

AN OBJECTIVE FUNCTION REDUCTION  
ALGORITHM FOR INTEGER  
LINEAR PROGRAMMING

By

JAMES MELVIN SHIRLEY

//

Bachelor of Science  
Oklahoma State University  
Stillwater, Oklahoma  
1963

Master of Science  
Oklahoma State University  
Stillwater, Oklahoma  
1968

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
DOCTOR OF PHILOSOPHY  
May, 1972

AUG 16 1973

AN OBJECTIVE FUNCTION REDUCTION  
ALGORITHM FOR INTEGER  
LINEAR PROGRAMMING

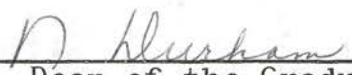
Thesis Approved:

  
Thesis Adviser







  
Dean of the Graduate College

## PREFACE

The principal goal of this research is to extend the existing theory of solution procedures for pure integer linear programming. This study is concerned with the development of a new algorithm for solving the pure integer linear programming problem. The procedure presented in this thesis uses combinatorial search methods to find the solution to the problem. A family of objective function hyperplanes is examined until an integer solution is found. Beginning at the optimum noninteger solution, the algorithm inspects parallel objective function hyperplanes in the feasible solution space.

Additional, secondary problems are considered in this research. These are (1) to identify any heuristic procedures that will speed the convergence of the algorithm, (2) to develop a procedure for finding a good approximate solution to the problem, and (3) to write a computer code to evaluate the algorithm. One stage of the algorithm proposes a ranking scheme for the variables to potentially eliminate many combinatorial solution possibilities from explicit consideration. A heuristic method of ranking is developed for a certain class of problems. This heuristic method allows the algorithm to take full advantage of

techniques that speed convergence. A technique is examined for finding an approximate solution to the pure integer linear programming problem. Also, this procedure can be used to establish a lower bound on an objective function that is to be maximized. A computer code is presented for further evaluation of the algorithm and any refinements or additions that may be considered.

The members of my doctoral advisory committee have given generously of their time and effort throughout my study and research. Dr. James E. Shamblin, the committee chairman and thesis adviser, continually offered the inspiration for new ideas. His perspective and understanding gave direction throughout the research and preparation of the thesis. Dr. Hamed K. Eldin guided in establishing research objectives. He provided an overview that added significantly to the continuity of this research. Dr. M. Palmer Terrell carefully reviewed this research as it developed. His perception added immeasurably to the accuracy and composition of this thesis. Dr. David L. Weeks gave direction and insight to my graduate study in statistics. His ability to find the central issue of any logic continually challenged me to reason clearly.

In particular, I gratefully acknowledge the contributions of Dr. James E. Shamblin during our years of work together. He provided the opportunity to do research in mathematical programming and teach in engineering. He gave the motivation to make this thesis possible.

I am thankful to Miss Velda Davis for her excellent typing. Her suggestions on appearance and form were invaluable.

I sincerely appreciate the work, patience, and understanding of my wife, Elaine, and son, Steven.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION . . . . .	1
Objectives . . . . .	3
General Concepts of the Objective	
Function Reduction Algorithm . . . . .	5
A Two-Variable Introductory Example . . . . .	9
Notation . . . . .	20
II. LITERATURE REVIEW OF INTEGER LINEAR PROGRAMMING . . . . .	22
General Classification of Algorithms . . . . .	23
Cutting Plane Algorithms . . . . .	25
Backtrack Algorithms . . . . .	30
Other Methods . . . . .	34
III. SOME CONCEPTS OF LINEAR DIOPHANTINE EQUATIONS. .	37
The Greatest Common Divisor . . . . .	39
Euclid's Algorithm . . . . .	44
Linear Diophantine Equations . . . . .	47
IV. THE OBJECTIVE FUNCTION REDUCTION ALGORITHM . . .	52
Stage One . . . . .	55
Stage Two . . . . .	58
Stage Three . . . . .	63
Stage Four. . . . .	67
A Three Variable Example . . . . .	74
V. SOME IMPLICATIONS OF THE ALGORITHM . . . . .	90
The Computer Code . . . . .	90
The Scheme of Ranking the Variables . . . . .	93
Modifying the Coefficients in the Objective Function . . . . .	97
VI. SUMMARY AND CONCLUSIONS . . . . .	103
Important Findings . . . . .	103
Areas for Further Investigations . . . . .	106
Conclusions . . . . .	107

Chapter	Page
SELECTED BIBLIOGRAPHY . . . . .	109
APPENDIX A - FLOW OF LOGIC IN THE ALGORITHM . . . . .	112
APPENDIX B - MAIN TEST PROGRAM . . . . .	124
APPENDIX C - SUBROUTINE GCD . . . . .	130
APPENDIX D - SUBROUTINE RANK . . . . .	133
APPENDIX E - SUBROUTINE SEARCH . . . . .	136
APPENDIX F - TEST PROBLEMS AND SOLUTIONS . . . . .	146

## LIST OF TABLES

Table	Page
I. An Example With the Variables Ranked According to Range Size . . . . .	95

## LIST OF FIGURES

Figure	Page
1. Two Variable Introductory Example Problem . . . . .	10
2. Two Variable Introductory Example Problem With Ranked Variables . . . . .	15
3. Family of Objective Functions in the Solution Space . . . . .	19
4. Stage 1 Logic Flow Diagram . . . . .	57
5. Stage 2 Logic Flow Diagram . . . . .	62
6. Stage 3 Logic Flow Diagram . . . . .	66
7. Stage 4 Logic Flow Diagram . . . . .	72



## NOMENCLATURE

$a_{ij}$	a coefficient in the $i^{\text{th}}$ constraint for the original problem variable, $x_j$ .
$\hat{a}_{ij}$	a coefficient in the $i^{\text{th}}$ constraint for the ranked variable, $y_j$ .
$b_i$	the right-hand side value of the $i^{\text{th}}$ constraint.
$b_i^t$	the temporary right-hand side value of the $i^{\text{th}}$ constraint.
$c_j$	an objective function coefficient for variable $x_j$ . It must be integer.
$\hat{c}_j$	an objective function coefficient for variable $y_j$ . It must be integer.
$g$	the greatest common divisor of the coefficients in the objective function.
$i$	index for number of functional constraints.
IFLAG	a key word indicating all successive subscripts in the ranking $\geq$ IFLAG have even coefficients.
$j$	index for number of variables.
$l$	a superscript indicating a lower bound.
$m$	the number of functional constraints in the problem. That is, the nonnegativity constraints are not counted in $m$ .
MAX	a key word indicating a maximization problem when it

equals one, indicating a minimization problem when it equals zero.

$n$  the number of problem variables, not including the slack or artificial variables.

$r(x)_j$  the range of a variable  $x_j$  in the original problem.

$r(y)_j$  the range of a variable  $y_j$  in the ordered ranking of the variables.

$x^*$  an optimum continuous value for a variable.

$\underline{x}^*$  an optimum integer solution vector.

$\underline{x}$  a solution vector.

$x_j$  a problem variable.

$x_j^l$  a lower integer bound on variable  $x_j$ .

$x_j^u$  an upper integer bound on variable  $x_j$ .

$y^*$  an optimum integer value for a ranked variable.

$\underline{y}^*$  an optimum integer solution vector in terms of ranked variables.

$\underline{y}$  a solution vector in terms of ranked variables.

$y_j$  a ranked variable.

$y_j^l$  the lower integer bound on ranked variable  $y_j$ .

$y_j^u$  the upper integer bound on ranked variable  $y_j$ .

$y(j)_t^l$  the temporary lower bound on the  $j^{\text{th}}$  ranked variable.

$y(j)_t^u$  the temporary upper bound on the  $j^{\text{th}}$  ranked variable.

$z$  a value of the objective function.

$z^*$  the value of the objective function at the optimum integer solution.

$z_t$  a temporary value of the objective function.

$ZOF$  the value of the objective function that is being

searched for a solution.

ZSIM     the value of the objective function for the  
         simplex linear programming solution.

ZSUBT   a temporary value of the objective function.

## CHAPTER I

### INTRODUCTION

The subject of pure integer linear programming is approximately fifteen years old. Even though a great amount of successful work has been done when the problem variables can take on continuous values, the area of integer programming is still difficult in practice. Many early algorithms suggested cutting-plane methods that added new constraints at each iteration. Later, branch-and-bound techniques were developed to solve the integer programming problem. Enumeration schemes and heuristic techniques have been examined as possible solution procedures. While most algorithms offer convergence in a finite number of steps, in practice finite can often be very large.

Many situations of both industrial and theoretical importance can be formulated as an integer linear programming problem. Problems involving equipment utilization, labor allocation, capital budgeting and others require that the variables can take on only integer values. Therefore, integer linear programming has offered the promise of solving several operations research models. The algorithms available to date have not always been able to fulfill the promised solution. Some algorithms are only useful on

certain limited problems. In any of the algorithms, certain examples can be devised that require approximately an equivalent amount of effort as complete enumeration of the feasible solutions.

So, the paradox exists. Integer linear programming has offered the solution to many problems of operations research, while the available algorithms have provided only limited practical success. The research of this thesis explores a new, potentially useful method of extending the analysis of integer linear programming. This algorithm can provide a pure integer programming solution or a good approximate solution with a lower bound on the maximand. Even though the exact solution is important, an approximate solution greatly increases the efficiency of some optimal algorithms. Hillier (14) indicates the great importance of good approximate solutions in integer linear programming problems.

