A ONE-COMPLETION ENUMERATIVE METHOD FOR ZERO-ONE INTEGER PROGRAMMING

By

DENNIS DON BROWN

Bachelor of Science

Texas A&M University

College Station, Texas

1980

Submitted to the Graduate Faculty of the
Department of Management
College of Business Administration
Oklahoma State University
in partial fulfillment of the
requirements for the Degree of
MASTER OF BUSINESS ADMINISTRATION
December, 1985

Name: Dennis Don Brown Date of Degree: December 1985

Institution: Oklahoma State University

Location: Stillwater, Oklahoma

Title of Study: A ONE-COMPLETION ENUMERATIVE METHOD FOR ZERO-ONE INTEGER PROGRAMMING

Pages in Study: 66 Candidate for Degree of Master of Business Administration

Major Field: Business Administration

Scope and Method of Study: This study examines the effectiveness of a one-completion enumerative algorithm for solution of zero-one integer linear programming problems. The algorithm utilizes a search tree data structure to select partial solution vectors for active processing. A one-completion test is incorporated in the algorithm to determine the need for explicit enumeration of search tree branches. Five zero-one integer problems are solved via the one-completion method. These same five problems are also used to test the effectiveness of reordering problem variables with respect to objective function coefficient magnitude before beginning the one-completion procedure.

Findings and Conclusions: The one-completion algorithm used in this study was shown to be as effective as the basic Balas additive algorithm for solution of small zero-one problems. For the five problems tested, three were solved faster with the one-completion method including problem reordering. For these same five problems, reordering reduced one-completion processing time by an average of 41%.

ADVISOR'S APPROVAL Mitchell O. Locks

A ONE-COMPLETION ENUMERATIVE

METHOD FOR ZERO-ONE INTEGER

PROGRAMMING

Report Approved:

Mitchell O. Locks

Advisor

Director of Graduate Studies

Head, Department of Management

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

a_{ij} - coefficient of variable X_j in constraint equation g_i.

b_i - numerical constant in constraint equation g_i.

C_j - objective function coefficient of variable X_j.

CP - candidate problem.

F(P)- set of feasible solutions for problem P.

g_i - constraint equation.

g_o - objective function.

g_o^* - best feasible solution value located thus far.

h_k - lower bound for variable X_k.

m - the number of constraints.

n - the number of variables.

P - linear programming problem.

P_R - relaxation of LP problem P.

S - partial solution vector

T - set of variables with $C_j < (\bar{z}-g_o^*)$ and a positive coefficient in some constraint in V.

u_k - upper bound for variable X_k.

V - set of violated constraints when S is zero-completed.

X_i - LP problem variable.

$\underline{X}_i$ - partial solution vector.

$\underline{X}^*$ - interim optimum solution vector.

y' - vector used for development of surrogate constraint.

INTRODUCTION

The one-completion algorithm is an enumerative procedure for solving zero-one integer programming problems. In this paper, a very early form of the algorithm developed by Locks, Sharda, and LeClaire (14) is shown to be as effective as the basic Balas additive algorithm for solving small zero-one programming problems. For five problems tested, three were solved faster with the one-completion method. Suggestions for possible improvement of the algorithm are presented in the conclusion of this paper. Locks, Sharda, and LeClaire report that a newer version of the one-completion algorithm written in PL1 has proven to be much faster than the Balas algorithm (14).

The one-completion algorithm utilizes a search tree data structure to select partial solution vectors for active processing. As with other enumerative methods, the fathoming criteria used are based primarily on the logical implications of the problem constraints. One such criterion used in this algorithm is the one-completion test. By one-completing partial solution vectors and computing the corresponding solution value, a quick determination is made of the possibility for achieving an improved solution by continued processing of a given tree branch.

The report begins with an overview of the methods currently being studied and used for solution of integer programming problems. Particular attention is given to the Balas additive implicit enumeration procedure in order to provide a basis for examination of the one-completion algorithm.

A detailed explanation of the one-completion algorithm appears in the following chapter. An example problem is also solved via one-completion to provide a better understanding of the mechanics of the algorithm.

Finally, five zero-one integer problems are solved via one-completion and the basic Balas additive algorithm in order to gauge the computational efficiency of the one-completion method. The results of this test are presented in Table I of this report.

CHAPTER 1

LITERATURE SURVEY

INTEGER AND ZERO-ONE LINEAR PROGRAMMING

Integer linear programming (ILP) problems are formed from linear programming problems by constraining some or all controllable variables to have integer values. Those problems with a combination of integer and continuous variables are referred to as mixed integer linear programming (MILP) problems while those problems with no continuous variables are referred to as all-integer linear programming (AILP) problems. Limiting ILP solution values to discrete alternatives rather than a continuum makes these problems much more difficult to solve than ordinary LP problems.

AILP problems are referred to as zero-one programming problems when all controllable variables are required to be less than or equal to 1, $x \leq 1$. Thus, after accounting for nonnegativity requirements, all variables are limited to values of either 0 or 1. Of course, all-integer and zero-one problems can be classified as special cases of each other. To represent a zero-one variable as a general integer variable, all that is required is the addition of an upper bound constraint, $x_j \leq 1$. To represent a general integer variable as a zero-one variable, a sum of zero-one variables can be used. Another, more economical, method of representing a general integer variable as a zero-one variable is to use a sum of 0-1 variables whose coefficients are powers of 2.

A few examples of problems that lend themselves to solution via ILP include: equipment utilization, problems where setup costs are incurred if a project is selected, production planning problems with minimum batch sizes for selected products, and problems with go-no-go decisions. Zero-one programming is used to solve this last type of problem where the Ø or 1 values of variables represent yes-no, go-no-go, or either-or decisions.

Dantzig has shown that any deterministic problem which can be precisely described in quantitative terms can be approximately formulated as accurately as desired as a mixed integer programming problem. Integer variables allow representation of constraint sets which are nonconvex (3).

General Framework of ILP and Zero-One Programming

In an effort to develop a general algorithmic framework for integer programming, Geoffrion and Marsten (1) have identified three key features common to most known ILP computational approaches. These features are separation, relaxation, and fathoming criteria.

Separation can be considered a divide and conquer approach to ILP problems. The rudimentary separation strategy presented by Geoffrion and Marsten involves: (1) making a reasonable effort to solve the problem, (2) if unsuccessful, separate the problem into two or more problems and add these to a candidate list, (3) extract a candidate problem from the list and attempt to solve it, (4) if solved, extract another candidate problem, if not solved, separate the candidate problem and add these to the candidate list, and (5) continue until the candidate list is exhausted.

The usefulness of the separation approach depends upon its success in solving candidate problems without further separation. Two of the more common separation techniques are addition of contradictory constraints on a single integer variable and separation on multiple choice constraints.

Relaxation of an optimization problem involves "loosening" constraints and forming a new relaxed problem. The only requirement for relaxed problem (P_R) to be a valid relaxation for original problem (P) is that $F(P_R) \subseteq F(P)$ where F(P) and $F(P_R)$ are the sets of feasible solutions for the original problem and relaxed problem respectively. This yields the following relationships for a minimization problem: (1) If (P_R) has no feasible solutions, the same is true for (P), (2) the minimum value of F(P) is no less than the minimum value of $F(P_R)$, and (3) if an optimal solution of (P_R) is feasible in (P), then it is an optimal solution of (P).

The primary criteria for selection of the type of relaxation are: (1) the relaxed problem should be easier to solve than the original and (2) the relaxed problem should yield an optimal solution as close to the original problem solution as possible. Omitting constraints, dropping integrality requirements, and dropping nonnegativity conditions are three of the most common relaxation techniques.

Fathoming criteria, as described by Geoffrion and Marsten, are introduced to clarify the role of relaxation in solving a sequence of candidate problems. Fathoming criteria are used to determine if continued processing of a candidate problem is worthwhile. A candidate problem has been fathomed if any one of the following criteria is satisfied. (1) An analysis of the relaxed candidate problem (CP_R) reveals that the candidate problem (CP) has no feasible solution. (2) An analysis of (CP_R) reveals that (CP) has no feasible solution better than the incumbent. And (3) an analysis of (CP_R) reveals an optimal solution of (CP) (i.e., an optimal solution of (CP_R) which is feasible in (CP). There is considerable variation among ILP algorithms as to the type and combination of analyses used.

An Overview of Some Current ILP and Zero-One Algorithms

There are many different methods in existence for solving ILP and zero-one problems. A major portion of these approaches can be catagorized as cutting plane algorithms, group theoretic algorithms, decomposition algorithms, or tree search type algorithms. The cutting plane, group theoretic and decomposition methods, along with the tree search methods branch and bound plus direct search, will be discussed very briefly below. The additive tree search method proposed by Balas will be discussed in greater detail in the following section.

CUTTING PLANE ALGORITHMS

In the cutting plane method, linear cut constraints are added to the original problem in order to construct a new problem which has an optimal integer corner solution. Each cut removes part of the feasible region without removing any of the feasible integer solutions. In terms of the general framework discussed earlier, the approach is based on successively improved relaxations of the original problem with no use of the separation technique. Most methods begin by relaxing all integrality requirements and solving the LP problem. The relaxation is then tightened by the addition of cutting plane constraints.

Most cut methods either begin with a dual feasible (dual methods) or a primal feasible (primal methods) starting solution. Cut constraints are generated and utilized until a feasible solution is located. One of the major disadvantages of the cut method is that a feasible solution is not located until the final iteration, when the problem is solved. For some methods, it may not be possible to obtain a feasible solution with a finite number of cut constraints. While some methods have been proven to converge if an optimum solution exists (3), solution of the problem may not be economical due to the number of cut iterations involved.

Examples of current cutting plane algorithms include Gomory's fractional, all-integer, and mixed integer algorithms, the Dantzig method, Balas' intersect cut, and primal algorithms developed by Young and Glover (3)(5). Some success has been reported by Gorry and Shapiro in combining cutting plane techniques with enumerative algorithms (1)(8).

GROUP THEORETIC APPROACHES

The group theoretic approach, which has been applied almost exclusively to pure integer programming problems, begins by transforming the problem to an equivalent form using a dual feasible basis. Zionts (3) refers to it as an all-integer, primal dual feasible starting solution, constructive method. In the method proposed by Gorry and Shapiro (8), the candidated problem is relaxed to a group problem by dropping the nonnegativity conditions on basic variables. As a separation technique, the group problem solution is used to compute lower bounds on the minimal values of the new candidate problems. The candidate with the lowest bound is then selected for fathoming (1)(3)(8).

BENDER'S DECOMPOSITION

Bender's decomposition is a method for solving mixed integer linear programming problems. The basic idea behind this approach is to alternate between (1) taking trial values for the discrete variables and finding the optimum values for the continuous variables and (2) taking the resulting continuous variable optimum and seeking improved values for the integer variables (1)(2).

BRANCH AND BOUND

The branch and bound method has been classified by Hu (5) as a tree search type algorithm. These algorithms are easier to understand and program than the methods discussed previously. According to Anderson, Sweeney and Williams (7), the branch and bound method is currently the most efficient general purpose procedure for ILPs and MILPs and is used in almost all commercially available ILP programs.

The general branch and bound procedure described by Land and Doig (6) has the following basic steps. (1) Relax all integrality constraints and solve the problem via simplex or some other LP method. This problem assumes the title of problem B. (2) If the solution to problem B is all integer, the problem is solved. If not, proceed to the next step.

(3) A variable, Xa, with a fractional value, y, is selected from the solution of B and used for separation. Two new problems are formed from B and solved by relaxing the integrality constraints. One of the new problems has the added constraint Xa$\geq$ the smallest integer greater than y and the other problem has the added constraint Xa$\leq$ the largest integer less than y. These problems are then added to the candidate list. (4) The problem from the candidate list with the best solution value is selected to become problem B and the procedure moves back to step number 2.

It appears that the primary difference among branch and bound procedures is the heuristic used to select the separation variable. For example, some of the methods currently in use include (a) arbitrary selection, (b) selecting the variable which is furthest from integral, and (c) selecting the variable based on penalties derived from studying the simplex tableau, studying the first dual simplex iteration, or some other method (1)(2)(3)(5)(6)(7).

DIRECT SEARCH

The direct search method proposed by Lemke and Spielberg (9) for solution of zero-one ILP problems is very similar to the Balas additive algorithm to be discussed in the next section. Both involve implicit enumeration. The first step of the Lemke-Spielberg approach is to restate the problem with all less than or equal to constraints. Following this, the constraints are transformed to equalities with slack variables added. The slack variables can assume only nonnegative integer values.

Three tests are then performed to reduce explicit enumeration of partial solutions. First, the "projected exclusion test" is performed by adding a constraint derived from the function g_0 which is to be minimized. Next, an "infeasibility test" is performed on each constraint to determine if it can possibly be made feasible by adding free variables (variables with no assigned value) to the partial solution. If not, a backtracking procedure is performed. Finally, "preferred variable tests" are performed to select the next variable to be added to the partial solution. The heuristic recommended by Lemke-Spielberg is to select the variable which most greatly reduces negative deviation of the slack variables (4)(5)(9).

Balas' Additive Algorithm

Methods such as the Balas additive algorithm are often referred to as implicit enumeration procedures. These methods, by themselves, are used almost exclusively for all-integer programming problems. Most applications have been for zero-one type integer problems. The discussion which follows will concentrate solely on zero-one applications.

Implicit enumeration procedures methodically search the set of all possible solutions in such a way that all possibilities, or combinations, are considered either explicitly or implicitly. Of course, the objective is to arrive at the optimal feasible solution with as little explicit enumeration as possible. The fathoming criteria used are based primarily on logical implications of the problem constraints.

Hu (5) presents four common features of implicit enumeration algorithms. (1) They are easy to understand. (2) They are easy to program. (3) The upper bound on the number of solution steps is known. And (4) they lack the mathematical structure of the cutting plane or group theoretic type approaches. The first two features are clearly advantages of the implicit enumeration procedures. The major disadvantage of the implicit enumeration approaches is indicated in feature number three. For zero-one problems, the number of possible solutions, or Ø-1 combinations, is 2^n where n is the number of variables. This implies that computing times, on average, will increase exponentially with the number of variables. Hu reports that empirical results support this idea. In general, the implicit enumeration procedures require less computing time than cutting plane algorithms for small problems but their growth in computing time is more rapid as the number of variables increases (5)(1).

GENERAL PROCEDURE FOR IMPLICIT ENUMERATION

A block flow diagram of the Balas additive algorithm, as presented by Plane and McMillan (6), is presented in figure I. To use the procedure as stated, zero-one integer programming problems must be expressed in the form:

$$\text{Min } g_o = \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to } g_i = \sum_{j=1}^{n} a_{ij} x_j - b_i \leq 0 \qquad i=1,\ldots,m$$

where $m \equiv$ the number of constraints
$n \equiv$ the number of variables
$c_j, a_{ij}, b_i \equiv$ numerical coefficients

As the procedure begins, none of the variables have been assigned a value of Ø or 1. Therefore, the partial solution, S, contains no variables. The zero completions of S described in steps 2 and 4 will require that all the constraints (step #2) and g_o (step #4) be calculated with all variables temporarily assigned values of zero.

