step $3^{(4)}$:

Set $x_{31}=1$. Solution is infeasible, $Z = \infty$ (Node 11).

Step $4^{(4)}$:

Set $x_{32}=1$. Solution is 8.0, all x_{ij} integers (Node 12).

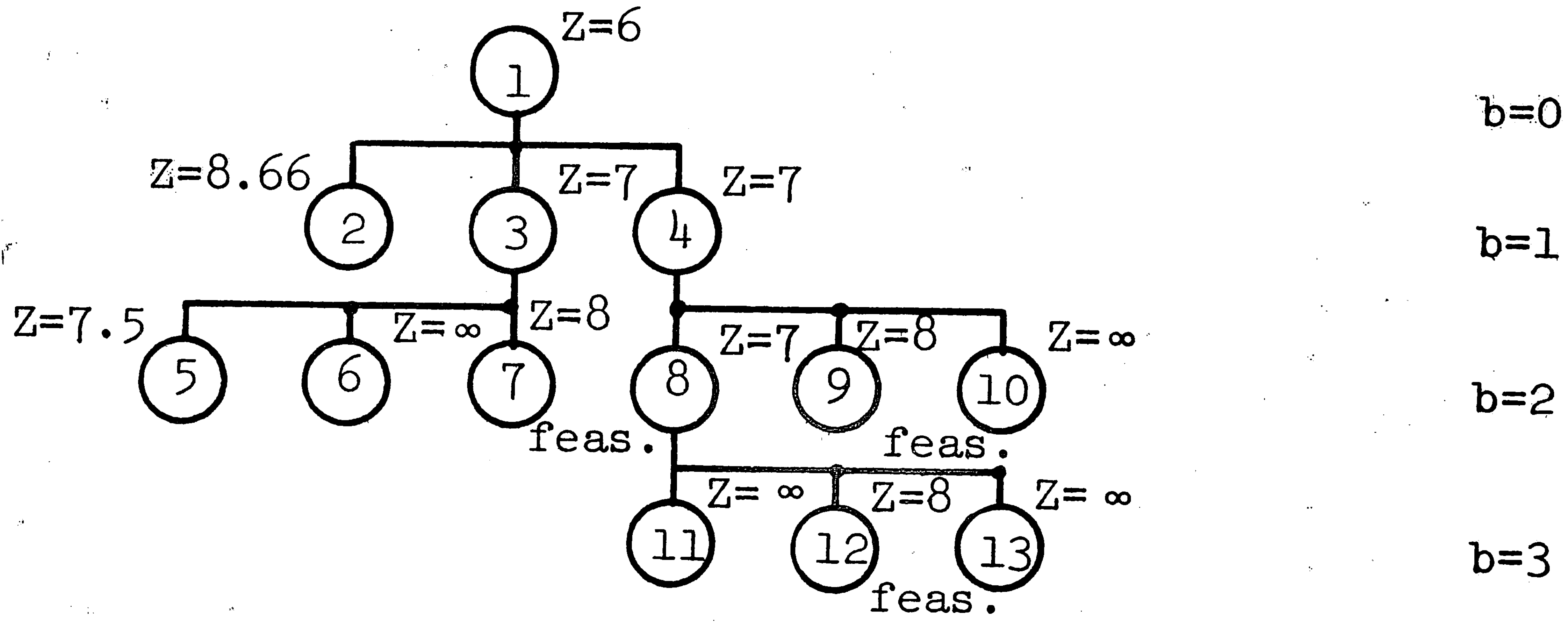
Step $5^{(4)}$:

Set x_{33}=1. Solution is infeasible, Z = ∞ (Node 13).

Step $6^{(4)}$:

Label node 12 as feasible solution.

Tree structure at this point is as follows:



Step $7^{(4)}$:

Z = 7.5 is lowest value. This value occurs at node 5. Since the solution at node 5 is not [0,1] , an optimal solution has not been found. Branch to the next level from node 5. Set b_m=2.

Step $8^{(4)}$:

L=0. Set b = b_m+1 = 3.

Step $9^{(4)}$:

L=0, go to step 2.

Step $2^{(5)}$:

Select job 3 to be brought into solution.

Step $3^{(5)}$:

Set x_{31}=1. Solution is infeasible, Z = ∞ (Node 14).

Step $4^{(5)}$:

Set $x_{32}=1$. Solution is infeasible, $Z = \infty$ (Node 15).

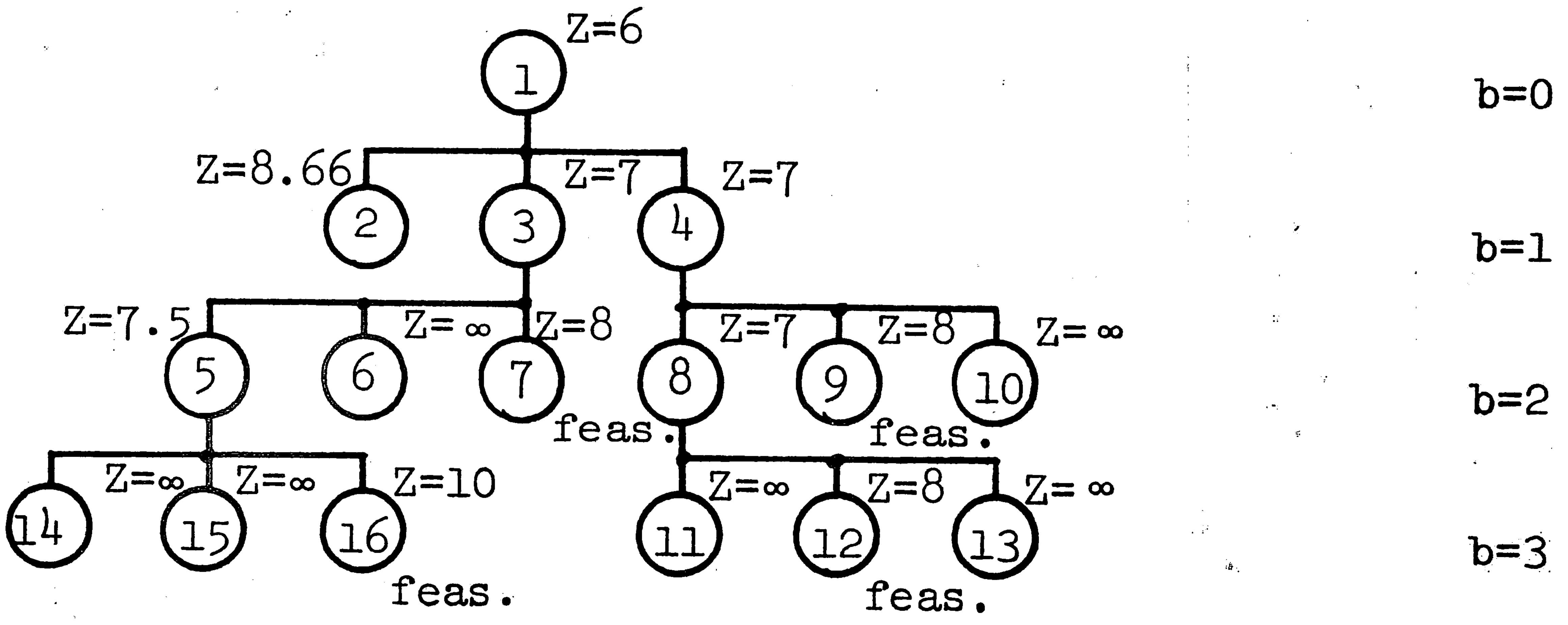
Step $5^{(5)}$:

Set $x_{33}=1$. Solution is 10.0, all x_{ij} integers (Node 16).