The difficulties associated with solving the integer linear programming problem can be enormous. Constrained optimization problems often imply a finite solution space exists. Nevertheless, the potential combinatorial possibilities can be great even in problems with a moderate number of variables and constraints. It is no wonder that some early thinkers concluded the problem was impossible to solve.

Complete enumeration quickly becomes impractical, since each combinatorial possibility within the solution space must be tested for its feasibility and must have the

objective function value tested. Because of the efficiency of the simplex method, a common procedure in practice is to solve the problem for the optimal noninteger solution and then use a rounding procedure to obtain an approximate solution. On certain problems, this technique can lead to solutions far from the optimum integer solution. Wagner (30) and Hillier and Lieberman (16) present examples of how rounding can sometimes lead to poor solutions. The possibility of rounding the optimal noninteger solution to simply a feasible solution can be a difficult problem when dealing with several constraints in a multispace system.

The method of searching the feasible integer solution space must be intelligently structured or certain problems could not be solved in several lifetimes. For example, if a problem contained only 25 variables and each could take on only two values, then  $2^{25} = 33,554,432$  possible combinations exist. Any practical algorithm must take advantage of techniques to avoid complete enumeration and examination. The algorithm developed in this research takes advantage of several methods that implicitly examine and eliminate a large number of possible solutions.

### Objectives

The primary goal of this research is to extend the existing theory of solution procedures for pure integer linear programming. In particular, a new algorithm for solving the pure integer linear programming problem is

developed. Additionally, other objectives are (1) to identify any heuristic procedures that will speed the convergence of the algorithm, (2) develop a procedure for finding a good approximate solution to the problem, and (3) write a computer code to evaluate the algorithm. To meet these objectives, this thesis presents an algorithm that uses combinatorial search methods to find the solution to the pure integer linear programming problem.

Heuristic procedures can point the way to obtaining more insight into the structure of a solution technique. Also, they can often be used to speed convergence of an algorithm. Since computational efficiency is a prime consideration in integer programming, heuristic procedures can frequently be used to move the algorithm quickly to the solution.

Many integer linear programming algorithms can benefit from a good approximate solution. Hillier (14, 15) describes the importance of finding methods that will provide near optimum solutions. Oftentimes, large problems with a great many variables and constraints can only be economically solved with approximate procedures. Therefore, a method for finding a good approximate solution of the integer linear programming problem will be investigated in this research.

To be able to evaluate the solution procedure described in this thesis a computer code is required. Problems of practical and theoretical importance often involve several

variables and constraints. A computer code is essential to solve large problems. Also, it provides a useful method for evaluating modifications and variations in solution procedures. A brief summary of the main concepts of the algorithm developed in this research is given in the following section.

### General Concepts of the Objective Function Reduction Algorithm

The objective function reduction algorithm seeks a solution to the pure integer linear programming problem. The problem can be expressed in canonical form as follows:

$$\text{maximize} \quad z = \sum_{j=1}^n c_j x_j \quad (1-1)$$

$$\text{subject to} \quad \sum_{j=1}^n a_{ij} x_j \leq b_i \quad \text{for } i = 1, 2, \dots, m \quad (1-2)$$

$$x_j \geq 0 \quad \text{for } j = 1, 2, \dots, n \quad (1-3)$$

$$x_j, c_j \text{ INTEGER} \quad \text{for } j = 1, 2, \dots, n. \quad (1-4)$$

It is assumed that the set of constraints of Equations (1-2) and (1-3) bound the solution space. The nonnegativity requirement of Equation (1-3) is not necessary for the conceptual approach developed in this research. Nevertheless, this is a typical and often a necessary condition in a practical problem. The algorithm restricts the objective



function coefficients to integers, while the coefficients in the constraints and their right-hand side values may be non-integer. Constraint coefficients and objective function coefficients are not restricted as to sign and may be negative or positive.

Very generally, the objective function reduction algorithm moves through a family of parallel objective function planes within the solution space, away from the optimal noninteger solution, until a feasible integer solution is found, which is the optimal integer solution. The basic flow of the logic of the algorithm can be described in four stages. Stage 1 requires that the simplex method be used to find the optimal continuous-variable solution. If this solution is all-integer, the algorithm stops. Otherwise, the algorithm requires knowledge of the value of the objective function at the optimal noninteger solution. The next part of Stage 1 is to identify the bounds on each problem variable, as defined by the functional and nonnegativity constraints. If  $n$  is the number of problem variables, then this can be done by solving  $2n$  linear programming problems. These linear programming problems are subject to the functional constraints and have objective functions of the form

$$\text{maximize} \quad z = x_j \quad (1-5)$$

and,

$$\text{minimize} \quad z = x_j \quad (1-6)$$

for each  $j$ ,  $j = 1, 2, \dots, n$ . In practice these bounds can

be found reasonably fast. They may be readily known from experience of working with the problem. Minimization problems with only greater-than-or-equal-to constraints must have a finite upper bound defined for each variable. These bounds become integer bounds when the quantities found in Equation (1-5) are selected so the upper bound on the variable is the greatest integer less than or equal to  $x_j$ . Similarly, the lower integer bound on each variable is selected such that it is the least integer greater than or equal to  $x_j$  found in Equation (1-6).

Stage 2 examines the coefficients of the objective function. Using concepts from the study of linear Diophantine equations and the theory of numbers, the greatest common divisor of the objective function coefficients is found. Assuming a maximization problem, the value of the objective function as found with the simplex method in Stage 1 is rounded down to the greatest integer that has the greatest common divisor as a factor.

Stage 3 determines how the variables should be ranked so the search of Stage 4 will implicitly examine and exclude several solutions. Also, Stage 3 identifies any successive sequence of objective function coefficients in the ranking such that all succeeding values are even integers. This sequence of even coefficients is used to take advantage of additional concepts of linear Diophantine equations.

The final stage, Stage 4, is the heart of the implicit search of the feasible integer solution space. Again

assuming a maximization problem, the first variable in the ranking of Stage 3 is set at its upper bound. The objective function is set equal to the integer value determined in Stage 3, giving an upper bounding hyperplane on the maximand. Using the objective function and the functional constraints, new, potentially tighter bounds are found on the next variable in the ranking. This next variable in the ranking is then set at its new upper bound. This process of finding tighter upper and lower bounds continues until the algorithm finds it can take advantage of some concepts of linear Diophantine or the next to last variable in the ranking is reached. Using the objective function and the held value of the variables, the value of the final variable in the ranking is calculated. If this solution is integer, its feasibility is tested in the functional constraints. If the final variable is noninteger or an all-integer solution is found infeasible, then the algorithm backtracks through the ranking until it has explicitly or implicitly examined each combinatorial possibility for a particular objective function value. The first feasible integer solution found is the optimum integer solution for the problem. When all solutions have been considered for the first objective function value, and no feasible solution has been found, then the objective function is incremented down one greatest common divisor increment. The algorithm returns to the beginning of Stage 4 to examine the possibility of a feasible integer solution on a new, reduced hyperplane.

Some of the essential concepts of the objective function reduction algorithm can be introduced with a simple two-variable example. The following section describes such an example.

#### A Two-Variable Introductory Example

One advantage of a two-variable example is the solution space can be easily visualized. The simplicity of finding the solution in two dimensions often indicates a particular search technique will be very valuable. When several variables are considered, that is, the problem is multidimensional, the satisfactory techniques of two dimensions often become tedious and ineffectual. Nevertheless, because of the visual properties of a two-dimensional example, it is useful to fall back on to point out some of the features of an established algorithm, such as the one developed in this research.

Consider the following two-variable example:

$$\text{maximize} \quad z = 8 x_1 + 10 x_2 \quad (1-7)$$

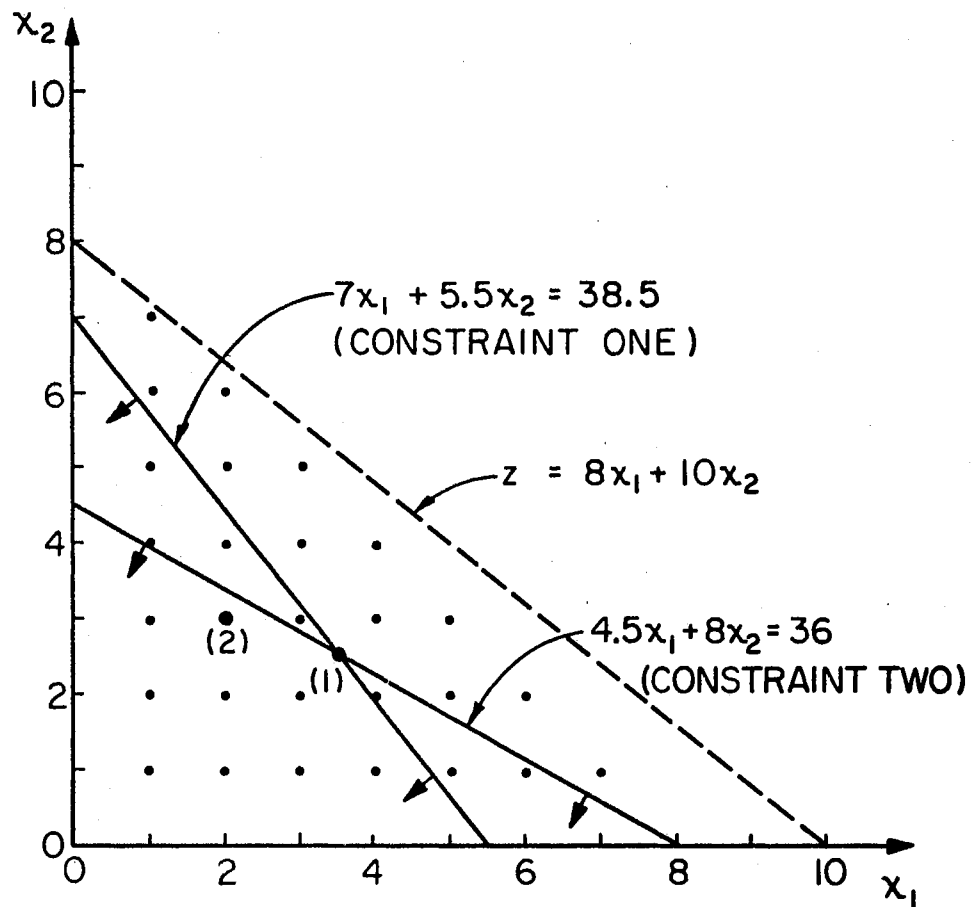
$$\text{subject to} \quad 7 x_1 + 5.5 x_2 \leq 38.5 \quad (1-8)$$

$$4.5 x_1 + 8 x_2 \leq 36.0 \quad (1-9)$$

$$x_j \geq 0 \quad \text{for } j = 1, 2 \quad (1-10)$$

$$x_j, c_j \text{ INTEGER for } j = 1, 2. \quad (1-11)$$

Figure 1 shows how the constraints bound the solution area.



NOTES:

(1) OPTIMUM NONINTEGER SOLUTION,  $\underline{x} = (3.52, 2.52)$

(2) OPTIMUM INTEGER SOLUTION,  $\underline{x}^* = (2, 3)$

Figure 1. Two Variable Introductory Example Problem

The Stage 1 analysis of the algorithm determines first the optimum noninteger solution of the problem. As shown in Figure 1, the optimum noninteger solution is  $\underline{x} = (x_1, x_2) = (3.52, 2.52)$ . Since this solution is not all-integer, the algorithm continues its search for a feasible integer solution. The value of the objective function, Equation (1-7), at the optimum noninteger solution is  $z = 53.36$ . By inspecting the constraint boundaries of Figure 1, the maximum over-all integer bounds of the problem variables can be found. Therefore, the upper and lower integer bounds can be defined by the inequalities

$$0 \leq x_1 \leq 5 \quad (1-12)$$

and

$$0 \leq x_2 \leq 4. \quad (1-13)$$

Stage 1 says the objective function at the optimal non-integer solution is

$$z = 8 x_1 + 10 x_2 = 53.36. \quad (1-14)$$

This would be a straight line parallel to the objective function equation shown in Figure 1, intersecting the optimum noninteger solution at  $\underline{x} = (x_1, x_2) = (3.52, 2.52)$ . Since this solution is not integer optimum, the optimum solution must lie on some parallel objective function line below the maximum described in Equation (1-14). Also, notice that the  $c_j$  coefficients are restricted to integer values. Therefore, the left-hand side of the objective

function must equal some integer value, since the product and sum of integers must be an integer value. To force the right-hand side of the objective function to an integer value, it seems justifiable to round Equation (1-14) down to the greatest integer below 53.36 to get the equation

$$z = 8 x_1 + 10 x_2 = 53. \quad (1-15)$$

Using the methods that will be explained in Chapter III, Stage 2 calculates the greatest common divisor of the coefficients ( $c_j$ ) in the objective function, Equation (1-7). For this simple introductory example problem, the greatest common divisor can be determined by inspecting the coefficients  $c_1 = 8$  and  $c_2 = 10$  in Equation (1-7). It can be seen that the greatest integer that will divide evenly into 8 and 10 is the number 2. Therefore, the greatest common divisor for the objective function is 2.

As will be shown in Chapter III with the study of linear Diophantine equations, no integer solution is possible for Equation (1-15). The optimum noninteger solution value must be rounded down until it has the greatest common divisor as a factor. That is, begin the recursive search for a feasible integer solution with the first reduced objective function being

$$z = 8 x_1 + 10 x_2 = 52. \quad (1-16)$$

The greatest common divisor, 2, divides evenly into the integer value  $z = 52$ . This says there is at least one

integer solution to this objective function. If this integer solution is within the solution area, as defined by the functional and nonnegativity constraints, the optimum solution has been found. Otherwise, the optimum solution must be located on a further reduced objective function parallel to Equation (1-16). Stage 2 is complete when  $z = 52$  of Equation (1-16) is found.

In Stage 3, the variables are ranked or ordered according to the number of feasible integer possibilities each variable can take on. Equations (1-12) and (1-13) describe the lower and upper bounds on each variable. Variable  $x_1$  can take on six integer values, while variable  $x_2$  can take on five integer values. As described later in Chapter IV, the ranking scheme is to assign the variable with the tightest bound the highest ranking position. A new symbol,  $y_1$ , will be used to indicate a ranked variable. Therefore, the direct change of variables will give

$$y_1 = x_2 \quad (1-17)$$

and,

$$y_2 = x_1. \quad (1-18)$$

Therefore, the bounds on the ranked variable are

$$0 \leq y_1 \leq 4 \quad (1-19)$$

and,

$$0 \leq y_2 \leq 5. \quad (1-20)$$



Based on the change of variables given in Equations (1-17) and (1-18) a new problem described in terms of the ranked variables ( $y_j$ ) is defined. This new problem will be solved using the objective function reduction algorithm. To avoid notation difficulties, it is often convenient to think completely in terms of the ranked variables and the new problem. Later, the variables can be returned to their original form. The new problem, written with ranked variables, is

$$\text{maximize} \quad z = 10 y_1 + 8 y_2 \quad (1-21)$$

$$\text{subject to} \quad 5.5 y_1 + 7 y_2 \leq 38.5 \quad (1-22)$$

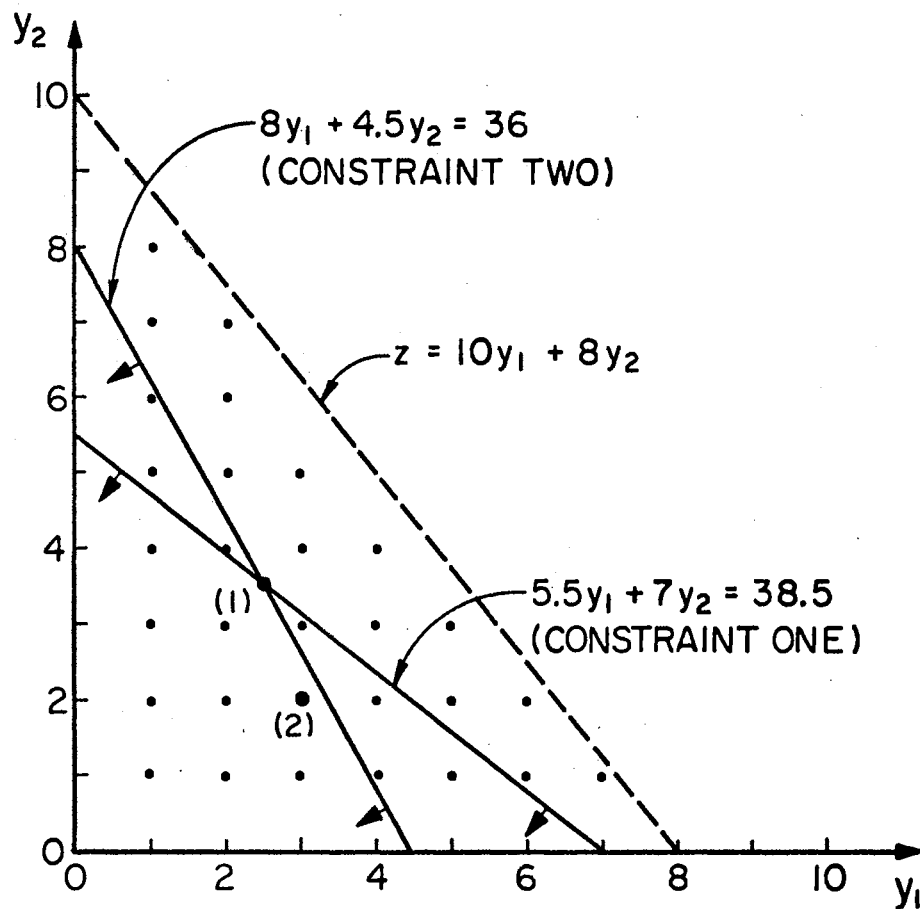
$$8 y_1 + 4.5 y_2 \leq 36.0 \quad (1-23)$$

$$y_j \geq 0 \quad \text{for } j = 1, 2 \quad (1-24)$$

$$y_j, \hat{c}_j \text{ INTEGER for } j = 1, 2. \quad (1-25)$$

Figure 2 shows how the ranked constraints bound the solution area.