The procedure uses two basic fathoming criteria for partial solutions. First, the partial solution has been fathomed if it is established that no completion is capable of yielding an improved solution. Completing the partial solution simply involves adding Ø or 1 valued variables to S. Steps 4, 5, 6 and 11 are used to determine if an improved solution is possible. In step 4, all variables not in S are temporarily assigned a value of Ø and g_o is computed. This value is then subtracted from the best feasible solution value located thus far (g_o^*). This establishes a limit on the objective function values of variables which will be considered for addition to S. If no free variables with objective function coefficients less than the limit exist, then the set T is empty and step 6 sends the algorithm to a backtracking procedure for selection of a new partial solution.

The partial solution has also been fathomed if it is established that no completion of S can possible yield a feasible solution. This test is accomplished in steps 2, 5, 6, and 7. The set of constraints violated by the zero completed partial solution (set V) is established in step 2. In step 5, those free variables which could possibly improve feasibility and have objective function values within the limit established in step 4 are added to set T. In step 7, it is determined if all constraints in V can be made feasible by adding only variables in T. If this is possible, the variable in T with the largest coefficient sum is added to S. If this is not possible, the partial solution has been fathomed and backtracking begins.

As a subcase of the first fathoming criterion, it should be noted that the partial solution has been fathomed if it is feasible. Clearly, for a minimization problem with all positive objective function coefficients, no improvement is possible by adding one valued variables to a feasible partial solution. Therefore, step 3 sends all feasible partial solutions to backtracking.

As a further note, the heuristics used in steps 7 and 8 are a primary source of variation among implicit enumeration approaches. In step 7, the approach used by Plane and McMillan (6) is to complete each violated constraint by assigning a 1 value to every variable in T which has a positive coefficient in that constraint. Step 8 has already been discussed. Some alternate approaches will be discussed later.

Steps 10 and 11 comprise the backtracking procedure which was mentioned earlier. This procedure facilitates coverage of the entire solution tree without reexamination of partial solutions. Backtracking begins once it has been established that a partial solution has been fathomed. In step 10, the rightmost (most recently added) positive (one valued) variable in S is replaced with its complement (assigned a zero value).

An IBM BASIC translation of the Balas implicit enumeration algorithm presented by Plane and McMillan is provided in appendix A. This program was used to study the comparative efficiency of the one-completion method to be discussed later in this paper.

SURROGATE CONSTRAINTS

Many current variations of the Balas additive algorithm utilize surrogate constraints. The purpose of surrogate constraints is to speed the solution of zero-one problems. It has been shown that a surrogate can be constructed which captures a great deal of the joint logical implications of the entire set of constraints (1)(10)(3). By adding such a joint constraint, many infeasible partial solutions that slip by step 7 of the Balas additive algorithm might be picked up and fathomed implicitly.

As mentioned, it is desirable that the surrogate constraint represent the logical implications of the entire set of constraints as strongly as possible. A surrogate constraint can be represented by $y'Ax \leq y'b$ where $Ax \leq b$ is the constraint set and y' is a vector of appropriate order. Balas has shown that, given two surrogate constraints ($a_0 x \leq b_0$ and $a_1 x \leq b_1$), the stronger constraint yields the larger objective function value in a minimization problem subject only to the surrogate constraint and the nonnegativity constraint.

It has been shown that, for a given linear programming problem (the continuous analog of the 0-1 problem), the optimun dual solution yields multipliers for constructing the strongest surrogate constraint (3).

Zionts (3) presents this general outline for employing surrogate constraints based on separate articles by Balas (11), Geoffrion (10) and Glover (13). (1) The objective function is adjoined as a constraint requiring that any feasible solution have an objective function value better than the current optimum. (2) The corresponding LP is solved and the surrogate is added. A generalized procedure, such as the Balas additive algorithm, is then used. However, just prior to choosing a variable for addition to the solution vector, a new surrogate constraint is added by holding the assigned variables fixed and solving an LP problem. If the primal solution is integral, it is recorded and backtracking begins. If there is no feasible LP solution, then there is no feasible completion and backtracking begins. Some specified number of constraints are retained. (3) While backtracking, any surrogate constraints conditional upon partial solutions being deleted are dropped.

Geoffrion reports that for 30 problems tested, 29 required less time for solution when the addition of surrogate constraints was included in the solution procedure. The basic method used was Balas' additive algorithm. One of the 30 problems was not solved by either method. (3)

AGGREGATING CONSTRAINTS

It has been shown that is it possible to construct a single aggregate constraint which has the same integer solution set as the original constraints (3)(6). The potential benefit of combining all constraints into a single constraint is obvious. Most approaches involve combining two constraints, combining this with a third, and so on. The primary disadvantage of this approach is that the aggregate constraint variables quickly become too large to be stored as integer in a single computer word.

ZIONTS GENERALIZED ADDITIVE ALGORITHM UTILIZING VARIABLE BOUNDS

One other implicit enumeration algorithm will be discussed briefly. This is the generalized additive algorithm developed by Zionts (3). Zionts claims to have developed an algorithm which is simpler and more powerful than the basic Balas additive algorithm by generating upper and lower bounds on variables, and by using a simplified Balas structure of implicit enumeration.

The primary difference between the generalized method and the Balas algorithm is the generation of upper and lower bounds for each zero-one variable in every constraint. If $\emptyset < h_k \leq 1$, where h_k is the lower bound for variable X_k, it is implied that $X_k=1$ in all completions of the current partial solution. If $h_k > 1$, there is no feasible continuation and backtracking occurs. If $\emptyset \leq u_k < 1$, where u_k is the upper bound for variable X_k, it is implied that $X_k=0$ in all continuations of the current partial solution. If $u_k < 0$, no feasible continuation exists and backtracking occurs. If, for all variables, $u_k \geq 1$ and $h_k \leq 0$, no tighter bounds are available.

CONCLUSION

This completes the literature survey of current integer linear programming procedures. While this survey was by no means exhaustive, it was intended to provide enough information to effectively analyze and understand the one-completion method. The one-completion method will be compared directly with the basic Balas additive algorithm discussed in this chapter.

CHAPTER 2

THE ONE-COMPLETION ALGORITHM

The one-completion algorithm differs from the basic Balas additive algorithm in four principal ways:

1. A search tree data structure is used to select partial solution vectors (nodes) for active processing.
2. A one-completion test is incorporated in the algorithm to determine if continued processing of tree branches might yield an improved solution.
3. The zero-completion test for feasibility is differentiated from the zero-completion test of the objective function for a potential improved solution.
4. The sequence of node processing decisions has been changed.

A search tree for a five variable problem is given in Figure II. Kaufmann and Labordere refer to this structure as an arborescence (4). The search tree is an acyclic structure with all nodes except the root (top of the tree) and leaves (bottom of the tree) having indegree one and outdegree two. The root has indegree zero and the leaves have outdegree zero. Each node of the tree represents a partial solution vector $\underline{X}i=(X_1,\ldots,X_j,\_)$ with either $\emptyset$ or 1 specified for variables X_1 through X_j and nothing specified for X_{j+1} through X_n. The root, $X_o=(\_)$, has no specified variables while the leaves, $\underline{X}_i=(X_1,\ldots,X_n)$, have all variables specified.

Each node, except for the leaves, is the father of two sons (outdegree two). The elder son is the father augmented by $X_{j+1}=1$. The younger son is the father augmented by $X_{j+1}=0$.

In order to use the one-completion algorithm as presented in this report, a model must be stated in the following form:

$$\text{Max } g_o=\sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } g_i=\sum_{j=1}^{n} a_{ij}X_j-b_i\leq 0 \quad i=1,\ldots,m$$

$$C_j\geq 0,\ X_j=0,1 \ ,\quad j=1,\ldots,n$$

where $m \equiv$ the number of constraints
$n \equiv$ the number of variables
$C_j, a_{ij}, b_i \equiv$ numerical coefficients

Since the model is formulated such that the objective function is to be maximized, an improvement in the objective function can only be found by augmenting $\underline{X}_i$ with one-valued variables. Therefore, only those nodes with X_i=1 are processed. All other nodes are implicitly enumerated. This is reflected in the search tree presented in Figure III.

A block flow diagram of the one-completion algorithm is given in Figure IV. Decision points are represented by diamond shaped boxes, operations are represented by rectangles, and circles are used for labeling. The algorithm begins at label A with the root, $\underline{X}_o=(\underline{.})$, being the first node selected for processing.

At label A, a zero-completion of the current node is used to check for feasibility. The zero-completion of a node, $(\underline{X},0)$, is simply the partial solution vector $\underline{X}_i=(X_1,\ldots,X_j,\underline{.})$ augmented by a subvector of zeros for all free variables X_{j+1} through X_n. Feasibility is achieved when the value of each constraint equation is less than or equal to zero.

In Figure IV, the feasibility test is stated in the form of the question; is $g_j(\underline{X0}) \leq 0$, j=1,...,m? If all constraints are satisfied, a feasible solution has been found. A zero completion test of the objective function is then performed to determine if a new interim optimum solution has been located. In Figure IV, the zero-completion test of the objective function is represented by the question; is $g_o(\underline{X0}) > g_o^*$? If $g_o(\underline{X0}) > g^*$, or if this is the first feasible solution located, the interim optimum solution becomes $\underline{X}^*=(\underline{X0})$ and the interim optimum objective function value becomes $g_o^*=g_o(\underline{X0})$.

If the current node, $\underline{X}_i$, is not a leaf $(\underline{X}_i \neq \underline{X}1)$ then forward search is used to select the next node to be processed, X_{i+1}. Forward search begins at label F. Forward search proceeds down a tree branch from father to elder son with a one value being assigned to the next free variable, X_{j+1}, in lexicographical order. Therefore, if the current node is $\underline{X}_i$=(01101...), then $\underline{X}_{i+1}$=(011011..)....

If the current node, $\underline{X}_i$, is a leaf, then it is necessary to move to a different tree branch. This is called backtracking. The first step in backtracking involves reversing direction and moving up the tree to an ancestor. This is accomplished by freeing all variables in reverse lexicographical order until the second one valued variable is reached and freed. Therefore, if the current node is $\underline{X}$=(1101.), then the first step of backtracking will take us to the ancestor $\underline{X}'$=(1....)(refer to Figure I).

Once the ancestor has been reached, it is necessary to proceed down a different free branch. Since every ancestor has only two outgoing branches and the branch containing the ancestor's eldest son has already been processed, the next branch processed will be that containing the ancestor's youngest son. To reach the youngest son, a zero-value is assigned to the first free variable of the ancestor node (10...). However, this node was implicitly enumerated when the ancestor was processed earlier. Therefore, we must proceed down the branch one step further by assigning a one value to the next free variable (101..).

Once a feasible interim solution has been located, the one-completion test is performed each time backtracking is used to move to a different search tree branch. In Figure II, the one-completion test is represented by the question; "is $g_o(\underline{X1}) > g_o^*$?". Since the intent is to maximize a model objective function which has no negative coefficients, the one-completion test provides a quick determination of whether continued processing of the new search tree branch could possibly yield an improved solution.

The one-completion of a node is the partial solution vector $\underline{X}_i=(X_{\ },\ldots,X_{i},\_)$ augmented by a subvector of ones for all free variables X_{i+1} through X_n. For example, the one-completion of the eight variable search tree node $\underline{X}$=01101... is $(\underline{X1})$=01101111. It is obvious that there is no need for further processing of the current search tree branch if an improved solution cannot be obtained by assigning one values to all free variables.

If the new node, $\underline{X}_{i+1}$, passes the one-completion test, the algorithm proceeds to label A where the feasibility of the node is determined. If $\underline{X}_{i+1}$ fails the one-completion test, it is necessary to move to another search-tree branch. This is accomplished by moving to label E.

The search is completed when all nodes have been either explicitly or implicitly enumerated. One possible stopping point is the left most leaf on the tree. This leaf, $\underline{X}_i=(\underline{0}1)$, has zero values for all variables X_o through X_{n-1} and a one value for X_n. If this node is reached, no additional nodes will be processed. At that point, the current interim optimum solution $\underline{X}^*$ is the optimum problem solution. If no feasible solutions were located, then the problem has no solution. Please note that alternate optimum solutions could exist which may or may not have been explicitly enumerated.

Another possible stopping point is encountered when a node of the form $\underline{X}_i=(\underline{0}1\underline{.})$ fails the one-completion test. Remembering that a partial solution vector may be expressed as $\underline{X}_i=(X_1,\ldots,X_j,\underline{.})$, the node $\underline{X}_i=(\underline{0}1\underline{.})$ has zero values for all variables X_1 through X_{j-1}, an X_j value of one, and no value assigned to variables X_{j+1} through X_n. If a node of this form fails the one-completion test, the search is ended because further backtracking is not possible. Since a feasible solution had to exist in order for the one-completion test to be performed, the optimum problem solution is $\underline{X}^*$.

EXAMPLE PROBLEM USING THE ONE-COMPLETION ALGORITHM

The following example problem is presented to provide a clearer understanding of how the one-completion algorithm works.

$$\text{Max.} \quad g_0=2X_1 + 6X_2 + 2X_3 + 4X_4 + 3X_5 + 6X_6$$

$$\text{S.T.} \quad g_1 = X_1 - 2X_2 - 3X_3 - 6X_4 + X_5 + 2X_6 + 5 \leq 0$$

$$g_2 = -X_1 + 3X_2 - 2X_3 - 4X_4 - 2X_5 + 4X_6 + 4 \leq 0$$

$$X_j=0,1 \quad , \quad j=1,\ldots,n$$

The sequence of processing steps for this problem is shown in Figure B-5 of Appendix B. Only 30 nodes out of a total of 64 possible zero-one combinations (2^6) are processed before the search is completed. The optimum solution is $\underline{X}^*=111110$ with a solution value of $g_0^*=17$.

The first node processed is the root, $\underline{X}_0=(\underline{.})$, which has no specified variables. A zero completion of this node yields constraint values of $g_1=5$ and $g_2=4$. Since all constraints must be less than or equal to zero in order for the partial solution to be feasible, this node is clearly infeasible. Forward search is used to locate the next node for processing. This simply involves the assignment of a value of one to the first free variable of $\underline{X}_1$. Consequently, the next node chosen for processing is $\underline{X}_2=(1\underline{.})$.

The node $\underline{X}_2=(1\underline{.})$ is processed in the same manner as the previous node. A zero-completion of this node yields constraint values of $g_1=6$ and $g_2=3$. As shown in Figure B-5, forward search continues.