Step $6^{(5)}$:

Label node 16 as feasible solution.

Tree structure at this point is as follows:



Step $7^{(5)}$:

$Z = 8$ is lowest value. This value occurs at nodes 7, 9, and 12. Since the solutions at either of these nodes is feasible, $[0,1]$, an optimal solution has been found. Either of the three solutions is optimal.

The three optimal solutions are:

	Node 7	Node 9	Node 12
x_{11}	0	0	0
x_{12}	1	0	0
x_{13}	0	1	1
x_{21}	0	0	1
x_{22}	0	1	0
x_{23}	1	0	0
x_{31}	1	1	0
x_{32}	0	0	1
x_{33}	0	0	0

APPENDIX B

Program Listings

```
C
C*************************************************************
C     ****** PROGEN ******
C*************************************************************
C     PROGRAM USED TO GENERATE DATA FOR SOLUTE IN PROPER
C         FORMAT
C*************************************************************
C     BY    MERRICK H. COOPERMAN
C*************************************************************
C     ALL INPUT IS READ IN ON A 16I5 FORMAT
C*************************************************************
C
        DIMENSION MDUDA(100),IDUDA(10),ITPRC(10),
     1            NM(10),ITORD(10,10),IPROC(10,10),
     2            ICSTGN(10000),IEAST(10),IROW(1000),
     3            IPACK(20000),LNM(10),YVAL(1000),
     4            NRHS(20000),KNM(10,10)
C
C*************************************************************
C
C    NJOB=NUMBER OF JOBS
C    IT=NUMBER OF TIME PERIODS FOR SCHEDULING
C    LNM(J)= NUMBER OF MACHINES AVAILABLE IN CLASS J
C    KNM(I,J)=NUMBER OF MACHINES IN CLASS J NEEDED ON JOB I
C    MACH=MAX NUMBER OF MACHINE CLASSES ON ALL JOBS
C    ITORD(I,J)=TECHNOLOGICAL ORDER FOR JOB I , POSITION J
C    IPROC(I,J)=PROCESSING TIME FOR JOB I ON MACHINE J
C    ITPRC(I)=TOTAL PROCESSING TIME ON JOB I
C    IDUDA(I)=DUE DATE FOR JOB I
C    IEAST(I)=EARLIEST START TIME FOR JOB I
C       NENDCL=  10+NCOL
C
C
C*************************************************************
C    INITIALIZE AND READ IN DATA
C*************************************************************
C
C
        IOUT=11
        JOUT=12
        IN=1
        READ(IN,1001) NJOB,IT, MACH
        READ(IN,1001)(LNM(J),J=1,MACH)
        DO 600 I=1,NJOB
        NM(I)=MACH
        DO 600 J=1,MACH
  600   IPROC(I,J)=0
        DO 1 I=1,NJOB
        NMI=NM(I)
  1     READ(IN,1001) (ITORD(I,J),J=1,NMI)
```

```
      DO 2 I=1,NJOB
      NMI=NM(I)
 2    READ(IN,1001) (IPROC(I,J),J=1,NMI)
      DO 3 I=1,NJOB
      NMI=NM(I)
 3    READ(IN,1001)(KNM(I,J),J=1,NMI)
      READ(IN,1001)(ITPRC(I),I=1,NJOB   )
      READ(IN,1001)(IDUDA(I),I=1,NJOB)
      READ(IN,1001)(IEAST(I),I=1,NJOB)
      XVAL=1.
      ICOL=0
C
C*****************************************************
C  GENERATE SCHEDULE FOR EACH JOB
C*****************************************************
C
      DO 100 I=1,NJOB
      IFRST=1
      IKONT=0
      IKONT=IKONT+1
      ICOL=ICOL+1
      ITP=ITPRC(I)+1
      DO 20 II=1,ITP
20    IROW(II)=0
      NMI=NM(I)
C
C*****************************************************
C  CALCULATE ROW NUMBER CORRESPONDING TO DUE DATE
C*****************************************************
C
      IDEAD=NJOB+(ITORD(I,NMI)-1)*IT+IDUDA(I)
      NNMI=NMI
      ITO=ITORD(I,NNMI)
35    IF(IPROC(I,ITO))34,33,34
33    NNMI=NNMI-1
      ITO=ITORD(I,NNMI)
      IDEAD=NJOB+(ITORD(I,NNMI)-1)*IT+IDUDA(I)
      GO TO 35
C
C*****************************************************
C  GENERATE FIRST SCHEDULE FOR FIRST MACHINE
C*****************************************************
C
34    IROW(1)=I
      IROW(2)=NJOB+1+IEAST(I)+IT*(ITORD(I,1)-1)
      YVAL(1)=1.
      YVAL(2)=KNM(I,ITORD(I,1))*XVAL
      KK=2
      ITO=ITORD(I,1)
      IF(IPROC(I,ITO)-1)21,21,22
22    IP=IPROC(I,ITO)
```

```
C
C***********************************************************
C   IF PROC TIME GREATER THAN ONE,GENERATE ADDITIONAL
C        ENTRIES FOR FIRST MACHINE
C***********************************************************
C
       DO 23 M=2,IP
       KK=KK+1
       IROW(KK)=IROW(KK-1)+1
       YVAL(KK)=KNM(I,ITORD(I,1))*XVAL
23     CONTINUE
C
C***********************************************************
C   GENERATE FIRST SCHEDULE FOR REMAINING MACHINES
C***********************************************************
C
21     KK=KK+1
       DO 24 M=2,NMI
       IROW(KK)=IROW(KK-1)+1+IT*(ITORD(I,M)-ITORD(I,M-1))
       YVAL(KK)=KNM(I,ITORD(I,M))*XVAL
       ITO=ITORD(I,M)
       IF(IPROC(I,ITO)-1)25,25,26
26     IP=IPROC(I,ITO)
C
C***********************************************************
C   IF PROC TIME GREATER THAN ONE, GENERATE ADDITIONAL
C        ENTRIES FOR REMAINING MACHINES
C***********************************************************
C
       DO 27 MM=2,IP
       KK=KK+1
       IROW(KK)=IROW(KK-1)+1
       YVAL(KK)=KNM(I,ITORD(I,M))*XVAL
27     CONTINUE
25     KK=KK+1
       IF((M.EQ.NMI).AND.(IFRST.EQ.1))IBEGIN=IROW(ITP)
24     CONTINUE
       IFRST=2
C
C***********************************************************
C   PRINT OUT FIRST SCHEDULE
C***********************************************************
C
       DO 28 M=1,ITP
28     WRITE(IOUT,1002)ICOL,IROW(M),YVAL(M)
       ICSTGN(ICOL)=IROW(ITP)-IBEGIN+1
C
C***********************************************************
C   CALCULATE ROW NUMBERS CORRESPONDING TO LATEST
C        POSSIBLE SCHEDULING TIME PERIOD
C***********************************************************
```

```
C
      DO 40 M = 1,NMI
   40 MDUDA(M) = 0
      NNMI=NMI
      ITO=ITORD(I,NNMI)
   38 IF(IPROC(I,ITO))37,36,37
   36 NNMI=NNMI-1