An additional requirement of Stage 3 is to identify any successive sequence in the ranking such that all succeeding coefficient values in the objective function are even integers. Equation (1-21) shows that both coefficients in the objective function are even integers. The algorithm records and potentially uses this fact. From the theory of numbers and linear Diophantine equations, the algorithm uses a rather simple observation. If during any part of the



## NOTES:

(1) OPTIMUM NONINTEGER SOLUTION,  $\underline{y} = (2.52, 3.52)$

(2) OPTIMUM INTEGER SOLUTION,  $\underline{y}^* = (3, 2)$

Figure 2. Two Variable Introductory Example  
Problem With Ranked Variables

search, the objective function is given a value that is an odd integer, while all coefficients are even, then no integer solution can exist for that equation. Consequently, several solutions can be implicitly examined and eliminated.

As stated before, Stage 4 is the heart of the objective function reduction algorithm. Using the ranking scheme and the bounds found earlier, the first variable in the ranking is set at its upper bound. Therefore, set

$$y_1 = 4. \quad (1-26)$$

From Stage 2, the first reduced objective function value to be examined is

$$z = 10 y_1 + 8 y_2 = 52. \quad (1-27)$$

In a problem involving several variables, the held value of  $y_1$  ( $y_1 = 4$ ) would be substituted into the reduced objective function and the functional constraints. Obviously, with one variable held at a fixed value, the right-hand side of the constraints and the reduced objective function value can be modified. As Figure 2 shows, when the variable  $y_1$  is held fixed at  $y_1 = 4$ , the constraints will provide tighter bounds on the next variable the ranking,  $y_2$ . With  $y_1 = 4$ , the functional constraints of Equations (1-22) and (1-23) indicate tighter bounds on  $y_2$ . That is,

$$5.5(4) + 7 y_2 \leq 38.5 \quad (1-28)$$

$$8(4) + 4.5 y_2 \leq 36.0 \quad (1-29)$$

and this leads to

$$y_2 \leq 2.35 \quad (1-30)$$

from constraint one, and

$$y_2 \leq 0.88 \quad (1-31)$$

from constraint two. This is shown clearly in Figure 2.

The minimum value from Equations (1-30) and (1-31) provides a new, temporary upper bound on  $y_2$ . This means that  $y_2$  must be no greater than the greatest integer less than or equal to  $y_2 \leq 0.88$  from Equation (1-31). Therefore,

$$y_2 \leq 0 \quad (1-32)$$

is a new upper bound on the next variable in the ranking.

This procedure has reduced the number of possible values to examine.

With only two variables, the process of finding tighter bounds is unnecessary. For a fixed value of one variable, the reduced objective function of Equation (1-27) provides the value of the other variable. Nevertheless, the two-variable example easily shows the principle of finding tighter bounds. In the complete algorithm described in Chapter IV, the reduced objective function equation is used similarly as a constraint would be used to aid in identifying the tightest possible bound on the next variable in the ranking.

With two variables, Equation (1-27) leads quickly to

the solution, for a fixed value of  $y_1 = 4$ . That is,

$$10(4) + 8 y_2 = 52 \quad (1-33)$$

and

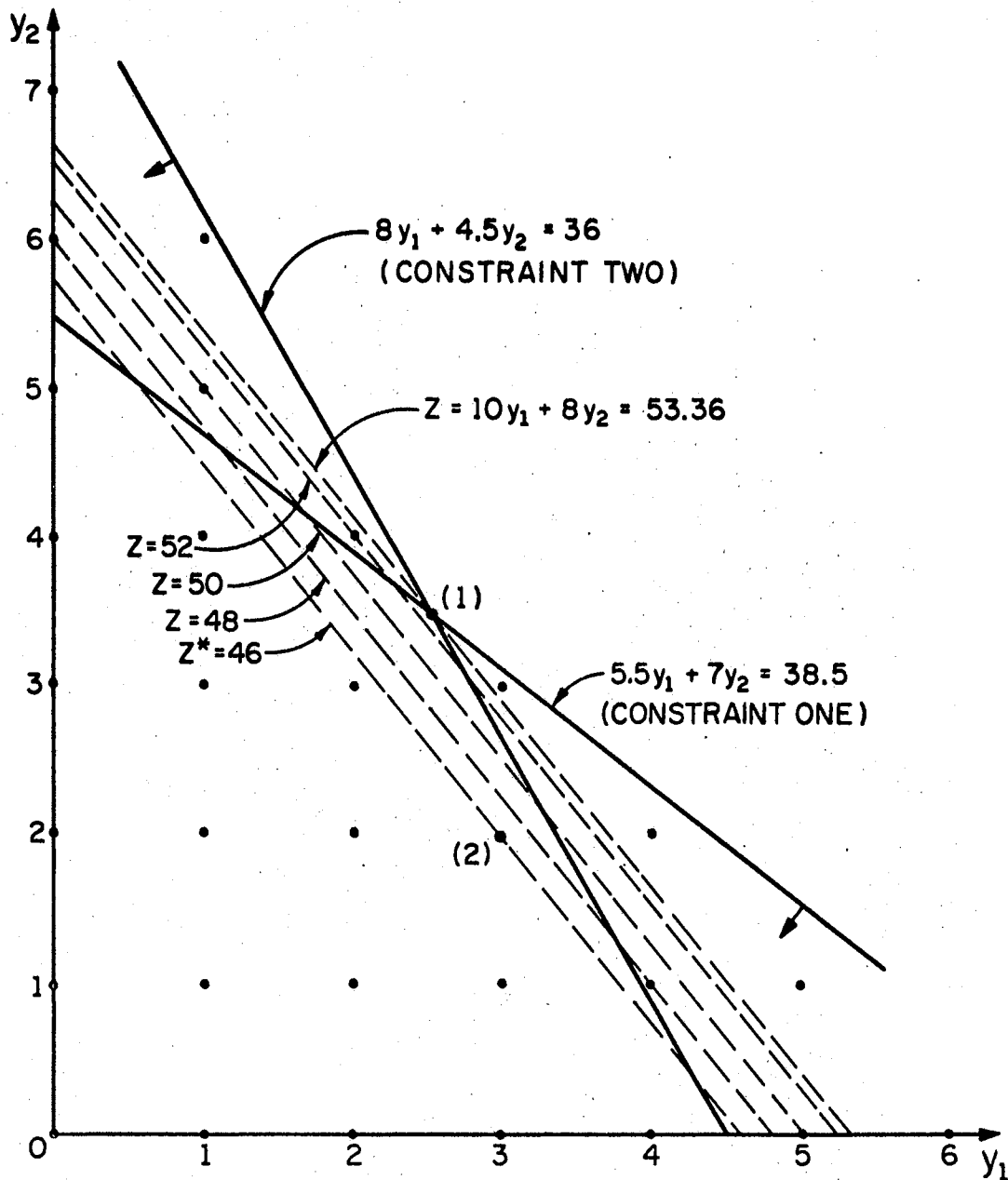
$$y_2 = 1.5. \quad (1-34)$$

Since  $y_2$  has been found to be a noninteger value, it is immediately eliminated as a feasible solution. If  $y_2$  had been found integer, the feasibility of the solution  $\underline{y} = (y_1, y_2)$  would be tested in the functional constraints. The first feasible, all-integer solution found is the optimum solution.

For this two-variable example, the algorithm says no feasible, integer solution exists for  $z = 52$  and  $y_1 = 4$ . The search continues by incrementing  $y_1$  down one integer and solving for  $y_2$ . This process continues until all integer possibilities for  $y_1$  have been tested, which implies all values of  $y_2$  have been tested. If the optimum solution is not found at  $z = 52$ , then the objective function is still further reduced by the amount of the greatest common divisor to  $z = 50$ . This gives a new objective function equation to be searched, which is

$$z = 10 y_1 + 8 y_2 = 50. \quad (1-35)$$

Figure 3 shows this process will continue until the optimum solution is found at  $\underline{y}^* = (3, 2)$  with  $z = 46$ . Notice how Figure 3 indicates that the first feasible integer solution that is found, as the objective function moves



NOTES:

(1) OPTIMUM NONINTEGER SOLUTION,  $y = (2.52, 3.52)$

(2) OPTIMUM INTEGER SOLUTION,  $y^* = (3, 2)$

Figure 3. Family of Objective Functions in the Solution Space

down through the solution space, is the optimum solution.

The limitation of only dealing with two variables did not allow all of the objective function reduction algorithm to be demonstrated. It does provide a visual reference for many concepts that will be extended and examined in more detail later.

Some of the notation used in this research will now be described.

### Notation

One part of the algorithm ranks the variables according to a ranking scheme. A ranked variable will be indicated by the letter  $y$ . The ranked variables do not necessarily have the same subscript number as the problem variable they represent. The subscripts on the  $y$ -variables indicate ranking order. Therefore,  $y_2$ , the second ranked variable, could be the ranked variable identification for problem variable  $x_8$ .

It is sometimes convenient to represent a solution as an  $n$ -vector of the form

$$\underline{y} = (y_1, y_2, \dots, y_n). \quad (1-36)$$

To indicate the optimal integer solution, the  $n$ -vector

$$\underline{y}^* = (y_1^*, y_2^*, \dots, y_n^*) \quad (1-37)$$

will be used.