The first feasible node located is $\underline{X}_5=(1111\underline{.})$. This node becomes $\underline{X}^*$ and the interim optimum value of the objective function becomes $g_0^*=14$. Once again, forward search is used to locate the next node for processing. Now that a feasible solution has been located, the one-completion test will be performed each time backtracking is used to move to a new search tree branch.

Forward search continues through node $\underline{X}_7$ with a new interim optimum solution being located at node $\underline{X}_6$=(11111.). Because node $\underline{X}_7$=(111111) is a leaf, it is necessary to backtrack to another tree branch. The first step of backtracking takes us to the ancestor $\underline{X}_7'$=(1111 ..) by freeing variables in reverse lexicographical order until the second one valued variable is reached and freed. Next, a zero value is assigned to the first free variable of $\underline{X}_7'$ yielding $\underline{X}_7''$=(11110.). Finally, a one value is assigned to the first free variable of $\underline{X}_7''$ leaving $\underline{X}_8$=(111101).

Since a feasible solution has already been located, node $\underline{X}_8$ must pass the one-completion test in order to proceed to the feasibility test. $\underline{X}_8$ is a leaf and is, therefore, essentially one-complete. The node yields a one-completion value of $g_0(\underline{X1})=20$ which exceeds the current interim optimum solution of $g_0^*=17$. This indicates that further processing of $\underline{X}_8$ could result in an improved solution and is therefore justified. However, further processing reveals that $\underline{X}_8$ is infeasible. Since $\underline{X}_8$ is a leaf, it is again necessary to backtrack to a different search tree branch.

The first node to fail the one-completion test is $\underline{X}_{11}$. The one-completion value of $\underline{X}_{11}$ is only $g_0(\underline{X1})=16$. Even if this node proved to be feasible, it cannot yield an improved solution. Therefore, it is necessary to return to label E and backtrack once again.

As indicated in Figure B-5 of Appendix B, nodes $\underline{X}_{12}$ through $\underline{X}_{15}$ are processed with backtracking and forward search being used as necessary. Please note that another feasible solution was located at node $\underline{X}_{12}$. A zero-completion of $\underline{X}_{12}$ yields constraint values of $g_1=-1$ and $g_2=0$. However, $\underline{X}_{12}$ has an objective function value of only $g_0(\underline{X0})=15$. Therefore, $\underline{X}_{12}$ does not replace the current interim optimum solution $\underline{X}^*$=(11111.), $g_0^*=17$.

The next node to fail the one-completion test is $\underline{X}_{16}$=(11001.). The one completed form of $\underline{X}_{16}$ is $(\underline{X1})$=(110011) which yields an objective function value of $g_0(\underline{X1})=17$. Although this equals g_0^*, an improved solution is not possible. Therefore, it is necessary to backtrack to another tree branch.

The final node to be processed is $\underline{X}_{30}$=(001...) . This node fails the one-completion test with a value of $g_0(\underline{X1})=15$. The search is ended because further backtracking is not possible. All tree branches have been enumerated, either explicitly or implicitly.

COMPUTER PROGRAM FOR ONE-COMPLETION

An IBM BASIC computer program for solving 0,1 programming problems via the one-completion method is presented in Appendix B. User instructions for the program and examples of program output are also presented in Appendix B.

The program contains one feature not discussed thus far. Following data input and printout of the data matrix, the objective function and constraint equations are reordered with respect to the magnitude of the objective function coefficients. The variable with the largest objective function coefficient is placed first and the other variables follow in order of decreasing magnitude. The reordered matrix is printed and is then used by the program for processing. Appendix B provides examples of input matrix and reordered matrix printout.

The intention of reordering the equations is to speed processing. Because the model has been stated as a maximization and because the one-completion test has been incorporated to halt forward processing when there is no possibility for an improved solution, it seems reasonable to assume that some benefit could be derived from reordering. Reordering will be discussed in much greater detail in the next chapter of this paper.

CHAPTER 3

DETERMINING THE EFFECTIVENESS OF ONE-COMPLETION

As mentioned earlier, the computer program for one-completion presented in Appendix B reorders the objective function and constraint equations before processing begins. The equations are reordered according to the magnitude of the objective function coefficients. For example, the problem

$$\begin{aligned} \max\ & 2X_1 + 6X_2 + 2X_3 + 4X_4 + 3X_5 + 6X_6 \\ \text{s.t. } & 1X_1 - 2X_2 - 3X_3 - 6X_4 + 1X_5 + 2X_6 + 5 \leq 0 \\ & -1X_1 + 3X_2 - 2X_3 - 4X_4 - 2X_5 + 4X_6 + 4 \leq 0 \end{aligned}$$

would be reordered to read

$$\begin{aligned} \max.\ & 6X_2 + 6X_6 + 4X_4 + 3X_5 + 2X_1 + 2X_3 \\ \text{s.t.} & -2X_2 + 2X_6 - 6X_4 + 1X_5 + 1X_1 - 3X_3 + 5 \leq 0 \\ & 3X_2 + 4X_6 - 4X_4 - 2X_5 - 1X_1 - 2X_3 + 4 \leq 0 \end{aligned}$$

Reordering the equations in this manner should speed processing due to the nature of the one-completion test. Once a feasible solution has been located, the one-completion test is performed following each backtracking procedure to determine if the new search tree branch could possibly yield an improved solution. If the new branch fails the one-completion test, all the nodes on the branch have been implicitly enumerated. The one-completion test simply involves (1) augmenting the partial solution vector $\underline{X}_j = (X_1, \ldots, X_j)$ with a subvector of ones for all free variables X_{j+1} through X_n, (2) calculating the objective function value of the one-completed vector, and (3) comparing this value to the current interim optimum solution value. Since the objective function is to be maximized, the one-completed vector value must exceed the current optimum value in order for processing to continue down the current branch.

Remembering the mechanics of the one-completion test should make the value of reordering apparent. If the last few variables have large objective function coefficient values, most nodes will have large one-completed objective function values. This makes it more difficult for nodes to fail the one-completion test. If fewer nodes fail the one-completion test, fewer nodes are enumerated implicitly. For example, given the original configuration of the problem stated above, a one-completion that assigns one values to the last two variables would increase the objective function value by 9.

However, assigning one values to the last two variables of the reordered problem increases its objective function value by only 4.

To determine the effectiveness of equation reordering, the one-completion program has been written in two forms. One contains reordering and one does not. Five problems will be solved by each of the two programs and the results will be compared. The effectiveness of the technique will be determined by comparing processing times and the number of nodes processed explicitly.

As mentioned earlier, an IBM BASIC translation of the Balas additive implicit enumeration program presented by Plane and McMillan is listed in Appendix A. The same five problems mentioned above will be solved via this method and the results will be compared with those obtained with the one-completion program presented in Appendix B. Each problem must be translated to the minimization form to be processed with the Plane and McMillan program. The major items of interest will be the number of nodes enumerated explicitly, the processing time per explicitly enumerated node, and total program execution time.

Fewer nodes should be processed using the Plane and McMillan program. One reason is the nature of the minimization problem versus the maximization problem. In the minimization problem, an effort is made to limit the number of variables added to the solution. If a feasible interim optimum solution is located, backtracking begins immediately. Continued forward search will obviously increase the objective function value and will not yield an improved solution. When a feasible interim optimum solution is located using the one-completion program, forward search continues until the leaf at the bottom of the current branch is processed.

Another factor which should contribute to fewer nodes being explicitly processed with the Plane and McMillan program is the manner in which variables are added to the partial solution vector. The one-completion program simply processes the next node in sequence unless the one-completion test is failed. No attempt is made to select variables which are most likely to contribute to feasibility. In the Plane and McMillan program, each violated constraint is checked to determine if it can be made feasible by adding only those variables with (1) objective function coefficients small enough to prevent the current interim optimum solution value from being exceeded and (2) a positive coefficient in some violated constraint. This set of variables is called Set T. If feasibility is not possible, backtracking occurs. If this test shows that feasibility is possible, a heuristic is used to select the next variable to be added to the partial solution vector. The variable selected is that variable in Set T with the greatest constraint equation coefficient sum.

One factor will increase the number of nodes processed in the Plane and McMillan program, however. This is the reprocessing of nodes as part of the backtracking procedure. As shown in Figure I, following (1) the location of a new optimum feasible solution (box 3), (2) the failure to find any variables to place in Set T (box 6), or (3) the inability to satisfy all infeasible constraints by adding variables in T (box 7), the backtracking procedure begins (box 10). The first backtracking step involves the assignment of a zero-value to the rightmost one valued variable (say X_j). This partial solution vector is then sent to box 2 for processing. However, this node was essentially processed two steps earlier. The only difference being that X_j was free and was assigned a one value because it was the variable in T with the largest coefficient sum. During the backtracking procedure, X_j is assigned a value of zero and cannot be placed in Set T.

While the Plane and McMillan program should have an advantage in the number of nodes processed, the processing time per node should be much shorter for the one-completion program. As can be seen by comparing Figures I and IV, the Plane and McMillan program performs many more computations per node. For each node, the Plane and McMillan program (1) calculates the value of each constraint and places those that are violated in Set V, (2) calculates the objective function value, (3) stores in Set T all free variables that might be capable of contributing to an improved feasible solution, (4) reevaluates all constraints in Set V to determine if they can be made feasible by adding only variables in T, and (5) adds the variable in T with the largest constraint coefficient sum.

The node processing steps for the one-completion program are much simpler. Once a violated constraint is located, constraint calculation stops. The objective function is calculated only if the node is feasible. The one-completion test adds an additional step but it is performed only after a backtracking procedure. These features should give the one-completion program a large advantage in node processing time. They might also give the one-completion program an advantage in processing problems with a large number of constraints.

The five problems used to test the three programs are presented in Figures VA, VIA, VIIA, VIIIA, and IXA. Problem

VA is a maximization translation of a problem used by Plane and McMillan to demonstrate the Balas Implicit Enumeration procedure. Problems VIA through VIIIA are given by Plane and McMillan as examples of problems requiring solution by zero-one programming methods. Finally, Figure IXA was formulated to provide a test problem with a larger number of constraints and variables.

Figures VB through IXB provide the minimization translations of these five problems. To be solved using the Balas Implicit Enumeration program, problems must be written in the form:

$$\text{min.} \quad g_o = \sum_{j=1}^{n} C_j X_j$$

$$\text{s.t.} \quad g_i = \sum_{j=1}^{n} a_{ij} X_j - b_i \geq 0 \qquad i=1,\ldots,m$$

$$C_j \geq 0, X_j = 0,1 \;, \; j=1,\ldots,n$$

CHAPTER 4

TEST RESULTS

Effectiveness of Reordering

The results of the five problem test of the one-completion method with and without problem reordering are given in Table I. These results indicate that the reordering procedure effectively reduces the number of nodes processed without adding significantly to the processing time per node.

Equation reordering reduced the number of nodes processed in each of the five problems. The smallest reduction in nodes processed occurred in the capital budget problem. In this problem, the number of nodes processed was decreased by 9% from 108 to 98. The largest reduction occurred in the Plane and McMillan example problem where the number of nodes processed was reduced 53% from 94 to 44. The average reduction in nodes processed for the five problems was 32%.

The average processing time per explicitly enumerated node was 0.44 sec with reordering and 0.50 sec without. The ranges were 0.34 sec/node -- 0.60 sec/node with reordering and 0.34 sec/node -- 0.62 sec/node without reordering. The time used to calculate processing time per node included only computation time and reordering time. The time required to print the input matrix, the reordered equation matrix, intermediate results and final results was not included. These items are discretionary and are not required for problem solution. The range in node processing times results from such things as the number of backtracking procedures performed, how quickly a feasible solution is found, the magnitude of the interim feasible solutions, how quickly violated constraints are located, the number of feasible solutions located, etc.

Finally, equation reordering effectively reduced the overall processing time for each of the five problems. Processing time was reduced by 41% on average. The smallest reduction in processing time occurred in the capital budget problem. In this problem, only ten fewer nodes were processed as a result of reordering while the processing time per node was .02 seconds higher for the reordering program. As a result, total processing time was reduced by only 3%. The largest reduction in processing time occurred in Problem #5. Here, overall processing time was reduced by 52% as a result of a 37% reduction in nodes processed and a 24% reduction in processing time per node.

One-Completion Method VS. The Basic Balas Additive Method

Table I also contains the results obtained from solving the five test problems with the Balas implicit enumeration procedure presented by Plane and McMillan. The results of the test were mixed. The one-completion method with problem reordering proved to be the quicker method for three of the five problems. Total processing time for all five problems was almost identical for the two methods. Total processing times for the one-completion method and the Balas Implicit Enumeration method were 247.22 seconds and 251.10 seconds respectively.

As was expected, the number of nodes processed using the Balas implicit enumeration method was considerably smaller for all problems. 58% fewer explicitly enumerated nodes were required for solution of the advertising media problem and 80% fewer explicitly enumerated nodes were required for solution of the Plane and McMillan example problem. The three remaining problems fell within this range. The total number of explicitly enumerated nodes required by the one-completion method for solution of all five test problems was 566. Only 187 nodes (67% fewer) were required by the Balas implicit enumeration procedure.

As was also expected, the processing time per node was considerably smaller for the one-completion method. Processing times ranged from 0.34 sec/node for the advertising media problem to 0.60 sec/node for the Plane and McMillan example problem. The average processing time per node for the five test problems was 0.44 seconds. Processing times for the Balas implicit enumeration method ranged from 1.03 sec/node for the advertising media problem to 1.55 sec/node for problem #5. The average processing time per node for all five problems was 1.35 seconds. As a general rule, when the ratio of the number of nodes processed using one-completion to the number of nodes processed using Balas implicit enumeration was less than 3 to 1, one-completion was the quicker method.

CHAPTER 5

CONCLUSIONS

The one-completion algorithm has proven to be a very promising approach to zero-one integer programming, even in these early stages of its development. The version of the one-completion algorithm presented in this paper was shown to be at least as effective as the basic Balas additive algorithm for solving small problems. Some possible improvements to the one-completion algorithm are given below. Considering the number of computations required for each constraint when using the Balas algorithm, the one-completion method may prove to be much more effective in solving larger problems. A larger assignment problem with 20 constraints and 25 variables was attempted with both programs. However, the results were inconclusive. Neither method had solved the problem after two hours of computation on an IBM PC Jr.

An improvement in computing time for the one-completion algorithm presented here might be realized by reversing the order in which the constraints and the objective function are evaluated following the location of a feasible solution. This would prevent the needless evaluation of constraints for nodes which do not offer the possibility of improving the current interim optimum solution. Consideration might also be given to rereversing the order of computation once it has been proven that all the remaining nodes on that branch offer the potential for an improved solution. Of course, following each backtrack these steps must be reversed again.

Perhaps a simpler method of achieving these results could be included in the one-completion procedure as follows: (1) Perform the one-completion test. If the incumbent node fails the one-completion test, backtrack. If the node passes the one-completion test, go to step 2. (2) One-complete the incumbent node one variable at a time. As each variable is added to the partial solution, its objective function value is added to the partial solution value. When the partial solution value finally exceeds the current optimum solution value, it is sent to label A and the feasibility test begins.