      ITO=ITORD(I,NNMI)
      GO TO 38
   37 MDUDA(ITO)=IDEAD
      NNMI=NNMI-1
      DO 10 J=1,NNMI
      JJ=NNMI-J+1
      ITO1=ITORD(I,JJ)
      ITO=ITORD(I,JJ+1)
      MDUDA(ITO1)=MDUDA(ITO)-IT*(ITO-ITO1)-IPROC(I,ITO)
   10 CONTINUE
C
C*************************************************************
C  GENERATE ADDITIONAL SCHEDULES FOR JOB I
C*************************************************************
C
   17 K=0
      NMII=NMI
      ITPR=ITP
   12 ITO=ITORD(I,NMII)
C
C*************************************************************
C  CHECK IF ALL SCHEDULES HAVE BEEN GENERATED
C*************************************************************
C
      IF(IROW(ITPR)-MDUDA(ITO))11,16,16
   16 ITPR=ITPR-IPROC(I,ITO)
      NMII=NMII-1
      ITO=ITORD(I,NMII)
      K=K+1
      IF(K-NMI)12,105,12
   11 CONTINUE
      ITO=ITORD(I,NMII)
      ITPR=ITPR-IPROC(I,ITO)+1
      IKONT=IKONT+1
      ICOL=ICOL+1
C
C*************************************************************
C  GENERATE NEXT SCHEDULE
C*************************************************************
C
      DO 13 J = NMII,NMI
      IF(J-NMII)19,18,19
   18 IROW(ITPR)=IROW(ITPR)+1
      GO TO 30
```

```
19    IROW(ITPR)=IROW(ITPR-1)+1+IT*(ITORD(I,J)-
     1  ITORD(I,J-1))
30    ITO=ITORD(I,J)
      IF(IPROC(I,ITO)-1)31,15,31
31    IP=IPROC(I,ITO)
      DO 14 JJ=2,IP
      ITPR=ITPR+1
      IROW(ITPR)=IROW(ITPR-1)+1
14    CONTINUE
15    ITPR=ITPR+1
13    CONTINUE
C
C*************************************************************
C   PRINT OUT GENERATED SCHEDULE AND RETURN TO
C       CHECK SOLUTION
C*************************************************************
C
      DO 32  JJ=1,ITP
32    WRITE(IOUT,1002)ICOL,IROW(JJ),YVAL(JJ)
      ICSTGN(ICOL)=IROW(ITP)-IBEGIN+1
      GO TO 17
105   CONTINUE
      IDUDA(I)=IKONT
100   CONTINUE
C
C*************************************************************
C   DELETE ROWS WITH ALL ZEROES FROM MATRIX
C
C   PLACE MATRIX IN A PACKED FORM TO REDUCE CORE SPACE
C*************************************************************
C
      NENDCL=ICOL+10
      WRITE(IOUT,1002)NENDCL
      ENDFILE IOUT
      REWIND IOUT
      ITP=ITPRC(1)
      DO 211 J=1,NJOB
      IF(ITP-ITPRC(J))212,211,211
212   ITP=ITPRC(J)
211   CONTINUE
      MP=ICOL*ITP
      DO 7000 II=1,MP
7000  IPACK(II)=0
7010  READ(IOUT,2011)II,JJ,VAL
      IF(II.EQ.NENDCL)GOTO7100
      IPACK(JJ)=1
      GOTO7010
7100  CONTINUE
      M=0
      K=0
      DJ 7020 IP=1,MP
      IF(IPACK(IP).EQ.0) GO TO 7021
      K=K+1
      IPACK(IP)=K
      GO TO 7020
 7021 M=M+1
      NRHS(M)=IP
 7020 CONTINUE
C
C*************************************************************************
C     ARRANGE RIGHT HAND SIDE TO CORRESPUND WITH PACKING
C*************************************************************************
C
      DO 213 I=1,M
      LVAL=NRHS(I)
      IF(LVAL)213,213,216
  216 MM=I+1
      DO 214 J=MM,M
      IF(LVAL-NRHS(J))214,215,214
  215 NRHS(J)=0
  214 CONTINUE
  213 CONTINUE
      MM=0
      DO 217 I=1,M
      IF(NRHS(I))218,217,218
  218 MM=MM+1
      NRHS(MM)=NRHS(I)
  217 CONTINUE
      WRITE(3,250)(NRHS(J),J=1,MM)
  250 FORMAT(' NRHS',20I5)
      NROW=K
      REWIND IOUT
 7030 READ(IOUT,2011)II,JJ,VAL
      IF(II.EQ.NENDCL)GOTO7050
      JJ=IPACK(JJ)
      WRITE(JOUT,2011)II,JJ,VAL
      GO TO 7030
 7050 WRITE(JOUT,2011)II
      NENDCL=II
      NCOL=NENDCL-10
      ENDFILE JOUT
      REWIND JOUT
      REWIND IOUT
C
C*************************************************************************
C     WRITE DATA FOR SOLUTE ONTO DISK
C*************************************************************************
C
      WRITE(IOUT,556) NENDCL,NJOB,NROW,NCOL,IT,MACH
      WRITE(IOUT,556)(ITPRC(I),I=1,NJOB)
      WRITE(IOUT,556)((IPROC(I,J),I=1,NJOB),J=1,MACH)
      WRITE(IOUT,556)(IDUDA(I),I=1,NJOB)
      WRITE(3,556)(IDUDA(I),I=1,NJOB)
      WRITE(IOUT,556)(IEAST(I),I=1,NJOB)
      DO 389 II=1,NCOL
      WRITE(IOUT,556)ICSTGN(II)
      WRITE(3,556)ICSTGN(II)
389   CONTINUE
7800  READ(JOUT,2011,END=7600)II,JJ,VAL
      WRITE(3,2011)II,JJ,VAL
      WRITE(IOUT,2011)II,JJ,VAL
      GO TO 7800
7600   CONTINUE
      XVAL=1.0
      DO 200 II=1,NJOB
200   WRITE(IOUT,2012) II,XVAL
      NMI=NM(1)
      DO 205 J=1,NJOB
      IF(NMI-NM(J))209,205,205
209   NMI=NM(J)
205   CONTINUE
      KK=NJOB
      KKK=NJOB
      DO 201 J=1,NMI
      DO 202 K=1,IT
      KK=KK+1
      XVAL=0.
      DO 220 N=1,MM
      IF(KK-NRHS(N))220,202,220
220   CONTINUE
      KKK=KKK+1
      XVAL=LNM(J)
      WRITE(IOUT,2012)KKK,XVAL
      WRITE(3    ,2012)KKK,XVAL
202   CONTINUE
201   CONTINUE
      ENDFILE IOUT
556   FORMAT(4I5)
1001  FORMAT(16I5)
1002  FORMAT(2I6,F8.4)
1003  FORMAT('0',10I5)
2011  FORMAT(2I6,F8.4)
2012  FORMAT(6X,I6,F8.4)
      STOP
      END
```

```
C
C*********************************************************
C    ****** SOLUTE ******
C*********************************************************
C    MAIN PROGRAM USED TO SOLVE 0-1 SCHEDULING PROBLEM
C*********************************************************
C    BY   MERRICK H. COOPERMAN
C*********************************************************
C    INPUT READ FROM DISK AND IS GENERATED BY PROGEN
C*********************************************************
C
      INTEGER DUEDAT(10),EAST(10),PERM,RMAT(4000),
     1        CMAT(1000),TIME
      COMMON NJOB,II,JOBSIN(10,10),PERM(10),INFLAG,
     1       A(53,0900),B(53),C(0900),P(0900),KO(6)
      DIMENSION ITPRC(10),IPROC(10,10),ILL(10),