During certain parts of the algorithm, it is necessary to round down or round up to the nearest integer. The

common notation that is used for this is

$$[y] = \text{the greatest integer } \leq y \quad (1-38)$$

and

$$\langle y \rangle = \text{the least integer } \geq y. \quad (1-39)$$

Upper and lower bounds often need to be identified as the algorithm is described. An upper integer bound on a variable will be indicated with the superscript  $u$  and a lower integer bound will be indicated by the superscript  $\ell$ . Some typical examples might be

$$y_3^{\ell} = 5 \quad (1-40)$$

and

$$y_3^u = 7. \quad (1-41)$$

The complete notation required for this thesis is listed in the Nomenclature section preceding this introductory chapter. Before looking more closely at the details of the objective function reduction algorithm, a brief review of the literature and some concepts from the study of linear Diophantine equations will be presented.



## CHAPTER II

### LITERATURE REVIEW OF INTEGER LINEAR PROGRAMMING

During the past ten years, there has been a great amount of research and publications on integer linear programming. This literature review will identify some of the more recent articles that are widely referenced and are typical of the work being done using a particular approach to the problem. In 1965, two surveys appeared that examined many of the important algorithms up to that year. Balinski (3) summarizes the major methods that have been successful or interesting in their method of approaching the problem. Included are some descriptions of general algorithms and computational experience dealing with integer linear programming. Beale (4) presents a survey of linear programming problems where some or all of the variables are required to take on integer values. Four separate methods of solving integer linear programming problems are reviewed and discussed. A survey of the literature of the late 1960's would make an excellent contribution to the literature of integer programming. This gap in the literature should certainly be filled in the coming months.

## General Classification of Algorithms

Several methods of classifying the algorithms of integer linear programming have been used. Any general method of classification will be incomplete because of the variety of methods proposed to solve discrete programming problems. Nevertheless, the two areas of classification suggested by Wagner (30) are appropriate for the literature reviewed in this thesis. The two main approaches for finding optimal solutions to integer programming problems are the cutting-plane algorithms and the backtrack algorithms.

The cutting-plane algorithms appear in several forms. They can be used to solve both the mixed integer programming problems and pure integer programming problems. The cutting-plane algorithms start at the optimum linear programming solution and then move toward the optimum integer linear programming solution. The early work of Ralph E. Gomory identified the significant contribution that this approach could make to solving integer linear programming problems. Generally, these methods assume the optimum linear programming solution has been found and is not integer. Additional cuts or constraints are then added to the original constraints. These new constraints are added in such a way that they reduce the feasible solution space, but they do not exclude any possible integer solutions. The algorithm is completed when a feasible integer programming solution has been found. These methods have been shown to be finite converging algorithms. With the addition of new

constraints at each iteration, the number of iterations in the finite convergence of even a moderate problem can be quite large.

The second classification group is somewhat broad and includes many approaches to the problem. Again, the backtrack algorithms can be used to solve both mixed and integer linear programming problems. Under this category are the branch-and-bound algorithms, implicit enumeration algorithms, shifted functional hyperplane methods, and many others. As in the cutting-plane algorithms, the backtrack algorithms begin at the optimal linear programming solution. These techniques then create a group of related linear programming algorithms. For example, in the branch-and-bound algorithms, a series of subproblems and a lower bound (for minimization) are determined. Similar to the concepts of dynamic programming, at each stage of subdivision, certain solutions are excluded as infeasible and are not examined. The name backtrack algorithms is given to these methods because they start at the optimal noninteger solution and back away from it, searching a sequence of generated problems for the optimal integer linear programming solution.

Naturally, all of the approaches to the integer programming problems cannot be classified with these two principal methods. The heuristic programming techniques or the statistical methods do not readily fall within either of the two categories. Some papers that are difficult to classify will be discussed at the end of the chapter.

## Cutting Plane Algorithms

The name most often mentioned when discussing cutting-plane algorithms is that of Ralph E. Gomory. Until some of his early work was presented in 1958, a general method for solving integer linear programming problems was assumed to be impossible by many people. Since that time, Gomory and several others have continued to explore the possibilities of cutting-plane algorithms.

Of the surveys made during the mid 1960's, the one by E. M. L. Beale (4) is most readable. The theory of Gomory's methods are explained quite well by Beale. In the early cutting-plane algorithms for pure integer programming, the method begins by finding the optimal noninteger solution to the linear programming problem. If the solution to this problem, where the variables can take on continuous values, happens to turn out to be all-integer, the algorithm stops. If some or all of the variables in the solution are noninteger, a new constraint is added to the problem. This new constraint eliminates a part of the feasible solution space near the optimum noninteger solution. It eliminates the optimal noninteger solution and other solutions near the optimum, but it does not eliminate any feasible integer solutions. Then, the simplex tableau is manipulated using the dual simplex method to move away from the optimal noninteger solution. If the dual simplex iteration finds an optimum integer solution, the algorithm is complete. If not, a new constraint is added and the process continues until

the optimum integer solution is found. The new constraints are often called cuts or cutting-plane constraints. They get their name from the way they cut away some of the feasible solution space. Their purpose is to force an integral solution.

After the work described above, Gomory (10) presented some new, important modifications to his all-integer integer programming algorithm. It differs from the earlier work in two main ways:

1. The technique is all-integer. The coefficients in the original matrix are integers and all coefficients remain integer during the whole calculation.
2. It is a uniform procedure similar to the dual simplex method. Also, the cycle of adding an inequality has been eliminated.

This method does not begin at the optimal solution determined by letting all the variables have a continuous range. It begins by making the problem dual feasible. This is done by adding an artificial constraint that the sum of the non-basic variables be less than or equal to some arbitrarily large number. From this point on each succeeding pivotal row is a new cut and is generated in such a way that it makes the pivot equal to minus one. This causes the integral tableau to remain integral.

This contribution of Gomory (10) is widely referenced in the literature on cutting-plane algorithms of integer

linear programming. This work apparently influenced much of the later research.

Economic applications of some of the initial work in integer programming began appearing in the literature in the early 1960's. Gomory and Baumol (11) discussed the topic of integer programming and pricing in a paper in 1960. The article describes and gives an example of integer linear programming. The majority of the paper discusses economic considerations such as prices, marginal yields of scarce indivisible resources, and efficient allocation of resources.

The work of Fred Glover has made a significant contribution to the study of integer programming. In 1967, Glover (8) presented a paper describing a primal integer programming algorithm. The technique is described by Glover as a new foundation for a simplified primal integer programming algorithm. The main focus of this research starts by considering the ordinary linear programming problem. Then, the same problem is considered again where the solution is required to be in pure integer form. Because the simplex technique is so effective for solving the ordinary linear programming problem, Glover sought an adoption of the simplex algorithm to solve the pure integer programming problem. The goal is to maintain a primal feasible and integer solution at each iteration. The author says such an adoption is straightforward and points to his earlier work and that of Richard D. Young (31). This adoption, called the rudimentary primal algorithm, draws on the early

concepts of Gomory where cuts are added to the feasible solution.

Glover's (8) simplified primal algorithm begins with the rudimentary primal algorithm and provides some rules that lead to a convergent algorithm. The author provides theorems and proofs to describe the success of this method. Even though Glover describes the rules as simple and the theorems as elementary, the analysis is still somewhat difficult to follow. Nevertheless, the fact that a primal integer programming algorithm has been found is a significant contribution to the literature.

A companion paper to Fred Glover's (8) article is one by Richard D. Young (31). Glover describes Young's work as a pioneering paper that produced a finite primal algorithm. He goes on to call it an outstanding, original contribution to integer programming. Young's algorithm is a complicated and difficult technique for primal integer programming.

Richard D. Young's (31) paper describes a primal, all-integer algorithm for solving a bounded and solvable pure integer programming problem. This algorithm is a primal analogy to some of Gomory's early work with cutting plane techniques. The method is tied closely to the simplex method, but Young's simplified primal algorithm adds a special row to the tableau and modifies the method of selecting the pivot column. At each iteration, a cutting-plane constraint is added to the tableau. Young shows his simplified primal algorithm is a finite procedure.

Young's (31) paper parallels Glover's (8) work.

Young's algorithm develops alternative rules for adding the new row of cutting-plane constraints. This method is not as general or the tableau format as elaborate as Glover's (8), according to Young's evaluation. Both papers were presented in the literature to speed the understanding and analysis of the basic approach to integer linear programming.

From an applications point-of-view, the ability of computer codes to solve integer programming problems is an essential consideration. Several cutting-plane algorithms have been coded and evaluated. Beale (4) mentions some codes that used the cutting-plane concepts of much of Gomory's early work in integer programming. As of Beale's (4) 1965 survey date, computer codes were available that solved about 100 equations and 2000 variables. Beale says that up to 1964 the largest single problem solved with this code had 215 equations and about 2600 variables. The author does not mention the speed or efficiency of the computer code, but refers his readers to the author of the computer code.

By the summer of 1967, C. A. Trauth, Jr. and R. E. Woolsey (28) had completed their analysis of four different computer codes. The codes were based primarily on Gomory's cutting-plane methods of integer linear programming. The authors compared the computational efficiency and practical applicability of the four codes. They discovered difficulties with machine round-off errors can sometimes be



controlled, but the procedures required can be time consuming. Another difficulty they noted was the amount of time required to actually solve a given integer programming problem. As mentioned earlier in this literature review, Gomory proved his methods will produce a solution in a finite number of steps. In practice, a finite number of steps can be so large as to be impractical, even in a moderate size problem.