One other suggestion which should reduce processing time for larger problems is to begin the one-completion algorithm by relaxing the integrality constraints of the 0-1 problem and using the simplex method to determine the optimum LP solution value. This establishes an upper bound on the 0-1 integral solution value. Following this, the first step in evaluating each partial solution will be to calculate its zero-completed value. If this value exceeds the optimum LP value, there is no feasible solution remaining on this tree branch. The algorithm then moves to the backtracking procedure.

BIBLIOGRAPHY

1. A. M. Geoffrion & R. E. Marsten, "Integer Programming Algorithms: A Framework and State of the Art Survey", Perspectives on Optimization - A Collection of Expository Articles, Addison-Wesley Publishing Co., Inc., Philippines, 1972.

2. Harvey M. Wagner, Principles of Operations Research with Applications to Managerial Decisions, Second Edition, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

3. Stanley Zionts, Linear and Integer Programming, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

4. Arnold Kaufmann, Arnaud Henry-Labordere, Integer and Mixed Programming - Theory and Applications, Academic Press, Inc., New York, 1977.

5. T. C. Hu, Integer Programming and Network Flows, Addison-Wesley Publishing Company, Inc., Philippines, 1970.

6. D. R. Plane, C. McMillan, Jr., Discrete Optimization: Integer Programming and Network Analysis for Management Decisions, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

7. D. R. Anderson, D. J. Sweeney, T. A. Williams, An Introduction to Management Science - Quantitative Approaches to Decision Making, Third Edition, West Publishing Co., St. Paul, Minnesota, 1982.

8. G. A. Gorry & J. F. Shapiro, "An Adaptive Group Theoretic Algorithm for Integer Programming Problems", Management Science, Vol. 17, No. 5, January 1971, pp. 285-306.

9. C. E. Lemke & K. Spielberg, "Direct Search Algorithm for Zero-One and Mixed-Integer Programming", J.ORSA, Sept.-Oct. 1967, pp. 892-915.

10. A. M. Geoffrion, "An Improved Implicit Enumeration Approach for Integer Programming", Operations Research, Vol. 17, No. 3, May-June, 1969, pp. 437-454.

11. E. Balas, "Discrete Programming by the Filter Method", Operations Research, 15, pp. 915-957, 1967.

13. F. Glover, "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem", Operations Research, 13, pp. 879-919, 1965.

14. Mitchell Locks, Ramesh Sharda, Brian LeClaire, "Revision of the Zero-One Implicit Enumeration", Presented at meeting of ORSA-TIMS, Atlanta, Georgia, Nov. 5, 1985.

15. M. R. Greenberg, Applied Linear Programming for the Socio-economic and Environmental Sciences, Acedemic Press, Inc., New York, 1978.

16. P. L. Hammer, E. L. Johnson, B. H. Korle, and G. L. Nemhauser, Studies In Integer Programming, North-Holland Publishing Co., New York, 1977.

TABLE I

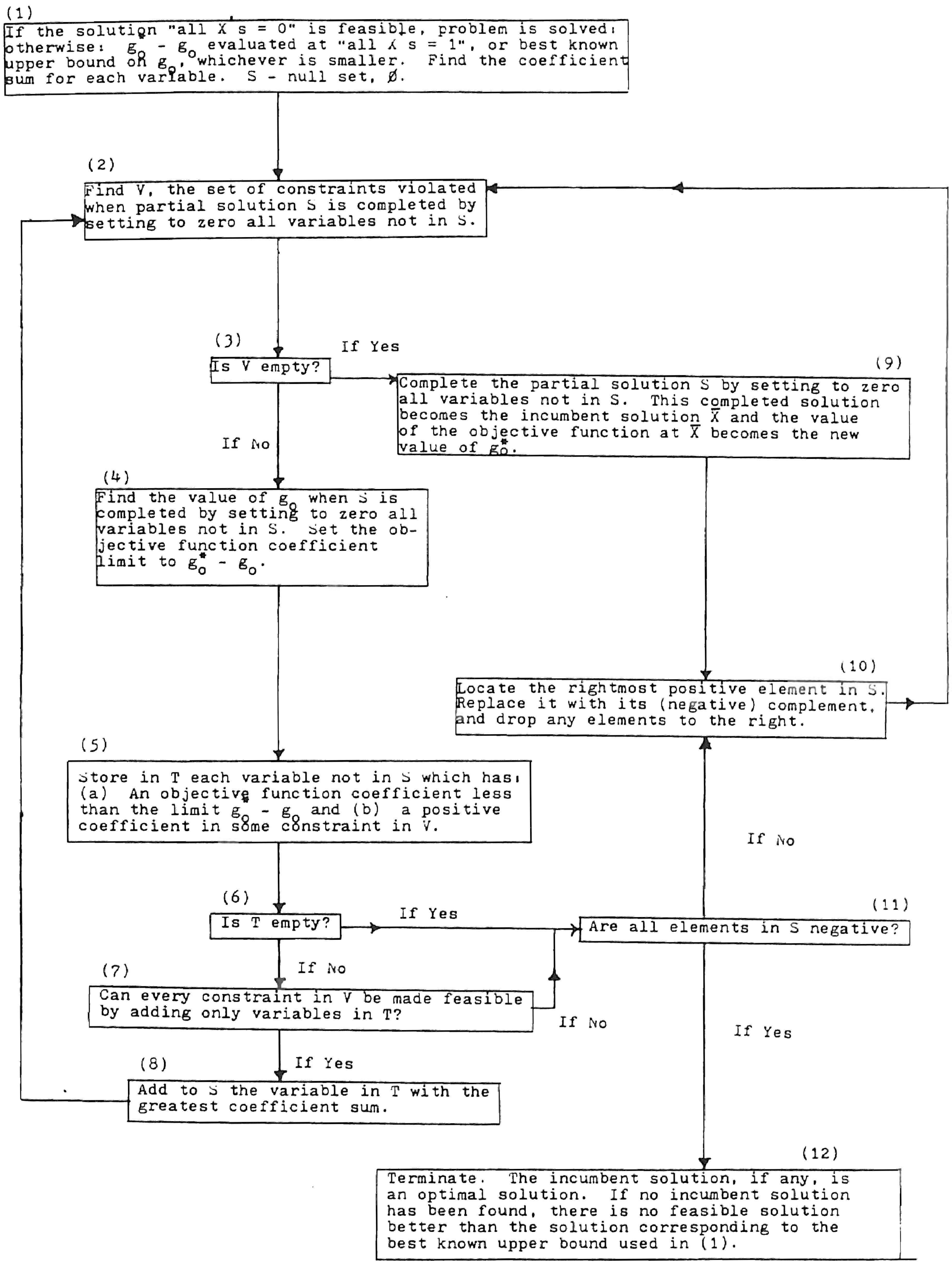
TEST RESULTS

	CONSTRAINTS M	VARIABLES N	ONE-COMPLETION ALGORITHM WITH PROBLEM REORDERING				ONE-COMPLETION WITHOUT REORDERING			BASIC BALAS ADDITIVE ALGORITHM		
			REORDER TIME (SEC)	EXECUTION TIME* (SEC)	NODES PROCESSED	TIME/NODE (SEC)	EXECUTION TIME* (SEC)	NODES PROCESSED	TIME/NODE (SEC)	EXECUTION TIME* (SEC)	NODES PROCESSED	TIME/NODE (SEC)
PLANE AND McMILLAN EXAMPLE PROBLEM	7	10	2.36	26.53	44	0.60	46.02	94	0.49	13.60	9	1.51
KNAPSACK PROBLEM	5	8	1.59	49.73	132	0.38	60.44	176	0.34	53.20	51	1.04
ADVERTISING MEDIA SELECTION	4	6	1.04	12.10	36	0.34	19.66	54	0.36	15.40	15	1.03
CAPITAL BUDGET PROBLEM	7	7	1.48	38.32	96	0.39	39.74	108	0.37	35.02	25	1.40
PROBLEM #5	9	10	2.58	120.54	256	0.47	251.92	406	0.62	134.58	87	1.55
TOTALS			9.05	247.22	566	0.44	417.78	838	0.50	251.80	187	1.35

* EXECUTION TIME INCLUDES COMPUTATION TIME AND TIME REQUIRED TO REORDER EQUATIONS. NO PRINTOUT TIME IS INCLUDED.