     1          SOLCUR(0900),CURIND(0900),NCOLEL(0900),
     2          RHS(53),COST(0900),PP(0900),IADDR(10),
     3          NSPJ(10),XMAT(4000),X(0900),BOUND(1000),
     4          Y(150),SUMX(1000),INEL(10),IBRAN(1000),
     5          FEAS(0900),CURSOL(0900)
C
C*********************************************************
C    INITIALIZE AND READ IN DATA
C*********************************************************
C
      READ(32,1006) NENDCL, NJOB,NROW,NCOL,IT
      READ(32,1006)(ITPRC(I),I=1,NJOB)
      READ(32,1006)(DUEDAT(I),I=1,NJOB)
      READ(32,1006)(EAST(I),I=1,NJOB)
      IY=1111
      IONE =1
      JOB=1
      DO 1000 I=1,NCOL
      IBRAN(I)=0
 1000 BOUND(I)=10**6
      DO 206 J=1,900
      DO 206 I=1,53
      A(I,J)=0.0
      C(J)=0.0
  206  X(J)=0.0
      DO 5005 I=1,900
      PP(I)=0.0
 5005 P(I)=0.0
      DO 6666 II=1,NCOL
      READ(32,9009)C(II)
 6666 CONTINUE
      II=1
      JJJ=NCOL+1
      JJC=NROW+NCOL
      DO 286 I=JJJ,JJC
```

```
286   X(I)=1.0
      JOBP=1
      FOBJVL=10**6
      IFNUM=0
      INFLAG=0
      TIME=0
      LEVEL=1
      IDUAL=0
      NBOUND=1
      IADDR(1)=1
      INEL(1)=DUEDAT(1)-EAST(1)
      NSPJ(1)=INEL(1)
      DO205 LM=2,NJOB
      LN=LM-1
      IADDR(LM)=IADDR(LN)+INEL(LN)
      INEL(LM)=DUEDAT(LM)-EAST(LM)
  205 NSPJ(LM)=NSPJ(LN)+INEL(LM)
      DO 140 I=1,NJOB
      DO 140 J=1,NJOB
140   JOBSIN(I,J)=-1
      XMINBD=10**6
C
C**********************************************************
C   READ IN MATRIX
C**********************************************************
C
      IICOL=1
      ISCOL=0
      L3=1
      L2=1
      CMAT(L3)=L2
203   READ(32,1002)ICOL,RMAT(L2),XMAT(L2)
      IF(IICOL.EQ.ICOL)GO TO 201
      NCOLEL(L3)=ISCOL
      ISCOL=0
      L3=L3+1
      IICOL=IICOL+1
      CMAT(L3)=L2
201   ISCOL=ISCOL+1
      L2=L2+1
      IF(ICOL.NE.NENDCL)GO TO 203
C
C**********************************************************
C   READ IN RHS VECTOR
C**********************************************************
C
      DO 202 L4=1,NROW
      READ(32,1003)RHS(L4)
202   CONTINUE
      REWIND 32
C
```

```
C**********************************************************
C    GENERATE SLACK VARIABLES
C
C**********************************************************
       IKK=L2-2
       NSCOLB=NCOL+1
       NSCOLE=NCOL+NROW
       ISLK=0
       DO 204 L4=NSCOLB,NSCOLE
       IKK=IKK+1
       NCOLEL(L4)=1
       CMAT(L4)=IKK
       ISLK=ISLK+1
       RMAT(IKK)=ISLK
       C(L4)=100.0
       IF(L4.GT.(NCOL+NJOB))C(L4)=0.0
204    XMAT(IKK)=1.0
       DO 608 IJ=1,NJOB
608    ILL(IJ)=IT-ITPRC(IJ)+1
       IL=1
       KL=1
       JL=0
       KJL=0
       JBOUN=0
       GO TO 207
  200 TIME=1
C
C**********************************************************
C    STEP 2
C    RANK JOBS BY BEST FIT
C    BYPASS SLOTTED JOBS
C**********************************************************
C
207    CONTINUE
C
C**********************************************************
C    THE FORMAT OF CALLING STATEMENT MUST AGREE WITH
C        FORMAT OF SUBROUTINE STATEMENT
C**********************************************************
C
       IF(NBOUND.EQ.1) CALL CHOICE
       WRITE(3,2016)(P(K),K=1,NCOL)
C
C**********************************************************
C    IF THIS IS INITIAL SOLUTION, SKIP STEP 2 AND
C        SOLVE INITIAL PROBLEM
C
C    STEP 1
C**********************************************************
C
       IF(IONE.EQ.1)GOTO142
  300 TIME=TIME+1
      JOB=PERM(JOBP)
      MSPB=1
      IJ=JOB-1
      IF(IJ.GT.0)MSPB=MSPB+NSPJ(IJ)
      MSPS=NSPJ(JOB)
      ITIME=TIME+MSPB-1
      WRITE(3,2007)JOB,MSPB,MSPS,L2
      WRITE(3,2009)(PERM(I),I=1,NJOB)
C
C*********************************************************
C   STEPS 3 THRU 5
C*********************************************************
C
      DO 108 I=MSPB,MSPS
      XMAT(CMAT(I))=0.0
      IF(I.EQ.ITIME)XMAT(CMAT(I))=1.0
108   CONTINUE
      X(ITIME)=1.0
      IF(TIME.EQ.1) GO TO 617
      X(MTIME)=0.
617   MTIME=ITIME
C
C*********************************************************
C   REDUCE RHS BY A(;TIME)
C*********************************************************
C
      NBB=CMAT(ITIME)
      NSS=CMAT(ITIME)+NCOLEL(ITIME)-1
      DO 109 I=NBB,NSS
      IF((RHS(RMAT(I))-XMAT(I)).LT.0.0)IDUAL=1
109   CONTINUE
      IF(IDUAL.EQ.1)GO TO 112
C
C*********************************************************
C   PREPARE FOR SOLUTION BY UNPACKING A MATRIX
C*********************************************************
C
142    DO 100 I=1,NSCOLE
      DO 100 J=1,NROW
100   A(J,I)=0.0
      DO 141 I=1,NSCOLE
      NBB=CMAT(I)
      NSS=CMAT(I)+NCOLEL(I)-1
      DO 110 J= NBB,NSS
110   A(RMAT(J),I)=XMAT(J)
  141 CONTINUE
      DO 111 J=1,NROW
      B(J)=RHS(J)
  111 CONTINUE
C
```

```
C*************************************************************
C    SET BASIS AND CALL SIMPLE TO SOLVE PROBLEM
C*************************************************************
C
       INFLAG=1
       CALL SIMPLE (NROW,NSCOLE,X)
       WRITE(3,2006)(KO(K),K=1,6)
       DO304 K=1,NCOL
       IF(KO(1).EQ.0)PP(K)=P(K)
       IF(KO(1).EQ.1)P(K)=PP(K)
       P(K)=ABS(P(K))
   304 CONTINUE
       IF(IONE.EQ.1) GO TO 609
       IF(ILL(JOB).NE.KL) GO TO 615
       JL=JL+1
       KJL=JL
       KL=0
615    IF(IL.LE.ILL(JOB)) GO TO 616
       KL=KL+1+KJL
       KJL=0
       IL=KL
616    IL=IL+1
609    WRITE(3,2016)(P(K),K=1,NCOL)
       GOTO113
   112 KO(1)=1
   113 IDUAL=0
C
C*************************************************************
C    STEP 6
C