Trauth and Woolsey (28) indicated the amount of time involved in obtaining a solution was related to the density of the constraint coefficient matrix. Also, the magnitude of the elements in this matrix had an effect on the solution time. The four codes were tested on some test problems that are commonly used in the literature to evaluate computer codes. The authors present tables and their analysis showing the solution time and number of iterations required for each computer code they evaluated.

Even though much important work has been done on cutting-plane algorithms, a great deal of attention has been given in the literature to backtrack algorithms. Some typical articles from this literature will now be reviewed.

#### Backtrack Algorithms

In the middle of 1960, a paper appeared that suggested an approach different from the cutting-plane method to solve discrete programming problems. The numerical algorithm of A. H. Land and A. G. Doig (20) is widely respected and

referenced in the literature of integer programming. This early work of Land and Doig (20) is often classified as a shifted hyperplane method or a branch-and-bound method. The algorithm described by the authors uses a systematic parallel shift of the objective function in the direction of a reduction of the maximand. This process is continued until an integer solution is found within the ordinary feasible solution space. The upper bound on the objective function is first found by solving the ordinary linear programming problem without the discrete variable constraints. This is the upper bound on the maximand since no higher value of the objective function can take an integer value.

The method of Land and Doig (20) then moves to identify a unique minimum and maximum for each variable at a particular value of the objective function. These minimum and maximum values of the variables can be found by solving the linear programming problems that minimize and maximize each variable. The authors extend these basic concepts to examining the convex set of feasible solutions as the objective function hyperplane is moved down from its maximum position. They give a step-by-step algorithm of their procedure and an example of its application.

E. L. Lawler and D. E. Wood (21) have written an excellent paper discussing branch-and-bound methods of integer linear programming. They describe the main concepts of the branch-and-bound approach to constrained optimization problems. Even though this article is not limited to a

discussion of branch-and-bound in integer linear programming, it still often appears as an important reference in the literature. The authors discuss several specific applications, including integer linear programming, nonlinear programming, the traveling-salesman problem, and the quadratic assignment problem.

Lawler and Wood (21) point out that, as in dynamic programming, the technique of branching-and-bounding is an intelligent examination of the feasible solution space. They describe the branch-and-bound method as repeatedly separating the feasible solution space into smaller and smaller subsets of feasible solutions. Within each subset a bound is calculated for the value of the objective function. After each separation of the feasible solution space, the subsets that have a bound that exceeds the value of the objective function for a known feasible solution are excluded from any further separation and examination. The authors present a generalized, formal mathematical description of the branch-and-bound algorithm.

In the last part of 1966, an article by Norman Agin (1) appeared. This article gave a generalized description of branch-and-bound algorithms. The author's paper points out the wide variety of applications of the branch-and-bound algorithm to many combinatorial problems. One of the best aspects of this paper is that it describes the branch-and-bound technique for optimum seeking in general. Agin says part of the philosophy of this paper is to introduce

branch-and-bound to those who are unfamiliar with the technique.

Agin (1) points out two interesting limitations of branch-and-bound methods. One is that each problem needs a specific method for finding the bound and for finding good heuristics for branching. Another limitation is that in large problems the computational time may exceed the available computer time.

One of the newest algorithms to appear is the bound-and-scan algorithm of Frederick S. Hillier (15). This technique applies to pure integer linear programming. The approach is to find tight bounds on the variables. Then, a sequence of constantly improving feasible solutions is identified by scanning the relevant solutions. Hillier (15) reports encouraging computational experience with this algorithm as compared to other existing methods. This is an excellent, readable paper that describes the new method and plans for increasing its efficiency even more.

Establishing bounds on the problem variables is a common principle in many backtrack algorithms. Patrick D. Krolak (17, 18) has completed some work that lead to a Bounded Variable Algorithm. These papers present some useful generalized equations to establish upper and lower bounds on variables. Krolak (18) presents some computational results of this algorithm and other existing methods when they are tested on some standard problems.

Stanley Zionts (32) proposed some ideas toward unifying

the theory of integer linear programming. Basically, the author generalizes much of the work in integer linear programming in the framework of upper and lower bounds on integer variables. The main contribution of this work is that it tends to unify several of the proposed methods of solving integer linear programming problems.

One of the most important applications of integer programming is in capital budgeting problems. Zero-one and mixed zero-one integer programming are mathematical tools that are essential to the solution of many capital budgeting problems. V. E. Unger, Jr. (29) describes how some of the zero-one algorithms can be used to assist a firm with the allocation of limited amounts of capital. Even though this article deals with only one class of the capital budgeting problem, the problem formulation and solution procedure make it an interesting article.

Some additional examples of papers from the literature that discuss the work on zero-one integer programming are Glover (9) and Geoffrion (7). Both of these authors are often referenced in the literature of integer linear programming.

#### Other Methods

The approach used in many methods of integer programming cannot be easily classified as cutting-plane or back-track algorithms. An example of this is an article by G. Graves and A. Whinston (13). The authors present a new

approach to discrete mathematical programming for zero-one integer programming. This paper describes the theoretical concepts that extend some of the enumeration methods. Where many of the backtrack algorithms would use bounds to truncate parts of the method, Graves and Whinston (13) use population statistics. The authors indicate that the term population statistics should not be confused with the methods of sampling statistics or random search procedures. The concepts described in this paper use the idea of selecting the optimal function among a certain class of functions. Set theory and functions viewed as maps are used to develop the concepts of this theoretical paper.

Approximation methods provide the only method of solving many integer programming problems. S. Senju and Y. Toyoda (27) have approached the zero-one integer linear programming problem from the point-of-view of trying to find a good approximate solution. The fundamental concept the authors say they use is to develop some ordinal scales for the proposed projects. They suggest this method is quite satisfactory when there are a large number of proposals and constraints.

Frederick S. Hillier (14) has developed an approximation method for integer linear programming. This paper is very well written and is an excellent example of some of the better work in the literature. This article presents the theoretical concepts of a heuristic procedure to find a good approximate solution which gives an objective function value

close to the optimum. The step-by-step method of analysis is outlined by the author. A small example is given to demonstrate the algorithm. A computer code and the results of evaluating the procedure on several test problems are given. Since many optimal algorithms can obtain greater efficiency when given a good approximate solution initially, this paper makes an outstanding contribution to the literature.

In a recent paper, R. E. Gomory and E. L. Johnson (12) introduce some theory that has applications to both cutting-plane and backtrack algorithms. The authors analyze some continuous functions and inequalities when some or all of the variables are restricted to be integers. The article shows how inequalities can be used to furnish cut-off points for integer programming algorithms. This paper is very theoretical and difficult to read. A companion paper demonstrating the basic technique would be a fine contribution to the literature.

The concepts of linear Diophantine equations are essential in the development of objective function reduction algorithm. Some of the necessary concepts will be reviewed in the following chapter.

## CHAPTER III

### SOME CONCEPTS OF LINEAR

### DIOPHANTINE EQUATIONS

The concepts of linear Diophantine equations from the theory of numbers are essential to the objective function reduction algorithm presented in this thesis. Several topics will be discussed in this chapter that are necessary to follow the flow of logic in the algorithm developed in this research. Even though number theory is often considered one of the prime examples of pure mathematics, some of the observations associated with linear Diophantine equations and the divisibility property of integers form a vital part of the objective function reduction algorithm. Some of the basic theorems from this area of mathematics allow implicit examination of several solutions in integer linear programming problems. Many of these concepts speed the algorithm and the computer code toward the integer feasible solution.

The wide variety of topics associated with linear Diophantine equations and number theory is discussed in several books. Anthony J. Pettofrezzo and Donald R. Byrkit (26) present several selected topics in number theory in a very readable and interesting book. This book describes



many essential topics of number theory at an introductory level. Ivan Niven and Herbert S. Zuckerman (24) have written another book that introduces several of the concepts of number theory and Diophantine equations. It includes many examples and proofs necessary to understand this area of mathematics.

In order to examine the philosophy of the divisibility properties of integers, some basic terms and notation are required. If  $r$ ,  $s$ , and  $t$  are integers such that

$$r \cdot s = t, \quad (3-1)$$

then  $r$  and  $s$  are called factors or divisors of  $t$ . Also,  $t$  is said to be a multiple of  $r$  and of  $s$ . If  $m$  is a factor of  $n$ , then it is written

$$m \mid n. \quad (3-2)$$

For example, if  $m = 5$  and  $n = 25$ , then  $5 \mid 25$  says 5 is a factor of 25. If  $m$  is not a factor of  $n$ , then the notation

$$m \nmid n \quad (3-3)$$

is written to describe how the integers are related. For example, if  $m = 2$  and  $n = 15$ , then  $2 \nmid 15$  says 2 is not a factor of 15.

The terms prime and composite often appear in discussions of linear Diophantine equations and the theory of numbers. A prime number is a number that has no positive factors other than one and itself. The number seven is a

prime number. A composite number is a number that has factors other than one and itself. Since the number nine has the number three as a factor, as well as one and itself, it would be called a composite number

The idea of the greatest common divisor is an integral part of the objective function reduction algorithm. This topic and its notation will be reviewed briefly.