FIGURE I

BALAS ADDITIVE ALGORITHM



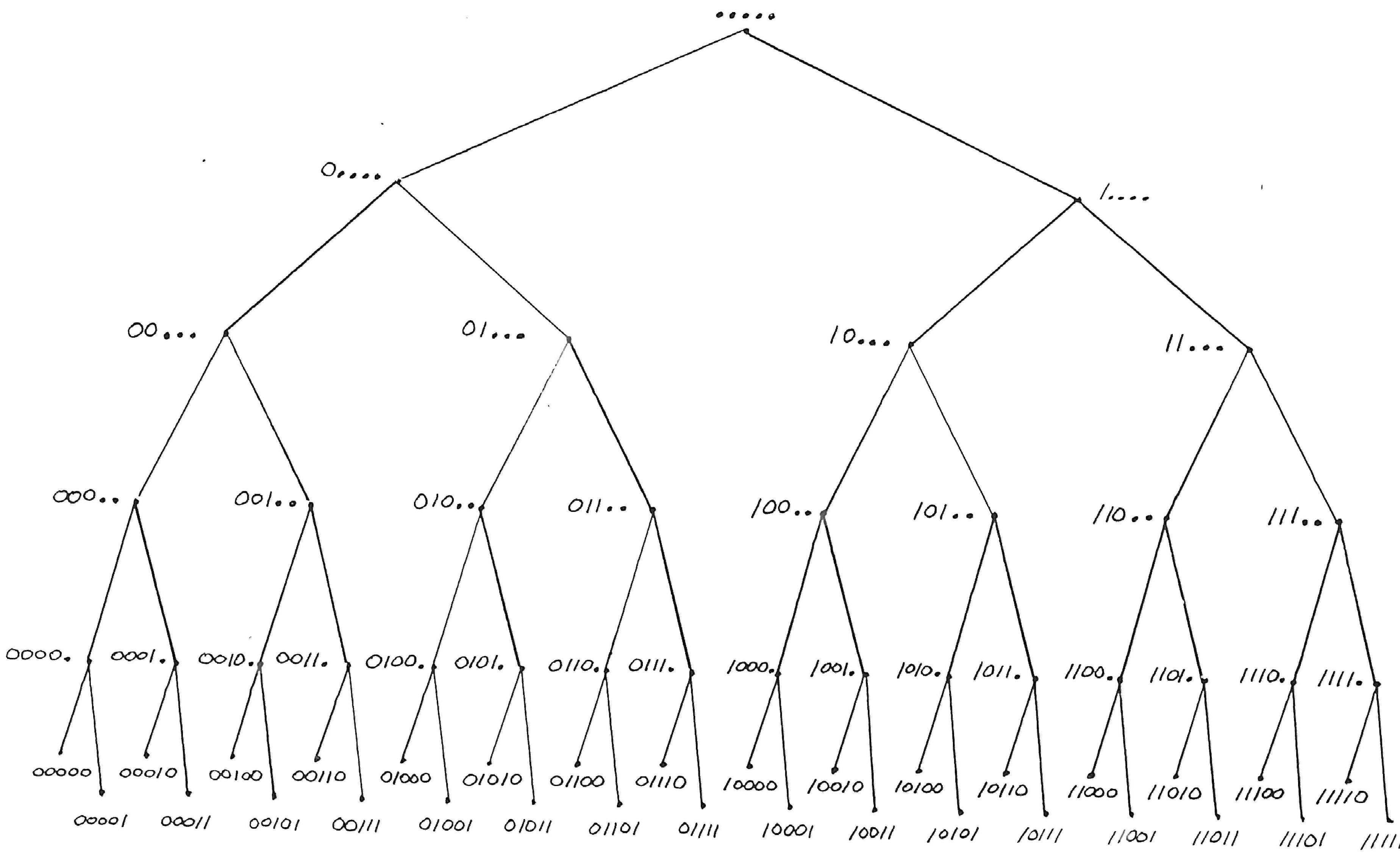


FIGURE II

SEARCH TREE

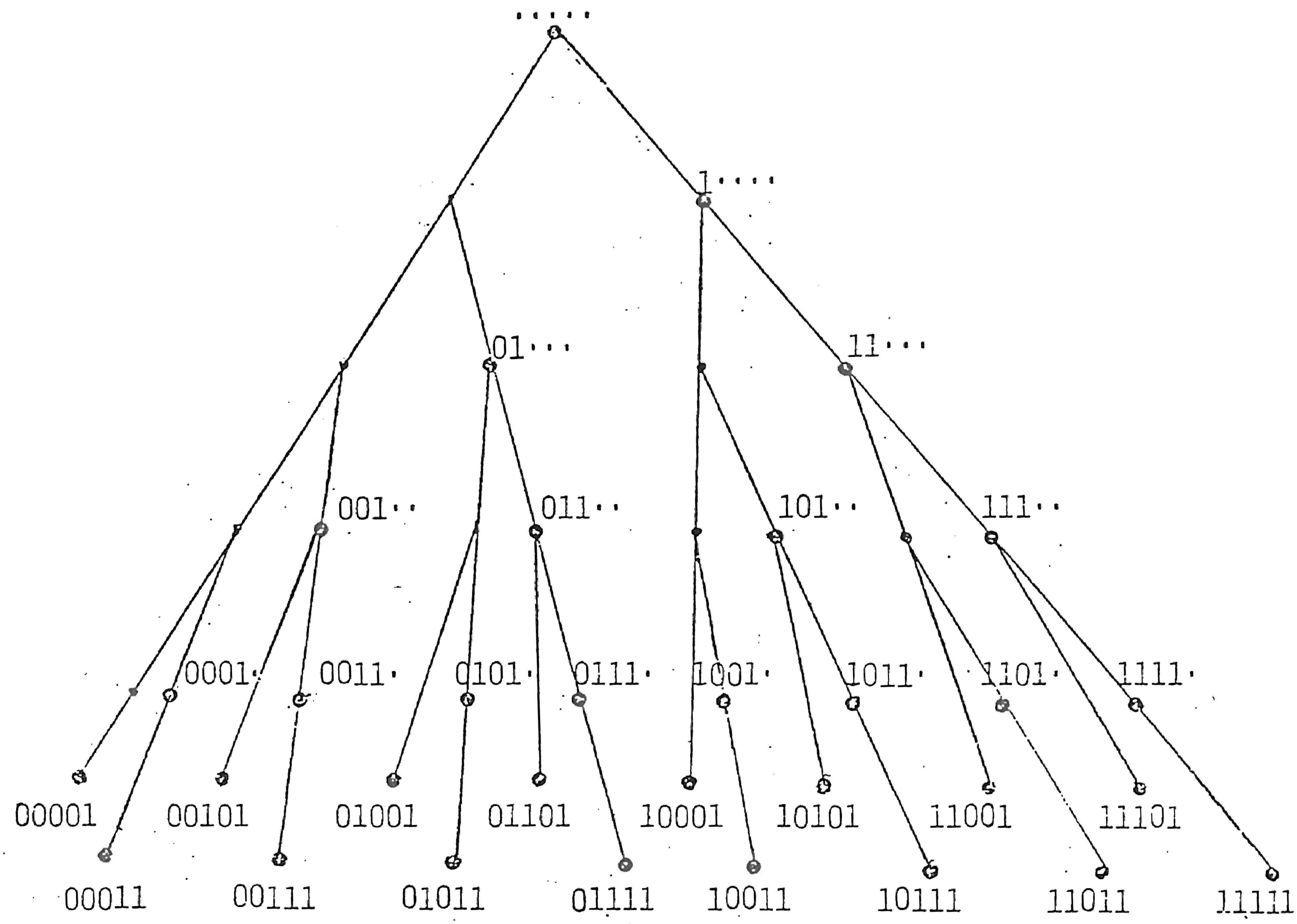


FIGURE III

THE SEARCH TREE FOR

ONE-COMPLETION

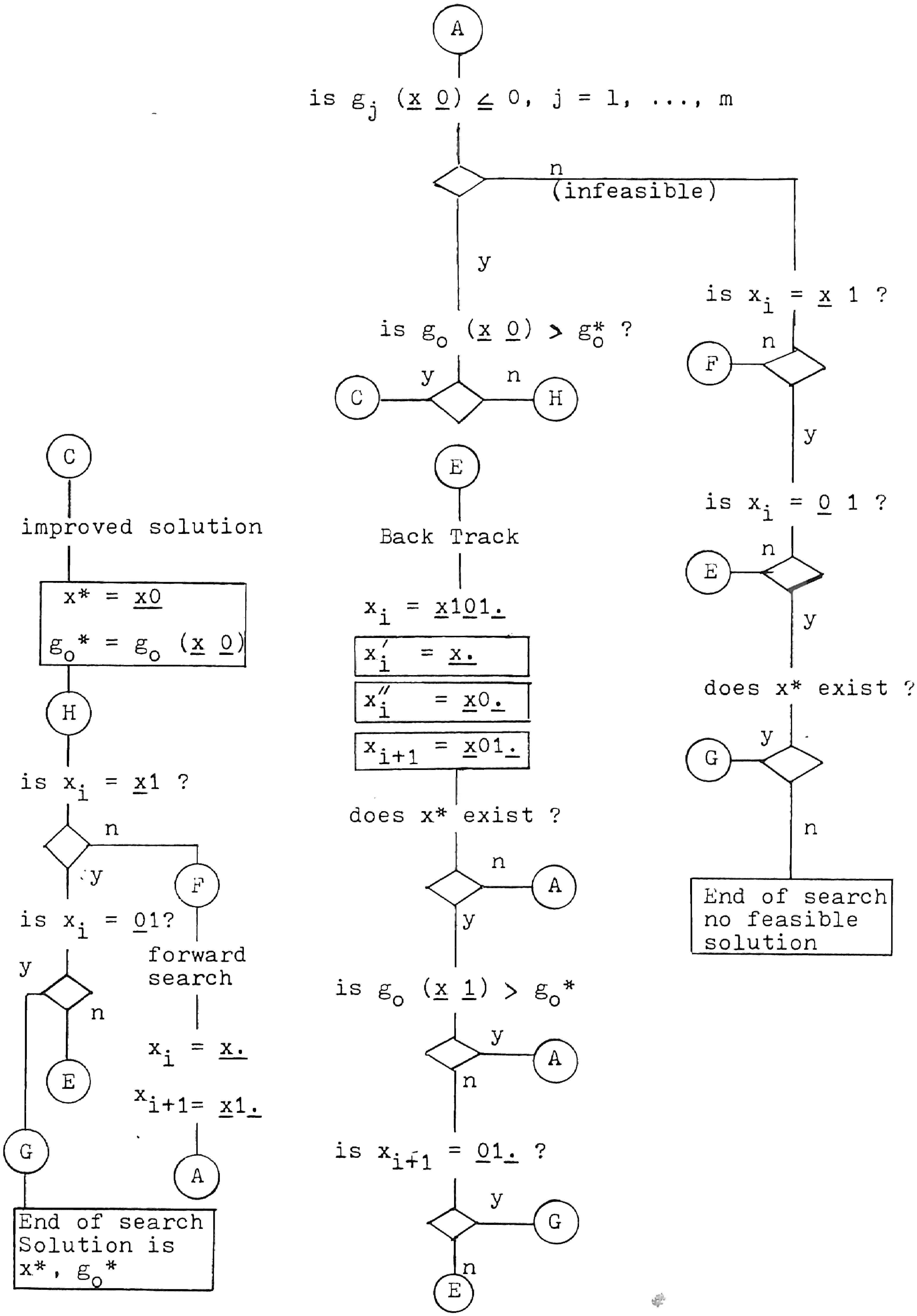


FIGURE IV
ONE-COMPLETION ALGORITHM

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7	X 8	X 9	X10
10.0	7.0	1.0	12.0	2.0	8.0	3.0	1.0	5.0	3.0

CONSTRAINTS

	CONSTANT										
G 1	-19.0	-3.0	12.0	8.0	-1.0	0.0	0.0	0.0	0.0	1.0	-2.0
G 2	-4.0	0.0	-1.0	10.0	0.0	5.0	-1.0	-7.0	-1.0	0.0	0.0
G 3	1.0	-5.0	3.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0	-1.0
G 4	-1.0	5.0	-3.0	-1.0	0.0	0.0	0.0	0.0	-2.0	0.0	1.0
G 5	-18.0	0.0	0.0	4.0	2.0	0.0	5.0	-1.0	9.0	2.0	0.0
G 6	-7.0	0.0	-9.0	0.0	12.0	7.0	-6.0	0.0	-2.0	15.0	-3.0
G 7	-23.0	8.0	-5.0	-2.0	7.0	1.0	0.0	5.0	0.0	10.0	0.0

FIGURE VA

BALAS EXAMPLE PROBLEM

MAXIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7	X 8	X 9	X10
10.0	7.0	1.0	12.0	2.0	8.0	3.0	1.0	5.0	3.0

CONSTRAINTS

	CONSTANT										
G 1	-2.0	-3.0	12.0	8.0	-1.0	0.0	0.0	0.0	0.0	1.0	-2.0
G 2	-1.0	0.0	-1.0	10.0	0.0	5.0	-1.0	-7.0	-1.0	0.0	0.0
G 3	-1.0	-5.0	3.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0	-1.0
G 4	1.0	5.0	-3.0	-1.0	0.0	0.0	0.0	0.0	-2.0	0.0	1.0
G 5	-3.0	0.0	0.0	4.0	2.0	0.0	5.0	-1.0	9.0	2.0	0.0
G 6	-7.0	0.0	-9.0	0.0	12.0	7.0	-6.0	0.0	-2.0	15.0	-3.0
G 7	-1.0	8.0	-5.0	-2.0	7.0	1.0	0.0	5.0	0.0	10.0	0.0

FIGURE VB

BALAS EXAMPLE PROBLEM

MINIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7	X 8
35.0	85.0	135.0	27.0	94.0	10.0	140.0	25.0

CONSTRAINTS

	CONSTANT								
G 1	-30.0	1.0	4.0	17.0	2.0	3.0	4.0	13.0	3.0
G 2	-4.0	0.2	0.6	1.4	0.9	1.3	0.3	2.4	0.6
G 3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0
G 4	-1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
G 5	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0

FIGURE VIA

KNAPSACK PROBLEM

MAXIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7	X 8
35.0	85.0	135.0	27.0	94.0	10.0	140.0	25.0

CONSTRAINTS

	CONSTANT								
G 1	-17.0	1.0	4.0	17.0	2.0	3.0	4.0	13.0	3.0
G 2	-3.7	0.2	0.6	1.4	0.9	1.3	0.3	2.4	0.6
G 3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0
G 4	-1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
G 5	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0

FIGURE VIB

KNAPSACK PROBLEM

MINIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6
200.0	50.0	400.0	300.0	75.0	600.0

CONSTRAINTS

	CONSTANT						
G 1	-700.0	100.0	40.0	300.0	250.0	100.0	400.0
G 2	-1000.0	600.0	0.0	900.0	300.0	100.0	0.0
G 3	-1000.0	200.0	0.0	300.0	700.0	0.0	400.0
G 4	-1000.0	800.0	0.0	100.0	200.0	0.0	0.0

FIGURE VIIA

ADVERTISING MEDIA SELECTION

MAXIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6
200.0	50.0	400.0	300.0	75.0	600.0

CONSTRAINTS

	CONSTANT						
G 1	-490.0	100.0	40.0	300.0	250.0	100.0	400.0
G 2	-900.0	600.0	0.0	900.0	300.0	100.0	0.0
G 3	-600.0	200.0	0.0	300.0	700.0	0.0	400.0
G 4	-100.0	800.0	0.0	100.0	200.0	0.0	0.0

FIGURE VIIB

ADVERTISING MEDIA SELECTION

MINIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7
100.0	150.0	35.0	75.0	125.0	60.0	30.0

CONSTRAINTS

	CONSTANT							
G 1	-450.0	300.0	100.0	0.0	50.0	50.0	200.0	70.0
G 2	-420.0	0.0	300.0	200.0	100.0	300.0	0.0	10.0
G 3	-11.0	4.0	7.0	2.0	6.0	3.0	0.5	0.0
G 4	-1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
G 5	1.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0
G 6	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
G 7	-1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0

FIGURE VIIIA

CAPITAL BUDGET PROBLEM

MAXIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7
100.0	150.0	35.0	75.0	125.0	60.0	30.0

CONSTRAINTS

	CONSTANT							
G 1	-320.0	300.0	100.0	0.0	50.0	50.0	200.0	70.0
G 2	-490.0	0.0	300.0	200.0	100.0	300.0	0.0	10.0
G 3	-11.5	4.0	7.0	2.0	6.0	3.0	0.5	0.0
G 4	-1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
G 5	1.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0
G 6	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
G 7	-1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0

FIGURE VIIIB

CAPITAL BUDGET PROBLEM

MINIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7	X 8	X 9	X10
100.0	150.0	200.0	75.0	50.0	250.0	200.0	400.0	25.0	90.0

CONSTRAINTS

	CONSTANT										
G 1	-20.0	2.0	1.5	7.0	1.0	1.0	5.0	3.0	8.0	0.4	2.0
G 2	-100.0	10.0	8.0	20.0	5.0	6.0	20.0	15.0	25.0	5.0	5.0
G 3	-200.0	10.0	10.0	30.0	10.0	0.0	40.0	30.0	90.0	10.0	0.0
G 4	-15.0	1.0	2.0	2.0	2.0	0.0	6.0	4.0	2.0	1.0	0.0
G 5	-100.0	10.0	10.0	15.0	10.0	0.0	15.0	25.0	18.0	10.0	0.0
G 6	-50.0	2.0	4.0	10.0	8.0	0.0	8.0	5.0	20.0	1.0	0.0
G 7	-500.0	0.0	100.0	50.0	0.0	0.0	150.0	50.0	220.0	10.0	0.0
G 8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	1.0
G 9	-3.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0

FIGURE IXA

PROBLEM #5

MAXIMIZATION

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7	X 8	X 9	X10
100.0	150.0	200.0	75.0	50.0	250.0	200.0	400.0	25.0	90.0

CONSTRAINTS

	CONSTANT										
G 1	-10.9	2.0	1.5	7.0	1.0	1.0	5.0	3.0	8.0	0.4	2.0
G 2	-19.0	10.0	8.0	20.0	5.0	6.0	20.0	15.0	25.0	5.0	5.0
G 3	-30.0	10.0	10.0	30.0	10.0	0.0	40.0	30.0	90.0	10.0	0.0
G 4	-5.0	1.0	2.0	2.0	2.0	0.0	6.0	4.0	2.0	1.0	0.0
G 5	-13.0	10.0	10.0	15.0	10.0	0.0	15.0	25.0	18.0	10.0	0.0
G 6	-8.0	2.0	4.0	10.0	8.0	0.0	8.0	5.0	20.0	1.0	0.0
G 7	-80.0	0.0	100.0	50.0	0.0	0.0	150.0	50.0	220.0	10.0	0.0
G 8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	1.0
G 9	-1.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0