C    CHECK FOR 0-1 SOLUTION
C*************************************************************
C
       IF((KO(1).EQ.1).AND.(IONE.EQ.0))GOTO116
       OBJVAL=0.0
       ICNT=0
       SUM=0.
       XNJOB=NJOB
       DO 114 I=1,NCOL
       OBJVAL=OBJVAL+C(I)*X(I)
       SUM=SUM+X(I)
       IF(X(I).LT.0.01)GOTO114
       IF(X(I).GT.0.99)ICNT=ICNT+1
114    CONTINUE
       IF(JBOUN.NE.0) GO TO 618
       IF(SUM.LE.(XNJOB-.1)) GO TO 101
       IBOUND=OBJVAL+.9999
       FBOUND=IBOUND
       JBOUN=1
618    CONTINUE
       WRITE(3,611)OBJVAL
```

```
      WRITE(3,1005)(X(I),I=1,NCOL)
      JILL=0
      IF(OBJVAL.LT.FOBJVL) GO TO 601
      IF(IONE.EQ.1) GO TO 601
      IF(IL.GT.ILL(JOB)) GO TO 601
      JILL=0
      ILLJO=ILL(JOB)
      DO 604 J=IL,ILLJO
      TIME=TIME+1
      JILL=JILL+1
604   CONTINUE
      IL=IL+JILL
      GO TO 115
601    IF(ICNT.LT.NJOB) GO TO 115
      IF(OBJVAL.GE.FOBJVL) GO TO 115
      FOBJVL=OBJVAL
      IFNUM=NBOUND
C
C*********************************************************
C   STORE BEST FEASIBLE SOLUTION
C*********************************************************
C
      DO301 I=1,NCOL
  301 FEAS(I)=X(I)
C
C*********************************************************
C   INSERT "GO TO 302" HERE IF JUST LOOKING FOR 0-1
C       FEASIBILITY
C*********************************************************
C
      GO TO 302
C
C*********************************************************
C   FOR SOLUTION WITHIN X PERCENT OF OPTIMAL INSERT
C      "IF(FOBJVL.GT.(1.X*FBOUND)) GO TO 115"
C
C   NOTE- PRESENTLY SET FOR WITHIN 10 PERCENT
C*********************************************************
C
      IF(FOBJVL.GT.(1.1*FBOUND)) GO TO 115
C
C*********************************************************
C   PRINT OUT SOLUTION
C*********************************************************
C
  302 DO303 J=1,NCOL
      WRITE(3,1008)J,FEAS(J),C(J)
  303 CONTINUE
      WRITE(3,1009)FOBJVL
      STOP
C
```

```
C***********************************************************
C    STORE BOUNDS
C***********************************************************
C
116    OBJVAL=10**6
115    BOUND(NBOUND)=OBJVAL
       IF(SUM.LE.(XNJOB-.1))BOUND(NBOUND)=10**5
       IF(KO(1).GT.) BOUND(NBOUND)=10**7
       WRITE(3,2010)(BOUND(I),I=1,NBOUND)
       IF(IONE.EQ.1)GOTO305
       MINT=TIME
       REWIND 18
       IF(NBOUND.EQ.1) GO TO 606
       IKI=1+(JJJ-1)/20
       IRI=(NBOUND-1)*IKI
       DO 605 J=1,IRI
       READ(18,602)Z,YZ
 605   CONTINUE
606    JILL=JILL+1
       DO 607 J=1,JILL
       CURSOL(1)=NBOUND
       DO 119 I=1,NCOL
119    CURSOL(I+1)=X(I)
       IF(KO(1).EQ.0) GO TO 150
       DO 151 I=1,NCOL
151    CURSOL(I+1)=1.
150    IF(OBJVAL.LT.0.0) CURSOL(2)=-10**3
       WRITE(18,600)(CURSOL(I),I=1,JJJ)
       IF(J.EQ.JILL) GO TO 607
        IF(JILL.GT.1) NBOUND=NBOUND+1
       IF(JILL.GT.1) BOUND(NBOUND)=10**6
607    CONTINUE
       REWIND 18
       JILL=0
C
C***********************************************************
C    CHECK IF LEVEL IS COMPLETE
C***********************************************************
C
       IF(JOB.EQ.1)GO TO 117
       ITCHK=NSPJ(JOB)-NSPJ(JOB-1)
       GO TO 118
117    ITCHK=NSPJ(1)
       WRITE(3,613) TIME,ITCHK
118    IF(TIME.LT.ITCHK) GO TO 650
       WRITE(3,620) FOBJVL
C
C***********************************************************
C    FOR SOLUTION WITHIN X PERCENT OF OPTIMAL INSERT
C        "IF((BK+.001).LT.((1-X)*FOBJVL)) GO TO 134"
C
```

```
C    NOTE- PRESENTLY SET FOR WITHIN 10 PERCENT
C*********************************************************
C
      DO 130 I=1,NBOUND
      KI=BOUND(I)+.9999
      BK=KI
      IF((BK+.001).LT.(.9*FOBJVL)) GO TO 134
130    CONTINUE
      GO TO 302
650   NBOUND=NBOUND+1
      WRITE(3,614)NBOUND,IONE
  305 IF(IONE.EQ.0)GOTO300
      IONE=0
      GOTO207
C
C*********************************************************
C    STEP 8
C
C    SCAN FOR IDENTICAL JOB SLOTS ACROSS LEVEL
C*********************************************************
C
  134 XTIME=TIME
      IL=1
      KL=1
      JL=0
      KJL=0
      XNBOND=ITCHK
      LJ=NBOUND-ITCHK+1
      DO 5012 I=1,NCOL
5012  SUMX(I)=0.0
      DO 120 J=LJ,NBOUND
5010  READ(18,600)(CURSOL(L),L=1,JJJ)
5011  DO 121 I=1,JJJ
      K=CURSOL(1)+.001
      IF(K.NE.J) GO TO 5010
      IF(I.EQ.1) GO TO 121
      SOLCUR(I-1)=CURSOL(I)
      IF(SOLCUR(I-1).GT.0.99)SUMX(I-1)=SUMX(I-1)+1.
121   CONTINUE
      IF(LEVEL.EQ.NJOB) BOUND(J)=10**6
120   CONTINUE
      DO 5013 I=1,NCOL
      CURIND(I)=10**6
      IF((SOLCUR(I).GT.0.0).AND.(SUMX(I).GE.XNBOND))
     1       CURIND(I)=-10**3
 5013 CONTINUE
      REWIND 18
C
C*********************************************************
C    STEP 9
C
```

```
C    SLOT JOBS (SLOT COL.OF CURSOL .LE. 0)
C    STORE IN JOBSIN(LEVEL,JOB NO.)
C*************************************************************
C
      L=2
      JOBSIN(LEVEL,1)=JOB
  149 IF(LEVEL.EQ.1)GOTO144
      DO145 I=1,II
      JOBSIN(LEVEL,L)=JOBSIN(I,1)
  145 L=L+1
C
C*************************************************************
C    STEP 7
C
C    STORE CURSOL
C    CHECK STATUS OF SOLUTION AND SET UP BRANCH CONDITIONS
C*************************************************************
C
  144 DO2011 I=1,NJOB
      WRITE(3,2012)(JOBSIN(J,I),J=1,LEVEL)
 2011 CONTINUE
      JK=NBOUND+1
  131 JK=-1
      XMINBD=10**5
      DO128 J=LJ,NBOUND
      IF(BOUND(J).GE.XMINBD)GOTO128
      XMINBD=BOUND(J)
      JK=J
  128 CONTINUE
      IF(JK.GT.0)GOTO129
      LEVEL=LEVEL-1
      IF(LEVEL.LE.0)GOTO302
      NBOUND=LJ-1