### The Greatest Common Divisor

If  $k|r$  and  $k|s$ , then the number  $k$  is called a common divisor or common factor of  $r$  and  $s$ . Suppose  $r = 50$  and  $s = 80$ . The integer number 5 is a common divisor to both 50 and 80. Therefore,  $5|50$  and  $5|80$  indicates that 5 is a common divisor to 50 and 80. The greatest common divisor can now be defined. The largest positive integer  $g$  that divides the absolute value of each of two integers  $r$  and  $s$  is called the greatest common divisor of  $r$  and  $s$ . The greatest common divisor is denoted

$$(r, s) = g. \quad (3-4)$$

This implies  $g|r$  and  $g|s$ , with  $g$  as the greatest positive integer that is a factor to both  $r$  and  $s$ .

An objective function for a two-dimensional linear programming problem would have the form

$$Z = c_1 x_1 + c_2 x_2. \quad (3-5)$$

For a particular problem, a greatest common divisor could be

found if both  $c_1$  and  $c_2$  were integers. That is, a factor of  $c_1$  and  $c_2$  can be found such that

$$(c_1, c_2) = g. \quad (3-6)$$

In an elementary integer linear programming problem, the objective function might be

$$z = 21 x_1 + 77 x_2. \quad (3-7)$$

The greatest common divisor of both  $c_1$  and  $c_2$  would be the number 7. This would be denoted

$$(c_1, c_2) = g \quad (3-8)$$

$$(21, 77) = 7. \quad (3-9)$$

The definition of the greatest common divisor says that  $g$  must be a common divisor to the absolute value of  $r$  and  $s$ . Therefore, negative coefficients can be handled within the greatest common divisor definition. If, in the example of Equation (3-7), one or both of the coefficients were negative, then

$$(-21, 77) = (21, -77) = (-21, -77) = 7. \quad (3-10)$$

The definition also implies that if  $r = s = 0$ , then  $(r, s)$  does not exist. Additionally, if  $r \neq 0$  and  $s = 0$ , then  $(r, s) = |r|$ ; if  $r = 0$  and  $s \neq 0$ , then  $(r, s) = |s|$ . As somewhat of a side issue, if  $d|r$  and  $d|s$ , then  $d|g$ . This says that if  $d$  is a factor of both  $r$  and  $s$ , where  $(r, s) = g$ , then  $d$  is also a factor of the greatest common divisor.

The concept of greatest common divisor is also applicable when more than two integers are involved. The definition can be generalized so that the greatest common divisor for  $n \geq 3$  integers is the largest positive integer  $g$  such that it is a factor of the absolute values of each of the  $n$  integers  $r_1, r_2, \dots, r_n$ . For the  $n$  integers,  $r_1, r_2, \dots$ , and  $r_n$ , the greatest common divisor is written

$$(r_1, r_2, \dots, r_n) = g. \quad (3-11)$$

For example, suppose an integer linear programming objective function were

$$z = 56 x_1 - 24 x_2 + 96 x_3 + 40 x_4. \quad (3-12)$$

The greatest common divisor of the objective function coefficients is

$$(r_1, r_2, r_3, r_4) = g \quad (3-13)$$

$$(c_1, c_2, c_3, c_4) = g \quad (3-14)$$

$$(56, -24, 96, 40) = 8. \quad (3-15)$$

As the general definition indicates, the greatest common divisor of the coefficients is the greatest integer,  $g$ , where

$$g | c_j \quad \text{for } j = 1, 2, \dots, n. \quad (3-16)$$

Two theorems are needed to give additional insight to the greatest common divisor and its application in the objective function reduction algorithm.

THEOREM 3.1: If  $v$  and  $w$  are positive integers such that  $v|w$  and  $w|v$ , then  $v = w$ .

PROOF: If  $v|w$ , then  $v$  times some number, call it  $x$ , must equal  $w$ . That is,

$$v \cdot x = w \quad (3-17)$$

and this implies  $x|w$ . Also, if  $w|v$ , then  $w$  times some number, call it  $y$ , must equal  $v$ . Therefore,

$$w \cdot y = v \quad (3-18)$$

and this implies  $y|v$ . Now, if Equations (3-17) and (3-18) are solved for  $x$  and  $y$ , respectively, then

$$x = \frac{w}{v} \quad (3-19)$$

and

$$y = \frac{v}{w}. \quad (3-20)$$

Now, Equation (3-19) can be written

$$x = \frac{w}{v} = \frac{1}{\frac{v}{w}} = \frac{1}{y}. \quad (3-21)$$

By the definition of the term factor as used in Equation (3-17),  $v$ ,  $x$ , and  $w$  are integers. But, Equation (3-21) says

$$x = \frac{1}{y}, \quad (3-22)$$

which means  $y$  must be equal to one for  $x$  to be integer. This implies from Equation (3-17) that

$$v = w. \quad (3-23)$$

Another theorem that is needed deals with the greatest common divisor for the situations where three or more numbers are involved. This theorem will make use of Theorem 3.1 during its proof.

THEOREM 3.2: If  $c_1, c_2, \dots$ , and  $c_n$  are nonzero integers where  $n \geq 3$ , then

$$(c_1, c_2, \dots, c_n) = ((c_1, c_2, \dots, c_{n-1}), c_n). \quad (3-24)$$

PROOF: Let

$$g_1 = (c_1, c_2, \dots, c_n) \quad (3-25)$$

and

$$g_2 = ((c_1, c_2, \dots, c_{n-1}), c_n). \quad (3-26)$$

Since  $g_1 | c_j$  for  $j = 1, 2, \dots, n$ , then

$g_1 | (c_1, c_2, \dots, c_{n-1})$  and  $g_1 | c_n$ . This implies

$g_1 | g_2$ . Also,  $g_2 | (c_1, c_2, \dots, c_{n-1})$  and  $g_2 | c_n$ ,

then  $g_2 | c_j$  for  $j = 1, 2, \dots, n$ . Therefore,

$g_2 | g_1$ . The definition of greatest common divisor

requires that  $g_1$  and  $g_2$  be positive integers.

Therefore,  $g_1 = g_2$  from Theorem 3.1 and

$$(c_1, c_2, \dots, c_n) = ((c_1, c_2, \dots, c_{n-1}), c_n). \quad (3-27)$$

Another important concept that can be used in integer linear programming is when some coefficients of variables in the objective function are zero. Should that case occur, the greatest common divisor is

$$(c_1, c_2, \dots, c_k, 0, 0, \dots, 0) = (c_1, c_2, \dots, c_k). \quad (3-28)$$

For small problems, the greatest common divisor often can be found by inspection. As the number of integers increases, the search for the greatest common divisor would become lengthy and tedious using inspection and trial and error methods. Fortunately, the ancient Greek mathematician, Euclid, developed an algorithm to determine the greatest common divisor. Euclid's algorithm will now be examined.

#### Euclid's Algorithm

Euclid's algorithm gives a method for finding the greatest common divisor for a group of integers. First, consider the case of two unequal positive integers  $d$  and  $e$ . If it is assumed that  $d > e$ , then  $d$  can be written as

$$d = qe + r, \quad \text{where } 0 < r < e. \quad (3-29)$$

For example, let  $d = 23$  and  $e = 7$ . Therefore,  $d$  can be written as

$$23 = 3 \cdot 7 + 2. \quad (3-30)$$

From Equation (3-29),  $q = 3$  and  $r = 2$  for the example of Equation (3-30). The integer  $q$  is the quotient resulting from the division of  $d$  by  $e$ . The integer  $r$  is the remainder after  $d$  is divided by  $e$ . Euclid developed a step-by-step procedure for continuing this type of analysis until it leads to the greatest common divisor.

Let  $q_i$  be the quotient from the  $i^{\text{th}}$  iteration and  $r_i$  the remainder associated with the  $i^{\text{th}}$  iteration. For a simple example that requires only three iterations, Euclid's algorithm would be as follows:

$$d = q_1 e + r_1, \text{ where } 0 < r_1 < e; \quad (3-31)$$

$$e = q_2 r_1 + r_2, \text{ where } 0 < r_2 < r_1; \quad (3-32)$$

$$r_1 = q_3 r_2 + r_3, \text{ where } r_3 = 0. \quad (3-33)$$

The algorithm is completed when the remainder in the  $i^{\text{th}}$  iteration is zero. The greatest common divisor is the last nonzero remainder ( $r_{i-1}$ ) found by the algorithm. If  $r_3$  were found to be zero in Equation (3-33), then the integer  $r_2$  is the greatest common divisor for  $d$  and  $e$ . That is,

$$(d, e) = r_2. \quad (3-34)$$

As an example, suppose an integer programming objective function were

$$z = 36 x_1 + 132 x_2. \quad (3-35)$$

Using Euclid's algorithm as described in Equations (3-31)



through (3-33), the greatest common divisor of the objective function coefficients can be found as follows:

If  $d = 132$  and  $e = 36$ , then

$$132 = 3 \cdot 36 + 24 \quad (3-36)$$

$$36 = 1 \cdot 24 + 12 \quad (3-37)$$

$$24 = 2 \cdot 12 + 0. \quad (3-38)$$

From Equation (3-38),  $r_3 = 0$  which means the greatest common divisor,  $g$ , is  $r_2 = 12$ . Therefore, in the notation developed here,

$$(d, e) = g \quad (3-39)$$

$$(d, e) = r_2 \quad (3-40)$$

$$(132, 36) = 12. \quad (3-41)$$

It should be noted that the integer coefficients in Equation (3-35) were ordered so the larger one ( $c_2 = 132$ ) became  $d$  for the algorithm of Equations (3-31) through (3-33), where it is required that  $d > e$ . This is a necessary condition so the larger integer can be set equal to a quotient times the smaller number plus a remainder.