FIGURE IXB

PROBLEM #5

MINIMIZATION

APPENDIX A

```
6 ' ***** BASIC TRANSLATION OF IMPLICIT ENUMERATION PROGRAM PRESENTED *****
7 ' *****                  BY PLANE AND McMILLAN                      *****
8 ' ---------------------------------------------------------------------
9 '
10 DEFINT I-N
20 DIM A(50,50),C(50),B(50),CS(50),W(50,50),IX(50),IS(50),IV(50),IT(50),NOTT(50)
,SUMS(50),IPRINT(50),ISAVE(50,50),ISTFP(50),INUM(50)
21 TIME$="01:00:00"
30 EPS=.000001
70 ITPCK=0:IFEAS=0:ICOUNT=0
75 PRINT CHR$(12)
80 INPUT "NO. CONSTRAINTS, NO.VARIABLES, PRINT INTERVAL: ",M,N,IINT
85 V1=TIMER
90 FOR II=1 TO N
100 IX(II)=9:IS(II)=0:IT(II)=0:NOTT(II)=0:NEXT II
101 FOR I=1 TO M:IV(I)=0:SUMS(I)=0:NEXT I
140 FOR I=1 TO 34
150 IPRINT(I)=0:NEXT
151 V2=TIMER
155 FOR I=1 TO N:READ C(I):NEXT I
160 FOR I=1 TO M:FOR J=1 TO N:READ A(I,J):NEXT J:NEXT I
165 FOR I=1 TO M:READ B(I):NEXT I
170 '
171 '  ************ LINES 200-300 --- DATA INPUT ****************
172 '  **                                                      **
173 '  ** These lines have been reserved for data entry.  Data **
174 '  ** is entered using the basic data statement.  The order**
175 '  ** of data input must be (1) objective function coef-   **
176 '  ** ficients, (2) constraint coefficients, and (3) con-  **
177 '  ** straint constants.                                   **
178 '  **                                                      **
179 '  **********************************************************
180 '
301 V3=TIMER
310 ZBAR=0!
320 FOR I=1 TO N
330 ZBAR=ZBAR +C(I):NEXT
340 FZBAR=ZBAR
350 FOR J=1 TO N: CS(J)=0!
360 FOR I=1 TO M
370 CS(J)=CS(J)+A(I,J):NEXT I:NEXT J
371 V4=TIMER
372 PV1=0
373 '
374 '
375 '   ************** this section prints out matrix input ***************
376 '
377 '
380 LPRINT CHR$(12)
390 LPRINT SPACE$(14);"OBJECTIVE FUNCTION":LPRINT
400 LPRINT SPACE$(16);"X";1;
410 FOR I=2 TO N
420 LPRINT SPACE$(4);"X";I;
```

```
430 NEXT:LPRINT:LPRINT
440 LPRINT SPACE$(14);
450 FOR I=1 TO N
460 LPRINT USING "####.#  ";C(I);
470 NEXT:LPRINT:LPRINT:LPRINT
480 LPRINT SPACE$(14);"CONSTRAINTS":LPRINT
490 LPRINT SPACE$(5);"CONSTANT":LPRINT
500 FOR I=1 TO M
510 LPRINT "G";I;
515 LPRINT USING "  ####.#";B(I);
520 FOR J=1 TO N
530 LPRINT USING "  ####.#";A(I,J);
540 NEXT J:LPRINT:LPRINT
550 NEXT I
560 NUMB=0:NS=0
570 LPRINT CHR$(12)
580 LPRINT SPACE$(21);"*";SPACE$(21);"*";SPACE$(30);"*VAR"
590 LPRINT "PARTIAL SOLUTION (S) *VIOLATED CONSTRAINTS *     VARIABLE IN SET (T)
     *ADD"
600 X$=STRING$(78,42)
610 LPRINT X$
611 V5=TIMER
612 '                          *****  STEP 2  *****
613 '  ***** Find V, the set of constraints violated when partial solution ****
614 '  ***** S is completed by setting to zero all variables not in the     ****
615 '  ***** set S.                                                         ****
616 '  ***** Find FP, the value of F when S is completed by setting to      ****
617 '  ***** zero all variables not in S.                                   ****
618 '
620 V6=TIMER
621 IF NUMB<=0 THEN GOTO 680
630 IP=7
640 IF NS>7 THEN GOTO 660
650 IP=NS
660 FOR I=1 TO IP
670 IPRINT(I)=IS(I):NEXT
671 PV1=PV1+TIMER-V6
680 FP=0!
690 NW=0
700 IF NS<=0 THEN GOTO 790
710 FOR J=1 TO NS
720 IF IS(J)<=0 THEN GOTO 780
730 NW=NW+1
740 JJ=IS(J)
750 FOR I=1 TO M
760 W(I,NW)=A(I,JJ):NEXT I
770 FP =FP+C(JJ)
780 NEXT J
790 NW=NW+1
800 FOR I=1 TO M
810 W(I,NW)=B(I):NEXT
820 MV=0
830 FOR I=1 TO M
840 SUMS(I)=0!
```

```
850 FOR J=1 TO NW
860 SUMS(I)=SUMS(I)+W(I,J):NEXT J
870 IF (SUMS(I)+EPS))=0 THEN GOTO 890
880 MV=MV+1: IV(MV)=I
890 NEXT I
900 '
910 '   *********              STEP 3              **********
920 '   *********        Is the set V empty        **********
921 '   *********                                  **********
922 '   *********      If yes -- go to step 9      **********
923 '   *********                                  **********
924 '   *********      If no  -- go to step 4      **********
925 '
926 '
930 IF MV<=0 THEN GOTO 1780
940 IP=7
950 IF MV>7 THEN GOTO 970
960 IP=MV
970 V16=TIMER
971 FOR I=1 TO IP
980 IPRINT(I+11)=IV(I):NEXT
981 PV1=PV1+TIMER-V16
990 '
1000 '  *********              STEP 4              **********
1010 '  ********* Set the objective function limit  **********
1011 '  ********* to ZBAR -FP.                       **********
1012 '
1020 CLIM=ZBAR-FP
1030 NW=0:NT=0:IT(1)=0
1040 '
1050 '  ***                    STEP 5                    ***
1060 '  *** Store in set T each variable not in S which has ***
1061 '  ***                                               ***
1062 '  *** 1.An obj funct. coefficient less than the limit ***
1063 '  *** 2.A positive coef. in some constraint in V      ***
1064 '
1070 FOR J=1 TO N
1080 NOTT(J)=0:NEXT
1090 IF NS<=0 THEN GOTO 1160
1100 FOR J=1 TO NS
1110 ITEMP=IS(J)
1120 IF ITEMP=>0 THEN GOTO 1140
1130 ITEMP=-ITEMP
1140 NOTT(ITEMP)=1
1150 NEXT J
1160 FOR J=1 TO N
1170 IF NOTT(J)>0 THEN GOTO 1290
1180 IF CLIM<=C(J) THEN GOTO 1290
1190 FOR I=1 TO MV
1200 ITEMP=IV(I)
1210 IF A(ITEMP,J)>0 THEN GOTO 1240
1220 NEXT I
1230 GOTO 1290
1240 NT=NT+1
```

```
1250 IT(NT)=J
1260 NW=NW+1
1270 FOR I=1 TO M
1280 W(I,NW)=A(I,J):NEXT I
1290 NEXT J
1300 IP=10
1310 IF NT>10 THEN GOTO 1330
1320 IP=NT
1330 V26=TIMER
1331 FOR I=1 TO IP
1340 IPRINT(I+22)=IT(I)
1350 NEXT
1351 PV1=PV1+TIMER-V26
1360 '
1370 '  ***                      STEP 6                       ***
1380 '  ***                Is the set T empty                 ***
1381 '  ***                                                   ***
1382 '  *** If yes -- set ITPCK=1 and go to output, then go ***
1383 '  ***             to step 11 (backtrack).              ***
1384 '  *** If no  -- go to step 7                           ***
1385 '
1390 IF NT>0 THEN GOTO 1440
1400 ITPCK=1:JMAX=0:GOTO 1920
1410 '
1420 '  ***                      STEP 7                       ***
1430 '  *** Can every constraint in V be made feasible by    ***
1431 '  *** adding only variables in T                       ***
1432 '  ***                                                   ***
1433 '  *** If no  -- set ITPCK=1 and go to output, then go ***
1434 '  ***             to step 11 (backtrack).              ***
1435 '  *** If yes -- go to step 8                           ***
1436 '
1440 FOR I=1 TO MV
1450 ITEMP=IV(I)
1460 FOR J=1 TO NW
1470 IF W(ITEMP,J)<=0 THEN GOTO 1490
1480 SUMS(ITEMP)=SUMS(ITEMP)+W(ITEMP,J)
1490 NEXT J
1500 IF SUMS(ITEMP)>=-EPS THEN GOTO 1550
1510 IPRINT(34)=ITEMP
1520 ITPCK=1
1530 JMAX=0
1540 GOTO 1920
1550 NEXT I
1560 '
1570 '  ***                      STEP 8                       ***
1580 '  ***                                                   ***
1581 '  *** Add to S the variable in T with the greatest     ***
1582 '  *** coeff. sum, go to output, then go to step 2      ***
1583 '
1590 JMAX=IT(1)
1600 CSMAX=CS(JMAX)
1610 IF NT<2 THEN GOTO 1700
1620 FOR J=2 TO NT
```

```
1630 JTEMP=IT(J)
1640 IF CS(JTEMP)<CSMAX THEN GOTO 1690
1650 IF CS(JTEMP)>CSMAX THEN GOTO 1670
1660 IF C(JTEMP)=>C(JMAX) THEN GOTO 1690
1670 JMAX=JTEMP
1680 CSMAX=CS(JTEMP)
1690 NEXT J
1700 GOTO 1920
1710 NS=NS+1
1720 IS(NS)=JMAX
1730 NUMB=NUMB +1
1740 GOTO 620
1750 '
1760 '  ***                        STEP 9                        ***
1770 '  *** Complete the partial solution S by setting to    ***
1771 '  *** zero all variables not in S.  This completed      ***
1772 '  *** solution becomes the incumbent solution x-bar,  ***
1773 '  *** and the value of the objective function at       ***
1774 '  *** x-bar becomes the new value of ZBAR              ***
1775 '
1780 FOR J=1 TO N
1790 IX(J)=0:NEXT
1800 ZBAR=0
1810 FOR J=1 TO NS
1820 JTEMP=IS(J)
1830 IF JTEMP=<0 THEN GOTO 1860
1840 IX(JTEMP)=1
1850 ZBAR=ZBAR+C(JTEMP)
1860 NEXT
1870 '
1880 '  *** Feasible soln encountered - set IFEAS=1 to save ***
1890 '
1900 IFEAS=1:JMAX=0:CLIM=0
1910 '
1911 '  ***                     OUTPUT SECTION                   ***
1912 '  ***                                                      ***
1913 '  *** Steps are printed according to the interval        ***
1914 '  *** specified by the user                               ***
1915 '
1920 V7=TIMER
1921 ICK=(NUMB/IINT)*IINT-NUMB
1930 IF ICK<>0 THEN GOTO 2070
1940 FOR I=1 TO 7
1950 LPRINT USING  "###";IPRINT(I);
1960 NEXT
1970 LPRINT "*";
1980 FOR I=12 TO 18
1990 LPRINT USING "###";IPRINT(I);
2000 NEXT
2010 LPRINT "*";
2020 FOR I=23 TO 32
2030 LPRINT USING "###";IPRINT(I);
2040 NEXT
```

```
2050 LPRINT "*";
2060 LPRINT USING "###";JMAX
2070 FOR I=1 TO 34
2080 IPRINT(I)=0
2090 NEXT
2091 PV1=PV1+TIMER-V7
2100 IF IFEAS=>1 THEN GOTO 2150
2110 IF ITPCK<1 THEN GOTO 1710
2120 '
2130 '  ***                    STEP 11                      ***
2131 '  ***   Are all elements in the set S negative        ***
2132 '  ***                                                 ***
2133 '  *** If not -- locate the rightmost positive element  ***
2134 '  *** in S.  Replace it with its complement (-) and    ***
2135 '  *** drop any elements to the right. then go to step 2 **
2136 '  ***                                                 ***
2137 '  *** If so  -- terminate                             ***
2138 '
2140 '
2150 NEWS=NS
2160 FOR J=1 TO NS
2170 JJ=NS-J+1
2180 IF IS(JJ)>0 THEN GOTO 2220
2190 NEWS=NEWS-1
2200 NEXT J
2210 GOTO 2340
2220 IS(JJ)=-IS(JJ)
2230 NS=NEWS
2240 IF IFEAS<1 THEN GOTO 2320
2250 IF ITPCK=>1 THEN GOTO 2320
2260 IF 50<=ICOUNT THEN GOTO 2320
2270 ICOUNT=ICOUNT + 1
2280 ISTEP(ICOUNT)=NUMB
2290 FOR I=1 TO N
2300 ISAVE(ICOUNT,I)=IX(I)
2310 NEXT
2320 IFEAS=0
2330 ITPCK=0:NUMB=NUMB+1:GOTO 620
2340 V9=TIMER
2341 '
2342 '  ***                    STEP 12                      ***
2343 '  *** Terminate -- the incumbent soln, if any, is opt. ***
2344 '  *** If none   -- there is no feasible solution better **
2345 '  ***               than the initial value of ZBAR      ***
2346 '
2349 LPRINT CHR$(12):LPRINT:LPRINT
2350 IF IX(1)<9 THEN GOTO 2380
2360 LPRINT "THERE IS NO FEASIBLE SOLUTION"
2370 GOTO 2520
2380 FOR I=1 TO ICOUNT
2390 LPRINT "FEASIBLE SOLUTION, STEP  ";ISTEP(I);SPACE$(3);
2400 FOR J=1 TO N
2410 LPRINT USING " #";ISAVE(I,J);
2420 NEXT J
```

```
2430 LPRINT
2440 NEXT I
2450 LPRINT:LPRINT:LPRINT
2460 LPRINT SPACE$(5);"OPTIMAL SOLUTION";SPACE$(3);
2470 FOR I=1 TO N
2480 LPRINT USING " #";IX(I);
2490 NEXT:LPRINT:LPRINT
2500 LPRINT SPACE$(5);"OPTIMAL VALUE OF OBJECTIVE FUNCTION= ";
2510 LPRINT USING "  #####.###";ZBAR
2520 VV=V2-V1+V4-V3+V9-V5
2522 LPRINT:LPRINT:LPRINT "A. Total execution time excluding input printout (sec
) = ";VV
2523 LPRINT "B. Time required to print results (sec) = ";PV1
2524 LPRINT "C. Real program execution time (A - B) = ";(VV-PV1)
2530 END
```

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6	X 7
100.0	150.0	35.0	75.0	125.0	60.0	30.0

CONSTRAINTS

	CONSTANT							
G 1	-320.0	300.0	100.0	0.0	50.0	50.0	200.0	70.0
G 2	-490.0	0.0	300.0	200.0	100.0	300.0	0.0	10.0
G 3	-11.5	4.0	7.0	2.0	6.0	3.0	0.5	0.0
G 4	-1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
G 5	1.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0
G 6	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
G 7	-1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0

FIGURE A-1

BALAS EXAMPLE PROBLEM INPUT

MATRIX

PARTIAL SOLUTION (S)							VIOLATED CONSTRAINTS							VARIABLE IN SET (T)										VAR ADD
0	0	0	0	0	0	0*	1	2	3	4	7	0	0*	1	2	3	4	5	6	7	0	0	0*	2
2	0	0	0	0	0	0*	1	2	3	6	7	0	0*	1	3	4	5	6	7	0	0	0	0*	5
2	5	0	0	0	0	0*	1	3	6	0	0	0	0*	1	3	4	6	7	0	0	0	0	0*	1
2	5	1	0	0	0	0*	5	6	0	0	0	0	0*	3	0	0	0	0	0	0	0	0	0*	0
2	5	-1	0	0	0	0*	1	3	6	0	0	0	0*	3	4	6	7	0	0	0	0	0	0*	3
2	5	-1	3	0	0	0*	1	0	0	0	0	0	0*	4	6	7	0	0	0	0	0	0	0*	6
2	5	-1	3	6	0	0*	0	0	0	0	0	0	0*	0	0	0	0	0	0	0	0	0	0*	0
2	5	-1	3	-6	0	0*	1	0	0	0	0	0	0*	7	0	0	0	0	0	0	0	0	0*	0
2	5	-1	-3	0	0	0*	1	3	6	0	0	0	0*	4	6	7	0	0	0	0	0	0	0*	0
2	-5	0	0	0	0	0*	1	2	3	6	7	0	0*	1	3	4	6	7	0	0	0	0	0*	1
2	-5	1	0	0	0	0*	2	3	5	6	7	0	0*	3	4	6	7	0	0	0	0	0	0*	0
2	-5	-1	0	0	0	0*	1	2	3	6	7	0	0*	3	4	6	7	0	0	0	0	0	0*	3
2	-5	-1	3	0	0	0*	1	3	7	0	0	0	0*	4	6	7	0	0	0	0	0	0	0*	6
2	-5	-1	3	6	0	0*	1	3	0	0	0	0	0*	4	7	0	0	0	0	0	0	0	0*	4
2	-5	-1	3	6	4	0*	0	0	0	0	0	0	0*	0	0	0	0	0	0	0	0	0	0*	0
2	-5	-1	3	6	-4	0*	1	3	0	0	0	0	0*	7	0	0	0	0	0	0	0	0	0*	0
2	-5	-1	3	-6	0	0*	1	3	7	0	0	0	0*	4	7	0	0	0	0	0	0	0	0*	0
2	-5	-1	-3	0	0	0*	1	2	3	6	7	0	0*	4	6	7	0	0	0	0	0	0	0*	0
-2	0	0	0	0	0	0*	1	2	3	4	7	0	0*	1	3	4	5	6	7	0	0	0	0*	5
-2	5	0	0	0	0	0*	1	2	3	4	0	0	0*	1	3	4	6	7	0	0	0	0	0*	1
-2	5	1	0	0	0	0*	2	3	0	0	0	0	0*	3	4	6	7	0	0	0	0	0	0*	3
-2	5	1	3	0	0	0*	3	0	0	0	0	0	0*	0	0	0	0	0	0	0	0	0	0*	0
-2	5	1	-3	0	0	0*	2	3	0	0	0	0	0*	4	6	7	0	0	0	0	0	0	0*	0
-2	5	-1	0	0	0	0*	1	2	3	4	0	0	0*	3	4	6	7	0	0	0	0	0	0*	0
-2	-5	0	0	0	0	0*	1	2	3	4	7	0	0*	1	3	4	6	7	0	0	0	0	0*	0