      LJ=NBOUND-(DUEDAT(JOBSIN(LEVEL,1))-
     1        EAST(JOBSIN(LEVEL,1)))+1
      BOUND(IBRAN(LEVEL))=10**6
      WRITE(3,2015)JK,LEVEL,NBOUND,(IBRAN(I),I=1,NBOUND),
     1        LJ,JOBSIN(LEVEL,1),NSPJ(JOBSIN(LEVEL,1))
      GOTO131
  129 IBRAN(LEVEL)=JK
      DO 5015 J=1,NBOUND
      READ(18,600)(CURSOL(L),L=1,JJJ)
      K=CURSOL(1)+.001
      IF(K.EQ.JK) GO TO 5016
5015  CONTINUE
5016  DO 132 I=2,JJJ
132   X(I-1)=CURSOL(I)
      REWIND 18
      WRITE(3,2015)JK,LEVEL,NBOUND,(IBRAN(I),I=1,NBOUND),
     1        LJ,JOBSIN(LEVEL,1),NSPJ(JOBSIN(LEVEL,1))
 2015 FORMAT(' ',20I5/)
```

```
C
C*****************************************************************
C   READ IN NEW INITIAL BASIS
C*****************************************************************
C
      DO138 J=1,NJOB
      XY=0.0
      DO137 JJ=1,NJOB
      IF(J.EQ.JOBSIN(LEVEL,JJ))GOTO143
  137 CONTINUE
      XY=1.0
  143 MSPB=1
      IJJ=J-1
      IF(IJJ.GT.0)MSPB=MSPB+NSPJ(IJJ)
      MSPS=NSPJ(J)
      DO138 K=MSPB,MSPS
      XMAT(CMAT(K))=XY
      IF(X(K).GT.0.999)XMAT(CMAT(K))=1.0
  138 CONTINUE
      LEVEL=LEVEL+1
      II=LEVEL-1
      DO139 I=1,NJOB
139   JOBSIN(LEVEL,I)=-1
      TIME=0
      INFLAG=0
      NBOUND=NBOUND+1
C
C*****************************************************************
C   THE FORMAT OF CALLING STATEMENT MUST AGREE WITH
C       FORMAT OF SUBROUTINE STATEMENT
C*****************************************************************
C
      CALL CHOICE
C
C*****************************************************************
C   RETURN TO STEP 2
C*****************************************************************
C
      GOTO207
101    WRITE(3,102)
102    FORMAT(' NON-INTEGER OPTIMAL ANSWER IS INFEASIBLE'/
     1' PROGRAM TERMINATED')
600   FORMAT(20E20.7)
602   FORMAT(2A4)
603   FORMAT(' ',2A4)
610   FORMAT('OCURSOL',5E20.7)
611   FORMAT('OOBJVAL',F20.7)
613    FORMAT('OTIME',I10,'ITCHK',I10)
614   FORMAT('ONBOUND',I10,'IONE',I10)
 620  FORMAT('OFOBJVL',F20.8)
1001  FORMAT(50F6.0)
```

```
1002  FORMAT(2I6,F8.3)
1003  FORMAT(12X,F8.3)
 1004 FORMAT(20F4.0)
 1005 FORMAT(10F10.5)
1006  FORMAT(4I5)
 1008 FORMAT(' VARIABLE',I6,'  AT LEVEL',F6.3,
     1       '  WITH COST',F8.3)
 1009 FORMAT(' OBJECTIVE VALUE',F8.3)
 2002 FORMAT(' B',21F6.0)
 2004 FORMAT(' C',21F6.0)
 2005 FORMAT(' A',21F6.0)
 2006 FORMAT(' KO',6I10)
 2007 FORMAT(' JOB MSPB MSPS L2',4I6)
 2008 FORMAT(' ',10F10.0)
 2009 FORMAT(' ',20I6)
2010  FORMAT(' BOUND',10E12.4/' ',10E12.4)
 2012 FORMAT(' JOBSIN',20I5)
 2016 FORMAT(' DUAL VARIABLES',10F10.5)
2222  FORMAT('  A MATRIX',10F10.4)
8500  FORMAT(' RMAT ',15I8)
8600  FORMAT(' XMAT ',15F8.2)
8700  FORMAT('  RHS ',15F8.2)
9009  FORMAT(F5.0)
      STOP
      END
```

```
C
C     THE ''SIMPLE'' SUBROUTINE SOLVES THE LINEAR
C     PROGRAMMING PROBLEM-MINIMIZE Z SUBJECT TO AX=B AND
C     X(J) EQUAL TO OR GREATER THAN 0. THE SUBROUTINE IS
C     NUMBER 3384 IN THE SHARE PROGRAM LIBRARY.
C
      SUBROUTINE SIMPLE (MX,NN,KB)
      INTEGER PERM
      COMMON NJOB,II,JOBSIN(10,10),PERM(10)
      COMMON INFLAG,A(53,900),B(53),C(900),P(900),KO(6)
      DIMENSION PE(100),Y(100),E(10000),JH(100),KB(1000),
     1          X(1000)
      EQUIVALENCE (XX,LL)
      LOGICAL FEAS,VER,NEG,TRIG,KQ,ABSC
C
C          SET INITIAL VALUES, SET CONSTANT VALUES
      ITER = 0
      NUMVR = 0
      NUMPV = 0
      DO2001 M=1,100
      PE(M)=0.0
      Y(M)=0.0
      JH(M)=0
 2001 CONTINUE
      DO2006 M=1,10000
 2006 E(M)=0.0
      DO 2002 J=1,1000
      X(J)=0.0
 2002 CONTINUE
 2005 FORMAT(' A',21F6.0)
 2003 FORMAT(' B',21F6.0)
 2004 FORMAT(' C',21F6.0)
      M = MX
      N = NN
      TEXP  =  .5**16
      NCUT  =  4*M + 10
      NVER  =  M/2  +  5
      M2 = M**2
      FEAS = .FALSE.
      IF  (INFLAG.NE.0)  GO TO 1400
C* @NEW@    START  PHASE ONE WITH SINGLETON BASIS
      DO 1402  J = 1,N
        KB(J) = 0
        KQ = .FALSE.
        DO 1403  I = 1,M
          IF (A(I,J).EQ.0.0)  GO TO 1403
          IF (KQ.OR.A(I,J).LT.0.0) GO TO 1402
          KQ = .TRUE.
 1403   CONTINUE
        KB(J) = 1
 1402 CONTINUE
 1400 DO 1401  I = 1,M
        JH (I) = -1
 1401 CONTINUE
C* @VER@    CREATE INVERSE FROM @KB@ AND @JH@      (STEP 7)
 1320 VER = .TRUE.
      INVC = 0
      NUMVR   =   NUMVR  +1
      TRIG = .FALSE.
      DO 1101  I = 1,M2
        E(I) = 0.0
 1101 CONTINUE
      MM=1
      DO 1113  I = 1,M
        E(MM) = 1.0
        PE(I)  =  0.0
        X(I) =  B(I)
        IF (JH(I) .NE.0) JH(I) = -1
        MM = MM + M + 1
 1113 CONTINUE
C                    FORM INVERSE
      JT=1
 1103   IF (KB(JT).EQ.0)  GO TO 1102
        GO TO 600
C 600   CALL JMY
C                           CHOOSE PIVOT
 1114   TY = 0.0
        KQ = .FALSE.
        DO 1104 I = 1,M
      IF(JH(I).NE.-1.OR.ABS(Y(I)).LE.TPIV) GO TO 1104
          IF (KQ) GO TO 1116
          IF (X(I).EQ.0.)  GO TO 1115
          IF (ABS(Y(I)/X(I)).LE.TY)  GO TO 1104
          TY = ABS(Y(I)/X(I))
          GO TO 1118
 1115     KQ = .TRUE.
          GO TO 1117
 1116     IF (X(I).NE.0..OR.ABS(Y(I)).LE.TY)  GO TO 1104
 1117     TY = ABS(Y(I))
 1118     IR = I
 1104   CONTINUE
      KB(JT)=0
C                          TEST PIVOT
        IF (TY.LE.0.)    GO TO 1102
C                          PIVOT
        GO TO 900
C 900   CALL PIV
 1102 JT=JT+1
      IF(JT.NE.(N+1))GOTO1103
C                   RESET ARTIFICIALS
      DO 1109  I = 1,M
        IF (JH(I).EQ.-1)  JH(I) = 0
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1109 CONTINUE
 1200 VER = .FALSE.