FIGURE A-2

BALAS EXAMPLE

INTERMEDIATE RESULTS

```
FEASIBLE SOLUTION, STEP   6     0 1 1 0 1 1 0
FEASIBLE SOLUTION, STEP   14     0 1 1 1 0 1 0


     OPTIMAL SOLUTION   0 1 1 1 0 1 0

     OPTIMAL VALUE OF OBJECTIVE FUNCTION=     320.000


A. Total execution time excluding input printout (sec) =  54.48999
B. Time required to print results (sec) =  19.46997
C. Real program execution time (A - B) =  35.02002
```

FIGURE A-3

BALAS EXAMPLE

FINAL RESULTS

APPENDIX B

USER INSTRUCTIONS FOR THE ONE-COMPLETION IMPLICIT ENUMERATION COMPUTER PROGRAM

An IBM BASIC computer program for solving 0,1 programming problems via the one-completion implicit enumeration method is attached. To use the program, a problem must be written in the form:

$$\text{Max } g_o = \sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } \quad g_i = \sum_{j=1}^{n} a_{ij} X_j - b_i \leq 0 \qquad i=1,\ldots,m$$

$$C_j \geq 0, X_j = 0,1 \;, \; j=1,\ldots,n$$

where m = The number of constraints
n = The number of variables
C_j, a_{ij}, b_i = Numerical coefficients

The following rules can be used to transform a problem, or model, to the form shown above:

1. To convert a problem from a minimization to a maximization, multiply the objective function, g_o, by -1.

2. If any objective function coefficient, C_j, is negative, substitute $X'_j = 1-X_j$ for the corresponding variable. Remember that this substitution must be made in each of the constraint equations as well.

3. If a constraint equation, g_i i=1,...,m, is greater than or equal to zero, multiply by -1.

4. Convert any constraint shown as an equality to two inequalities. For example:

$$gi = \sum_{j=1}^{n} a_{ij} X_j - b_i = 0$$

becomes
$$g_{i1} = \sum_{j=1}^{n} a_{ij} X_j - b_i \geq 0$$

$$g_{i2} = \sum_{j=1}^{n} a_{ij} X_j - b_i \leq 0$$

Program Execution

Program execution consists of three parts: (1) beginning execution, (2) data entry, and (3) resuming execution. Each of these parts is described in greater detail below.

(1) BEGINNING EXECUTION:

Program execution begins by simply entering the Basic command 'RUN'. This allows only the first eight lines of the program to be executed. This portion of the program simply places a request for data input on the monitor. At this point, the user is back in the Basic edit mode. The request for data will appear as follows.

PLEASE ENTER (1) THE OBJECTIVE FUNCTION COEFFICIENTS, (2) THE COEFFICIENTS OF ALL CONSTRAINT EQUATION VARIABLES, AND (3) ALL CONSTRAINT EQUATION CONSTANTS. LINES 3000-4000 HAVE BEEN RESERVED FOR DATA INPUT. FOR EACH LINE OF DATA FIRST ENTER A LINE NUMBER FOLLOWED BY THE WORD DATA (3000 DATA). ALL DATA ITEMS MUST BE SEPARATED BY COMMAS. EACH LINE MUST BE LESS THAN 254 CHARACTERS IN LENGTH. WHEN DATA ENTRY IS COMPLETE, ENTER 'RUN 100' TO CONTINUE EXECUTION.

(2) DATA ENTRY:

Lines 3000 through 4000 have been reserved for data entry. After the program requests data entry, program execution stops and the user is back in the Basic edit mode. Therefore, all Basic edit commands can be used for data entry.

As stated above, the order of data input must be (1) the objective function coefficients, (2) the coefficients of all constraint equation variables, and (3) all constraint equation constants. An example of proper data entry is given below.

Example Problem:

$$\max \quad g_0 = 2X_1 + 6X_2 + 2X_3 + 4X_4 + 3X_5 + 6X_6$$

$$\text{s.t.} \quad g_1 = X_1 - 2X_2 - 3X_3 - 6X_4 + X_5 + 2X_6 + 5 \leq 0$$

$$g_2 = -X_1 + 3X_2 - 2X_3 - 4X_4 - 2X_5 + 4X_6 + 4 \leq 0$$

Data Entry:

```
3000 DATA  2,6,2,4,3,6
3010 DATA  1,-2,-3,-6,1,2
3020 DATA  -1,3,-2,-4,-2,4
3030 DATA  5,4
```

Once entered, these lines of data become a part of the program. Read statements are used to assign these values to specific program variables. If the user desires to retain the data in the program file for later use, simply save the file after the data has been entered.

(3) RESUMING EXECUTION:

When data entry is complete, the user must enter 'RUN 100' to continue program execution. This sends the program to line 100 where computation begins. The program then requests that the user enter the number of constraints and the number of variables. For example, the problem given above has two constraints (g_1 and g_2) and six variables (X_1, X_2, ..., X_6).

Program Printout

The output for the example problem discussed earlier is attached. Figure B-1 is simply a printout of the data matrix input as supplied by the user. Figure B-2 is a printout of the data matrix used by the program for processing. This matrix is derived by reordering the objective function and constraint equations according to the magnitude of the objective function coefficients. Figure B-3 shows the intermediate program output and Figure B-4 gives the problem solution.

```
8 '     ############# TREE-SEARCH ONE-COMPLETION VERSION OF ############
9 '     #############       BALAS IMPLICIT ENUMERATION       ############
10 '
11 '           DATA INPUT - Lines 25 through 64 request data input.
12 '           Following the execution of line 64, the user is back
13 '           in the basic edit mode.  Lines 3000-4000 have been
14 '           reserved for data input.  Once data input is complete,
15 '           the user resumes program execution at line 100.
16 '
25 PRINT CHR$(12)
43 PRINT "PLEASE ENTER (1) THE OBJECTIVE FUNCTION COEFFICIENTS, (2) THE COEFFICI
ENTS OF"
46 PRINT "ALL CONSTRAINT EQUATION VARIABLES, AND (3) ALL CONSTRAINT EQUATION CON
STANTS."
49 PRINT "LINES 3000-4000 HAVE BEEN RESERVED FOR DATA INPUT.  FOR EACH LINE OF D
ATA,"
52 PRINT "FIRST ENTER A LINE NUMBER FOLLOWED BY THE WORD DATA (3000 DATA).  ALL
DATA"
55 PRINT "ITEMS MUST BE SEPARATED BY COMMAS. EACH LINE MUST BE LESS THAN 254 CHA
RACTERS"
58 PRINT "IN LENGTH. WHEN DATA ENTRY IS COMPLETE, ENTER 'RUN 100' TO CONTINUE EX
ECUTION."
64 END
79 '
80 '           LINES 100 THROUGH 370 - The user is requested to input
81 '           the number of constraints (M), and the number of vari-
82 '           ables (N).  With this information, the program reads the
83 '           objective function coefficients (C(I)), the constraint
84 '           coefficients (A(I,J)), and the constraint constants (B(I)).
85 '           these values are then printed in tabular form.
86 '
100 DEFINT I-N
105 OPEN "lpt1:" AS #1
110 WIDTH #1,200
120 DIM A(50,50),C(50),B(50),IX(50),IXSTAR(50),G(50),CNEW(50),IC(50),XA(50,50),I
X1COMP(50),IXPRINT(50)
125 PRINT CHR$(12)
130 INPUT "NO. CONSTRAINTS, NO.VARIABLES : ",M,N
150 FOR I=1 TO N
152 READ C(I):NEXT I
154 FOR I=1 TO M:FOR J=1 TO N:READ A(I,J):NEXT J:NEXT I
156 FOR I=1 TO M:READ B(I):NEXT I
200 LPRINT SPACE$(25);"OBJECTIVE FUNCTION":LPRINT
210 LPRINT SPACE$(27);"X";1;
220 FOR I=2 TO N
230 PRINT #1,SPACE$(7);"X";:PRINT #1,USING "##";I;
240 NEXT:LPRINT:LPRINT
250 LPRINT SPACE$(25);
260 FOR I=1 TO N
270 PRINT #1,USING "######.#  ";C(I);
280 NEXT:LPRINT:LPRINT:LPRINT
290 LPRINT SPACE$(25);"CONSTRAINTS":LPRINT
300 LPRINT SPACE$(15);"CONSTANT":LPRINT
310 FOR I=1 TO M
```

```
320 LPRINT SPACE$(10);"G";:LPRINT USING "##";I;
330 LPRINT USING "  ######.#";B(I);
340 FOR J=1 TO N
350 PRINT #1,USING "  ######.#";A(I,J);
360 NEXT J:LPRINT:LPRINT
370 NEXT I
372 PV1=0
373 TIME$="01:00:00"
374 V1=TIMER
375 '          LINES 380 THROUGH 580 - The model's equation variables are
376 '          rearranged according to the magnitude of the obj. function
377 '          coefficients.  The reordered obj. function coef.s are placed
378 '          in CNEW and the reordered constraint coef.s are placed in XA.
379 '          The new variable order is recorded in IC.  Lines 380 -420
380 '          locate the largest obj. funct. coef. and place it in CNEW(1).
381 '          Lines 430 - 525 reorder the remaining obj. funct. coef.s and
382 '          the constraints are reordered in lines 530 - 580.
383 '
388 CNEW(1)=C(1)
390 IC(1)=1
400 FOR I=2 TO N
410 IF CNEW(1)=> C(I) THEN GOTO 420
413 CNEW(1)=C(I)
416 IC(1)=I
420 NEXT I
430 FOR I=2 TO N
440 II=I-1
450 CNEW(I)=-1
460 FOR J=1 TO N
470 IF CNEW(II)<C(J) THEN GOTO 520
480 IF IC(II)=J THEN GOTO 520
490 IF C(J)<=CNEW(I) THEN GOTO 520
491 IF CNEW(II)=C(J) THEN GOTO 512
500 CNEW(I)=C(J)
510 IC(I)=J
511 GOTO 520
512 IF IC(II)=>J THEN GOTO 520
513 CNEW(I)=C(J)
514 IC(I)=J
520 NEXT J
525 NEXT I
530 FOR I=1 TO M
540 FOR J=1 TO N
550 JJ=IC(J)
560 XA(I,J)=A(I,JJ)
570 NEXT J
580 NEXT I
581 V2=TIMER
584 '
585 '          Lines 590 THROUGH 820 -  Reordered equation printout
586 '
590 LPRINT CHR$(12)
600 LPRINT
610 PRINT #1,SPACE$(10);"THE OBJECTIVE FUNCTION AND CONSTRAINT EQUATION VARIABLE
S HAVE BEEN"
```

```
620 PRINT #1,SPACE$(10);"REARRANGED IN ORDER TO SPEED PROCESSING.  THE ACTUAL EQ
UATIONS USED
630 PRINT #1,SPACE$(10);"FOR PROCESSSING APPEAR AS FOLLOWS:"
640 LPRINT:LPRINT:LPRINT
650 LPRINT SPACE$(25);"REORDERED OBJECTIVE FUNCTION":LPRINT
660 LPRINT SPACE$(27);"X";IC(1);
670 FOR I=2 TO N
680 PRINT #1,SPACE$(7);"X";:PRINT #1,USING "##";IC(I);
690 NEXT:LPRINT:LPRINT
700 LPRINT SPACE$(25);
710 FOR I=1 TO N
720 PRINT #1,USING "######.#  ";CNEW(I);
730 NEXT:LPRINT:LPRINT:LPRINT
740 LPRINT SPACE$(25);"REORDERED CONSTRAINTS":LPRINT
750 LPRINT SPACE$(15);"CONSTANT":LPRINT
760 FOR I=1 TO M
770 LPRINT SPACE$(10);"G";:LPRINT USING "##";I;
780 LPRINT USING "  ######.#";B(I);
790 FOR J=1 TO N
800 PRINT #1,USING "  ######.#";XA(I,J);
810 NEXT J:LPRINT:LPRINT
820 NEXT I
821 IJ=1
822 ITER.PRINTED%=0
830 LPRINT CHR$(12):LPRINT
840 LPRINT SPACE$(10);"ITER   NODE SELECTED FOR ACTIVE"
850 LPRINT SPACE$(10);" No.      PROCESSING (Xi)";
860 LPRINT SPACE$(31);"RESULTS"
870 PRINT #1,SPACE$(8);STRING$(85,223)
871 IF IJ=>2 THEN GOTO 965
879 V3=TIMER
900 FOR I=1 TO N
910 IX(I)=0   'first node processed is the root.
920 NEXT I
930 IXFEAS=0  'an interim solution has not been located.
940 GOTT=-1   'GOTT, or g*, is set at a low number.
942 IBACK=0   'if IBACK=1, the current node was reached by backtracking.
949 '
950 '
951 '         LINES 961 THROUGH 1120 - Printout of those nodes which are
952 '         explicitly enumerated.  A maximum of 30 nodes are printed
953 '         per page.
954 '
960 V4=TIMER
961 IF ITER.PRINTED%=30 THEN GOTO 822
965 ITER.PRINTED%=ITER.PRINTED%+1
969 LPRINT:LPRINT SPACE$(10);
970 LPRINT USING "### ";IJ;
980 IJ=IJ+1
990 FOR I=1 TO N
1000 II=N+1-I
1010 IF IX(II)=1 THEN GOTO 1030
1020 NEXT I
```

```
1021 LPRINT SPACE$(3);
1022 FOR J=1 TO N
1023 LPRINT ".";
1024 NEXT J
1025 GOTO 1110
1030 LPRINT SPACE$(3);
1040 FOR J=1 TO II
1050 LPRINT USING "#";IX(J);
1060 NEXT J
1070 II=II+1
1080 FOR J=II TO N
1090 LPRINT ".";
1100 NEXT J
1110 K=26-N
1120 LPRINT SPACE$(K);
1121 V5=TIMER
1122 PV1=PV1 + V5 - V4
1130 '
1131 '        ****** ONE COMPLETION TEST -- LINES 1140 THROUGH 1290 ******
1132 '        The one completion test is performed if a feasible solution
1133 '        exists (IXFEAS=1) and the current node was reached by back-
1134 '        tracking (IBACK=1).  If the test is passed, the program
1135 '        proceeds to the zero-completion/feasibility test beginning at
1136 '        line 1300. If the test is failed, the program proceeds to line
1136 '        2010 for further backtracking.
1138 '
1140 IF IXFEAS=0 THEN GOTO 1310
1141 IF IBACK=0 THEN GOTO 1310
1142 IBACK=0
1150 FOR I=1 TO N
1160 IX1COMP(I)=IX(I)
1170 NEXT I
1180 FOR I=1 TO N
1190 II=N+1-I
1200 IF IX1COMP(II)=1 THEN GOTO 1230
1210 IX1COMP(II)=1
1220 NEXT I
1230 VAL1 = 0
1240 FOR I=1 TO N
1250 VAL1 = VAL1 + (IX1COMP(I)*CNEW(I))
1260 NEXT I
1270 IF VAL1 > GOTT THEN GOTO 1310
1280 PRINT #1,"FAILS 1-COMPLETION";
1290 GOTO 2010
1300 '
1301 '        *********** ZERO COMPLETION TEST FOR FEASIBILITY ***********
1302 '        The node is feasible if all constraints , G(I), are less than
1303 '        zero.  If the node is feasible, it is compared to the current
1304 '        optimum solution beginning at line 1430.  If the node is
1305 '        infeasible, the program moves to line 1910.
1306 '
```

```
1310 FOR I=1 TO M
1320 G(I)=0
1330 FOR J=1 TO N
1340 G(I) = G(I) + (IX(J) * XA(I,J))
1350 NEXT J
1360 G(I) = G(I) + B(I)
1370 IF G(I) > 0 THEN GOTO 1400
1380 NEXT I
1390 GOTO 1420
1400 PRINT #1,"INFEASIBLE";
1410 GOTO 1910
1420 PRINT #1,"FEASIBLE";
1430 '
1431 '        ******** CHECK FEASIBLE NODE FOR IMPROVED SOLUTION ********
1432 '        Lines 1440 - 1500 compare the value of the current feasible
1433 '        node (GZERO) to the current interim optimum solution.
1434 '
1440 GZERO=0
1450 FOR I= 1 TO N
1460 GZERO =GZERO + (IX(I) * CNEW(I))
1470 NEXT I
1480 IF GZERO <= GOTT THEN GOTO 1570
1481 V6=TIMER
1490 PRINT #1," - INTERIM OPT. NODE - INT. SOLN.=";
1500 PRINT #1,USING "####.#";GZERO;
1501 V7=TIMER
1502 PV1=PV1+V7-V6
1510 '
1511 '        ******************** IMPROVED SOLUTION ********************
1512 '
1520 FOR I=1 TO N
1530 IXSTAR(I)=IX(I)  'X(I) becomes the interim  optimum solution.
1540 NEXT I
1550 GOTT = GZERO     'g0* = g0(x0)
1560 IXFEAS=1
1570 IF IX(N)=1 THEN GOTO 1700  'If node is leaf, goto backtrack.
1580 '
1581 '
1582 '          ******* FORWARD SEARCH - LINES 1590 THROUGH 1680 *******
1583 '          A one value is assigned to the first free variable of
1584 '          IX(II).  Processing of the new node begins at line 960.
1585 '
1590 FOR I=1 TO N
1600 II=N+1-I
1610 IF IX(II)=1 THEN GOTO 1640
1620 NEXT I
1630 GOTO 1670
1640 J=II+1
1650 IX(J)=1
1660 GOTO 960
1670 IX(1)=1
1680 GOTO 960
```

```
1690 '
1691 '         *** PROCEDURE FOR FEASIBLE LEAF & END OF SEARCH PRINTOUT ***
1692 '         If the current node is feasible and is the leftmost leaf on
1693 '         the tree, the search is ended.  If the leaf is not the left-
1694 '         most leaf , go to 2100 for backtracking. Lines 1731 - 1880
1695 '         are solution printout.  The problem solution is presented in
1696 '         the original order of input (X1,X2,...,Xn).
1697 '
1700 FOR I=1 TO (N-1)
1710 IF IX(I)=1 THEN GOTO 2100
1720 NEXT I
1730 V8=TIMER
1731 LPRINT CHR$(12)
1735 LPRINT SPACE$(10);"  -END OF SEARCH"
1740 LPRINT:LPRINT
1750 PRINT #1,SPACE$(10);"PROBLEM SOLUTION REACHED - AN OPTIMUM SOLUTION HAS BEE
N FOUND"
1760 PRINT #1,SPACE$(10);"THE SOLUTION GIVEN BELOW IS BASED ON THE ORIGINAL ORDE
R OF INPUT (X1,X2,...Xn)"
1770 LPRINT
1780 LPRINT SPACE$(5);"OPTIMAL SOLUTION";SPACE$(3);
1790 FOR I=1 TO N
1800 J=IC(I)
1810 IXPRINT(J)=IXSTAR(I)
1820 NEXT I
1830 FOR I=1 TO N
1840 LPRINT USING " #";IXPRINT(I);
1850 NEXT I
1860 LPRINT:LPRINT
1870 LPRINT SPACE$(5);"OPTIMUM VALUE OF OBJECTIVE FUNCTION= ";
1880 LPRINT USING "  #####.###";GOTT
1881 V9=TIMER
1882 PV1=PV1+V9-V8
1890 GOTO 2230
1900 '
1901 '        **************** INFEASIBLE NODE PROCESSING ****************
1902 '
1910 IF IX(N)=0 THEN GOTO 1590  'Sends nonleaf to forward search.
1920 FOR I=1 TO (N-1)
1930 IF IX(I)=1 THEN GOTO 2100  'Sends leaf, except leftmost, to backtracking.
1940 NEXT I
1950 IF IXFEAS=1 THEN GOTO 1730 'Sends leftmost leaf to feas. print.(X* exists).
1951 V10=TIMER
1951 V10=TIMER
1955 LPRINT CHR$(12)                'Lines 1955 -1980 are end of search print-
1960 LPRINT "  -END OF SEARCH"      'out for no existing feasible solution.
1970 LPRINT:LPRINT
1980 LPRINT "PROGRAM EXECUTION TERMINATED - NO FEASIBLE SOLUTION EXISTS"
1981 V11=TIMER
1982 PV1=PV1 + V11-V10
1990 GOTO 2230
```

```
2000 '
2001 '          ##### PROCEDURE IF NODE HAS FAILED ONE COMPLETION #####
2002 '          If IX(I) is of the form (X1,...,Xj,1,Xj+2,...,Xn) where
2003 '          X1 through Xj are zero and Xj+2 through Xn are not
2004 '          specified, further backtracking is not possible.  There-
2005 '          fore, proceed to 1730 for printout.  Otherwise, go to
2006 '          2100 for backtracking.
2007 '
2010 FOR I=1 TO N
2020 II=N+1-I
2030 IF IX(II)=1 THEN GOTO 2050
2040 NEXT I
2050 FOR I=1 TO (II-1)
2060 IF IX(I)=1 THEN GOTO 2100
2070 NEXT I
2080 GOTO 1730
2090 '
2091 '          ###################### BACKTRACKING ####################
2092 '          Moving from right to left, all variables, up to and
2093 '          including the second one valued variable, are freed.  The
2094 '          two left most free variables are given values of 0 1 .
2095 '          Backtracking covers lines 2110 - 2220.
2096 '
2100 FOR I=1 TO N
2110 II=N+1-I
2120 IF IX(II)=1 THEN GOTO 2140
2130 NEXT I
2140 IX(II)=0
2150 FOR I=1 TO N
2160 II=N+1-I
2170 IF IX(II)=1 THEN GOTO 2190
2180 NEXT I
2190 IX(II)=0
2200 J=II+1
2210 IX(J)=1
2211 IBACK=1
2220 GOTO 960
2230 V12=TIMER
2240 LPRINT:LPRINT
2250 PRINT #1,SPACE$(10);"A. total execution time excluding input printout (sec)
= ";(V12-V1)
2260 LPRINT:PRINT #1,SPACE$(10);"B. time required to reorder equations (sec)= ";
(V2-V1)
2270 LPRINT:PRINT #1,SPACE$(10);"C. time required to reprint equations (sec)= ";
(V3-V2)
2280 LPRINT:PRINT #1,SPACE$(10);"D. time required to print results (sec)= ";PV1
2290 LPRINT:PRINT #1,SPACE$(10);"E. real program execution time (A - C - D)= ";(
V12-V1-V3+V2-PV1)
4010 END
```

OBJECTIVE FUNCTION

X 1	X 2	X 3	X 4	X 5	X 6
2.0	6.0	2.0	4.0	3.0	6.0

CONSTRAINTS

	CONSTANT						
G 1	5.0	1.0	-2.0	-3.0	-6.0	1.0	2.0
G 2	4.0	-1.0	3.0	-2.0	-4.0	-2.0	4.0

FIGURE B-1

ONE-COMPLETION EXAMPLE PROBLEM

INPUT MATRIX

THE OBJECTIVE FUNCTION AND CONSTRAINT EQUATION VARIABLES HAVE BEEN REARRANGED IN ORDER TO SPEED PROCESSING. THE ACTUAL EQUATIONS USED FOR PROCESSSING APPEAR AS FOLLOWS:

REORDERED OBJECTIVE FUNCTION

X 2	X 6	X 4	X 5	X 1	X 3
6.0	6.0	4.0	3.0	2.0	2.0

REORDERED CONSTRAINTS

	CONSTANT						
G 1	5.0	-2.0	2.0	-6.0	1.0	1.0	-3.0
G 2	4.0	3.0	4.0	-4.0	-2.0	-1.0	-2.0

FIGURE B-2

ONE-COMPLETION EXAMPLE PROBLEM

REORDERED INPUT MATRIX

ITER No.	NODE SELECTED FOR ACTIVE PROCESSING (Xi)	RESULTS
1	......	INFEASIBLE
2	1.....	INFEASIBLE
3	11....	INFEASIBLE
4	111...	INFEASIBLE
5	1111..	INFEASIBLE
6	11111.	INFEASIBLE
7	111111	INFEASIBLE
8	111101	INFEASIBLE
9	11101.	INFEASIBLE
10	111011	INFEASIBLE
11	111001	INFEASIBLE
12	1101..	INFEASIBLE
13	11011.	INFEASIBLE
14	110111	INFEASIBLE
15	110101	INFEASIBLE
16	11001.	INFEASIBLE
17	110011	INFEASIBLE
18	110001	INFEASIBLE
19	101...	INFEASIBLE
20	1011..	INFEASIBLE
21	10111.	FEASIBLE - INTERIM OPT. NODE - INT. SOLN.= 15.0
22	101111	FEASIBLE - INTERIM OPT. NODE - INT. SOLN.= 17.0
23	101101	FAILS 1-COMPLETION
24	10101.	FAILS 1-COMPLETION
25	1001..	FAILS 1-COMPLETION
26	01....	FAILS 1-COMPLETION

FIGURE B-3

ONE-COMPLETION EXAMPLE

INTERMEDIATE RESULTS

```
-END OF SEARCH


PROBLEM SOLUTION REACHED - AN OPTIMUM SOLUTION HAS BEEN FOUND
THE SOLUTION GIVEN BELOW IS BASED ON THE ORIGINAL ORDER OF INPUT (X1,X2,...Xn)

     OPTIMAL SOLUTION    1 1 1 1 1 0

     OPTIMUM VALUE OF OBJECTIVE FUNCTION=        17.000


A. total execution time excluding input printout (sec)=  20.49024

B. time required to reorder equations (sec)=  .9902344

C. time required to reprint equations (sec)=  3.839844

D. time required to print results (sec)=  8.459472

E. real program execution time (A - C - D)=  8.190918
```

FIGURE B-4

ONE-COMPLETION EXAMPLE

FINAL RESULTS

ITER No.	NODE SELECTED FOR ACTIVE PROCESSING (Xi)	RESULTS
1	......	INFEASIBLE
2	1.....	INFEASIBLE
3	11....	INFEASIBLE
4	111...	INFEASIBLE
5	1111..	FEASIBLE - INTERIM OPT. NODE - INT. SOLN.= 14.0
6	11111.	FEASIBLE - INTERIM OPT. NODE - INT. SOLN.= 17.0
7	111111	INFEASIBLE
8	111101	INFEASIBLE
9	11101.	INFEASIBLE
10	111011	INFEASIBLE
11	111001	FAILS 1-COMPLETION
12	1101..	INFEASIBLE
13	11011.	FEASIBLE
14	110111	INFEASIBLE
15	110101	INFEASIBLE
16	11001.	FAILS 1-COMPLETION
17	101...	FAILS 1-COMPLETION
18	01....	INFEASIBLE
19	011...	INFEASIBLE
20	0111..	INFEASIBLE
21	01111.	FEASIBLE
22	011111	INFEASIBLE
23	011101	INFEASIBLE
24	01101.	FAILS 1-COMPLETION
25	0101..	INFEASIBLE
26	01011.	INFEASIBLE
27	010111	INFEASIBLE
28	010101	FAILS 1-COMPLETION
29	01001.	FAILS 1-COMPLETION
30	001...	FAILS 1-COMPLETION

FIGURE B-5

ONE-COMPLETION RESULTS

FOR

PROBLEM WITHOUT REORDERING

VITA

Dennis Don Brown

Candidate for the Degree of

Master of Business Administration

Report: A ONE-COMPLETION ENUMERATIVE METHOD FOR ZERO-ONE INTEGER PROGRAMMING

Major Field: Business Administration

Biographical:

Personal Data: Born in Amarillo, Texas, November 26, 1957, the son of Billy Don and Elizabeth Ann Brown.

Education: Graduated from Adrian High School, Adrian, Texas, May, 1976; received the Bachelor of Science degree from Texas A&M University with a major in Chemical Engineering, May 1980; completed requirements for the Master of Business Administration degree at Oklahoma State University, December, 1985.

Professional Experience: Petroleum Refinery Process Engineer, Conoco, Inc., 1980-1985.