C              ***       PERFORM ONE ITERATION    ***
C* @XCK@    DETERMINE FEASIBILITY                             (STEP 1)
      NEG = .FALSE.
      IF (FEAS)  GO TO 500
      FEAS= .TRUE.
      DO 1201  I = 1,M
        IF (X(I).LT.0.0)  GO TO 1250
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1201 CONTINUE
C* @GET@     GET APPLICABLE PRICES                            (STEP 2)
      IF (.NOT.FEAS)  GO TO 501
  500 DO 503 I = 1,M
        P(I) = PE(I)
        IF (X(I).LT.0.)  X(I) = 0.
  503 CONTINUE
      ABSC = .FALSE.
      GO TO 599
 1250 FEAS = .FALSE.
      NEG  = .TRUE.
  501 DO 504  J = 1, M
        P(J) = 0.
  504 CONTINUE
      ABSC = .TRUE.
      DO 505  I = 1,M
        MM = I
        IF (X(I).GE.0.0)  GO TO 507
        ABSC = .FALSE.
        DO 508 J = 1,M
          P(J) = P(J) + E(MM)
          MM = MM + M
  508   CONTINUE
        GO TO 505
  507   IF (JH(I).NE.0) GO TO 505
        IF (X(I).NE.0.) ABSC = .FALSE.
        DO 510  J = 1,M
          P(J) = P(J) - E(MM)
          MM = MM + M
  510   CONTINUE
  505 CONTINUE
C* @MIN@     FIND MINIMUM REDUCED COST                        (STEP 3)
  599 JT = 0
      BB  =  0.0
      DO 701  J =1,N
        IF (KB(J).NE.0)   GO TO 701
        DT = 0.0
        DO 303 I = 1,M
          DT = DT + P(I) * A(I,J)
  303   CONTINUE
```

```
        IF (FEAS)  DT = DT + C(J)
        IF  (ABSC)  DT = - ABS(DT)
        IF (DT.GE.BB)  GO TO 701
        BB = DT
        JT = J
  701 CONTINUE
C  TEST FOR NO PIVOT COLUMN
      IF (JT.LE.0)  GO TO 203
C  TEST FOR ITERATION LIMIT EXCEEDED
      IF (ITER.GE.NCUT)  GO TO 160
      ITER = ITER +1
C* @JMY@      MULTIPLY INVERSE TIMES A(.,JT)             (STEP 4)
  600 DO 610  I= 1,M
        Y(I) = 0.0
  610 CONTINUE
      LL = 0
      COST = C(JT)
      DO 605   I= 1,M
        AIJT = A(I,JT)
        IF (AIJT.EQ.0.) GO TO 602
        COST = COST + AIJT * PE(I)
        DO  606  J = 1,M
          LL = LL + 1
          Y(J) = Y(J) + AIJT * E(LL)
  606   CONTINUE
        GO TO 605
  602   LL = LL + M
  605 CONTINUE
C         COMPUTE PIVOT TOLERANCE
      YMAX = 0.0
      DO 620  I  =  1,M
        YMAX = AMAX1( ABS(Y(I)),YMAX )
  620 CONTINUE
      TPIV  =  YMAX * TEXP
C             RETURN TO INVERSION ROUTINE, IF INVERTING
      IF (VER)  GO TO 1114
C       COST TOLERANCE CONTROL
      RCOST = YMAX/BB
      IF (TRIG.AND.BB.GE.-TPIV)  GO TO 203
      TRIG = .FALSE.
      IF (BB.GE.-TPIV)  TRIG = .TRUE.
C* @ROW@     SELECT PIVOT ROW                                (STEP 5)
C AMONG EQS. WITH X=0, FIND MAX Y AMONG ARTIFICIALS,
C  OR,IF NONE, GET MAX POSITIVE Y(I) AMONG REALS.
      IR = 0
      AA = 0.0
      KQ = .FALSE.
      DO  1050  I =1,M
        IF (X(I).NE.0.0.OR.Y(I).LE.TPIV)  GO TO 1050
        IF (JH(I).EQ.0)  GO TO 1044
        IF (KQ) GO TO 1050
```

```
 1045    IF (Y(I).LE.AA)  GO TO 1050
         GO TO 1047
 1044    IF (KQ) GO TO 1045
         KQ = .TRUE.
 1047    AA = Y(I)
         IR = I
 1050 CONTINUE
      IF (IR.NE.0)  GO TO 1099
      AA = 1.0E+20
C                 FIND MIN. PIVOT AMONG POSITIVE EQUATIONS
      DO 1010  I  = 1,M
      IF(Y(I).GT.1.0E+50)Y(I)=1.0E+50
      IF(Y(I).LE.TPIV.OR.X(I).LE.0.0.OR.Y(I)*AA.LE.X(I))
     1        GO TO 1010
        AA = X(I)/Y(I)
        IR = I
 1010 CONTINUE
      IF (.NOT.NEG)  GO TO 1099
C FIND PIVOT AMONG NEGATIVE EQUATIONS, IN WHICH X/Y IS
C  LESS THAN THE MINIMUM X/Y IN THE POSITIVE EQUATIONS
C  THAT HAS THE LARGEST ABSF(Y)
      BB = - TPIV
      DO 1030  I = 1,M
      IF(X(I).GE.0..OR.Y(I).GE.BB.OR.Y(I)*AA.GT.X(I))
     1        GO TO 1030
        BB = Y(I)
        IR = I
 1030 CONTINUE
C  TEST FOR NO PIVOT ROW
 1099 IF  (IR.LE.0)  GO TO 207
C* @PIV@     PIVOT ON (IR,JT)                              (STEP 6)
      IA = JH(IR)
      IF (IA.GT.0)  KB(IA) = 0
  900 NUMPV  =  NUMPV  +  1
      JH(IR) = JT
      KB(JT) = IR
      YI = -Y(IR)
      Y(IR) = -1.0
      LL = 0
C                                     TRANSFORM INVERSE
      DO  904  J = 1,M
        L = LL + IR
        IF (E(L).NE.0.0)  GO TO 905
        LL = LL + M
        GO TO 904
  905   XY = E(L) / YI
        PE(J) = PE(J) + COST * XY
        E(L) = 0.0
        DO 906  I = 1,M
          LL  =  LL + 1
          E(LL) = E(LL)  + XY * Y(I)
  906   CONTINUE
  904 CONTINUE
C                                     TRANSFORM X
      XY  =  X(IR) / YI
      DO 908  I  = 1, M
        XOLD = X(I)
        X(I) = XOLD + XY * Y(I)
      IF(.NOT.VER.AND.X(I).LT.0..AND.XOLD.GE.0.)X(I)=0.
  908 CONTINUE
      Y(IR) = -YI
      X(IR) = -XY
      IF (VER) GO TO 1102
      IF (NUMPV.LE.M) GO TO 1200
C  TEST FOR INVERSION ON THIS ITERATION
      INVC    = INVC    +1
      IF  (INVC.EQ.NVER)   GO TO 1320
      GO  TO  1200
C*  END OF ALGORITHM, SET EXIT VALUES                      ***
  207 IF (.NOT.FEAS.OR.RCOST.LE.-1000.)  GO TO 203
C               INFINITE SOLUTION
      K   =  2
      GO TO 250
C             PROBLEM IS CYCLING
  160 K   =   4
      GO TO 250
C             FEASIBLE OR INFEASIBLE SOLUTION
  203 K   =   0
  250 IF (.NOT.FEAS)  K = K + 1
      DO 1399  J = 1,N
        XX   =  0.0
        KBJ   =  KB(J)
        IF (KBJ.NE.0)  XX = X(KBJ)
        KB(J) = LL
 1399 CONTINUE
      KO(1) = K
      KO(2) = ITER
      KO(3) = INVC
      KO(4) = NUMVR
      KO(5) = NUMPV
      KO(6) = JT
 9    FORMAT(1H ,'THIS IS JT AT PREVIOUS INTERRUPT',
     1         I10,'ERROR IF LARGER THAN 10')
 2000 FORMAT(' ',F10.6,I10)
59    FORMAT(1H ,I10,F10.0)
      RETURN
       END
```

BIBLIOGRAPHY

1. Agin, N., "Optimum Seeking with Branch-and-Bound", Management Science, Vol. 13 (1966), pp. B176-B185.

2. Balas, E., "An Additive Algorithm for Solving Linear Programs with (0,1) Variables", Operations Research, Vol. 13 (1965), pp. 517-546.

3. Balinski, M. L., "Integer Programming: Methods, Uses, Computations" Management Science, Vol. 12 (1965), pp. 253-313.

4. Balinski, M. L., and Spielberg, K., "Methods for Integer Programming: Algebraic, Combinatorial, and Enumerative", Progress in Operations Research, Vol. III, Julius S. Aronofsky (ed.), John Wiley & Sons, Inc., New York, 1969, Chap. 7.

5. Beale, E. M. L., "Survey of Integer Programming", Operational Research Quarterly, Vol. 16 (1965), pp. 219-228.

6. Byrne, E. R., "Optimal Scheduling of Indivisible Jobs on a Sequence of Machines without Job Passing", PhD Dissertation, Polytechnic Institute of Brooklyn, 1967.

7. Conway, R. W., Maxwell, W. L., and Miller, L. W., Theory of Scheduling, Addison-Wesley, Reading, Mass., 1967.

8. Dakin, R. J., "A tree-search Algorithm for Mixed Integer Programming Problems", The Computer Journal, Vol. 8 (1965), pp. 250-255.

9. Dorg, A. G., and Land, A. H., "An Automatic Method of Solving Discrete Programming Problems", Econometrics, July, 1960, pp. 497-521.

10. Driebeck, N. J., "An Algorithm for the Solution of Mixed Integer Programming Problems", Management Science, Vol. 12 (1966), pp. 576-587.

11. Geoffrion, A. M., "Integer Programming by Implicit Enumeration and Balas' Method", SIAM Review, Vol. 9 (1967), pp. 178-191.

12. Glover, F., "A Multiphase-Dual Algorithm for the (0,1) Integer Program Problem", Operations Research, Vol. 13 (1965), pp. 879-919.

13. Gomory, R. E., "All-Integer Integer Programming Algorithm", Industrial Scheduling, J. F. Muth, and G. L. Thompson (editors), Prentice-Hall, New York, 1963, pp. 193-206.

BIBLIOGRAPHY (cont.)

14. Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs", Recent Advances in Mathematical Programming, R. L. Graves, and P. Wolfe (editors), McGraw-Hill, New York, 1963, pp. 269-302.

15. Greenberg, H. H., "A Branch-and-Bound Solution to the General Scheduling Problem", Operations Research, Vol. 16 (1968), pp. 353-361.

16. Gue, R. L., Liggett, J. C., and Cain, K. C., "Analysis of Algorithms for the Zero-One Programming Problem", Communications of the ACM, Vol. 11 (1968), pp. 837-844.

17. Healy, W. C. Jr., "Multiple Choice Programming", Operations Research, Vol. 12 (1965), pp. 122-138.

18. Lawler, E. L., and Bell, M. D., "A Method for Solving Discrete Optimization Problems", Operations Research, Vol. 14 (1966), pp. 1098-1112.

19. Lawler, E. L., and Wood, D. E., "Branch-and-Bound Methods: A Survey", Operations Research, Vol. 14 (1966), pp. 699-719.

20. Lemke, C. E., and Spielberg, K., "Direct Search Algorithms for Zero-One and Mixed-Integer Programming", Operations Research, Vol. 15 (1967), pp. 892-914.

21. Little, J. D. C., Murty, Siweeney, and Karel, "An Algorithm for the Traveling Salesman Problem", Operations Research, Vol. 11 (1963), pp. 972-989.

22. Pritsker, A. A. B., and Watters, L. J., "A Zero-One Programming Approach to Scheduling with Limited Resources", Rand Corporation, Memorandum RM5561-PR, January, 1968.

23. Schultz, G. M., Private Communication (with permission).

24. Young, R. D., "A Primal (All-Integer) Integer Programming Algorithm", J. Res. National Bureau of Standards, Vol. 69B (1965), pp. 213-250.

25. Young, R. D., "A Simplified Primal (All-Integer) Integer Programming Algorithm", Operations Research, Vol. 16 (1968), pp. 750-782.

VITA

PERSONAL HISTORY

Name:	Merrick H. Cooperman
Date of Birth:	March 12, 1945
Place of Birth:	Chicago, Illinois
Parent:	Bernice Rapp
Wife:	Susan B.

EDUCATIONAL BACKGROUND

S. T. Mather High School Chicago, Illinois	Graduated-1962
University of Chicago Chicago, Illinois	1962-1963 1967-1968
Northwestern University Evanston, Illinois Bachelor of Science in Electrical Engineering	1963-1966
Roosevelt University Chicago, Illinois	1967
Lehigh University Candidate for Master of Science in Industrial Engineering	1968-1970

HONORS

Alpha Pi Mu, Industrial Engineering Honor Society

PROFESSIONAL EXPERIENCE

Western Electric Co., Inc. Chicago, Illinois Development Engineer	1966-1968
Western Electric Co., Inc. Princeton, New Jersey Research Engineer, Information Systems Staff Member-Lehigh Master's Program	1968-1970