The form of Euclid's algorithm described in Equations (3-31) through (3-32) allows for only three iterations. Naturally, other problems may require several iterations. In the example for  $(d, e)$  the greatest common divisor,  $g$ , must be found in a finite number of steps, since there is

only a finite number of positive integers less than  $e$ , where  $d > e$ . The general form of Euclid's algorithm to find  $(d, e) = g$ , where  $d > e$ , is as follows:

$$d = q_1 e + r_1, \text{ where } 0 < r_1 < e; \quad (3-42)$$

$$e = q_2 r_1 + r_2, \text{ where } 0 < r_2 < r_1; \quad (3-43)$$

$$r_1 = q_3 r_2 + r_3, \text{ where } 0 < r_3 < r_2; \quad (3-44)$$

. . .

$$r_{k-3} = q_{k-1} r_{k-2} + r_{k-1}, \text{ where } 0 < r_{k-1} < r_{k-2}; \quad (3-45)$$

$$r_{k-2} = q_k r_{k-1} + r_k, \text{ where } 0 < r_k < r_{k-1}; \quad (3-46)$$

$$r_{k-1} = q_{k+1} r_k + 0. \quad (3-47)$$

The concepts of the greatest common divisor and Euclid's algorithm provide a foundation for examining some of the properties of linear Diophantine equations.

### Linear Diophantine Equations

Any polynomial equation in several variables, where all of the coefficients, variables, and the right-hand side values are required to be integers, is called a Diophantine equation. In general, the form would be

$$\sum_{j=1}^n c_j x_j = z \quad (3-48)$$

where

$c_j$  integer      for  $j = 1, 2, \dots, n$

$x_j$  integer      for  $j = 1, 2, \dots, n$   
 $z$  integer.

The Greek mathematician, Diophantus, studied the form and solution of this type of linear equation. Since he was one of the first to study these equations at great length, they are named in his honor.

An interesting theorem associated with the greatest common divisor and linear Diophantine equations will be discussed.

THEOREM 3.3: If  $c_1, c_2, \dots$ , and  $c_n$  are integers which are not all zero, then the greatest common divisor  $(c_1, c_2, \dots, c_n)$  of the coefficients  $c_1, c_2, \dots$ , and  $c_n$  is the smallest positive integer that can be expressed as a linear homogeneous function of  $c_1, c_2, \dots$ , and  $c_n$ ; that is,  $(c_1, c_2, \dots, c_n)$  is the smallest positive integer such that

$$(c_1, c_2, \dots, c_n) = c_1 x_1 + c_2 x_2 + \dots + c_n x_n, \quad (3-49)$$

where  $x_j$  integer      for  $j = 1, 2, \dots, n$ .

Pettoufrezzo and Byrkit (26) indicate how this theorem can be proved. This theorem has a significant implication in the objective function reduction algorithm. If an integer programming objective function is of the form

$$\text{MAXIMIZE} \quad z = \sum_{j=1}^n c_j x_j; \quad (3-50)$$

then, the greatest common divisor could be found for the coefficients, such that

$$g = (c_1, c_2, \dots, c_n). \quad (3-51)$$

The implication of Theorem 3.3 is that the greatest common divisor,  $g$ , is the lower integer bound of a maximization objective function value. That is, the objective function hyperplane

$$\sum_{j=1}^n c_j x_j = g \quad (3-52)$$

is the smallest integer value of the maximand.

Another property of linear Diophantine equations involves the special case where all of the coefficients are even numbers and the right-hand side value is odd. For example, if an equation had the form

$$12 x_1 + 34 x_2 + 8 x_3 + 92 x_4 = 533, \quad (3-53)$$

then it can be observed that all the coefficients are even numbers, while the right-hand side value is an odd number. Since the left-hand side of Equation (3-53) must be an even integer, no integral solutions can exist for the equation. This comes from the observation that an even integer multiplied times an odd or even integer must give an even integer as the product. Consequently, some linear Diophantine equations have no integer solutions. This fact will be used in the objective function reduction algorithm as solutions are implicitly examined.

The next theorem is most important for the search method of the objective function reduction algorithm developed in this research.

THEOREM 3.4: The linear Diophantine equation

$$\sum_{j=1}^n c_j x_j = z \quad (3-54)$$

has a solution if and only if  $g|z$ , where  $g = (c_1, c_2, \dots, c_n)$ .

Pettoufrezzo and Byrkit (26) indicate how this theorem can be proved. The importance of this theorem lies in the fact that it can eliminate searching certain objective function hyperplanes for integral solutions. This says that an objective function of the form

$$z = \sum_{j=1}^n c_j x_j \quad (3-55)$$

where

$c_j$  integer for  $j = 1, 2, \dots, n$

$x_j$  integer for  $j = 1, 2, \dots, n$

can only have integral solutions for those values of  $z$  such that  $g|z$ , where  $g = (c_1, c_2, \dots, c_n)$ . Hyperplanes that have values such that  $g \nmid z$  need not be considered or searched, since integral solutions cannot lie on these hyperplanes.

As an example of how Theorem 3.4 could be used,

consider the following integer linear programming objective function:

$$z = 27 x_1 + 9 x_2 + 18 x_3. \quad (3-56)$$

Using Euclid's algorithm or by inspection, the great common divisor of the coefficients would be found to be  $g = 9$ .

Suppose the objective function had a value

$$z = 27 x_1 + 9 x_2 + 18 x_3 = 82 \quad (3-57)$$

during one part of the search by the objective function reduction algorithm. Since  $g = 9$  and  $9 \nmid 82$ , no integral solutions can lie on the plane defined in Equation (3-57).

With this brief review of some of the concepts of linear Diophantine equations and the theory of numbers, the objective function reduction algorithm will be examined.

## CHAPTER IV

### THE OBJECTIVE FUNCTION

#### REDUCTION ALGORITHM

The objective function reduction algorithm is a solution procedure that searches for a solution to a solvable pure integer linear programming problem. This algorithm uses the concept of examining a family of objective function hyperplanes until an integer solution is found. Basically, the solution procedure begins at the optimum noninteger solution and examines parallel objective function planes in the feasible solution space. For example, in maximization problems the algorithm starts at the upper bound on the objective function, as determined by the simplex method or some other primal optimum seeking procedure. Successively reduced values of the objective function effectively move the objective function down through the solution space. The problem to be solved can be expressed mathematically using the following canonical form:

$$\text{maximize} \quad z = \sum_{j=1}^n c_j x_j \quad (4-1)$$

$$\text{subject to} \quad \sum_{j=1}^n a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \dots, m \quad (4-2)$$

$$x_j \geq 0 \text{ for } j = 1, 2, \dots, n \quad (4-3)$$

$$x_j, c_j \text{ INTEGER for } j = 1, 2, \dots, n. \quad (4-4)$$

The requirement that each  $c_j$  be integer valued is not overly restrictive since this is always obtainable by scaling the objective function, as long as the original coefficients are rational numbers.

The algorithm developed in this research examines the solution space by considering the bounds on each variable. Therefore, this procedure requires that the variables have upper and lower integer bounds. The functional and nonnegativity constraints of Equations (4-2) and (4-3) are assumed to provide a bound on the solution space. A minimization problem with strictly greater-than-or-equal constraints must be modified to obtain finite upper bounds on each problem variable. This is required for computational efficiency.

The essential structure of the objective function reduction algorithm can be described by dividing the solution procedure into four stages. In Stage 1, the optimal continuous-variable solution is found using the simplex method or some similar procedure. Naturally, if this solution is all-integer, the algorithm goes no further since the desired solution has been found. Stage 1 also defines the over-all bounds on each problem variable. Stage 2 of the algorithm prepares the way to potentially take advantage of some techniques from the study of linear Diophantine equations. The greatest common divisor of the objective



function coefficients is established along with the first value of the objective function to be considered. Stage 3 selects the ranking that each variable will have in the implicit enumeration scheme. It also examines the objective function coefficients noting how even coefficients might be used to take advantage of additional concepts of linear Diophantine equations.

While the first three stages of the algorithm are somewhat preparatory, Stage 4 carries out the implicit and explicit enumeration of the feasible integer solution space. Using the ranking scheme of Stage 3, the problem variables are set at integer values that potentially will eliminate the necessity for complete enumeration of the integer solution space. New, potentially tighter bounds are found on successive variables in the ranking as the algorithm proceeds. This process continues to move through the ranking order, until the next to last variable is reached or until the algorithm can use a tool of linear Diophantine equations to eliminate additional solutions. The objective function and the previous variables at their held values are used to calculate the final variable in the ranking. An integer solution is tested for feasibility, while a noninteger final variable is immediately identified as infeasible. An infeasible solution causes the algorithm to begin moving back through the combinatorial solution possibilities. Should a feasible all-integer solution not be found at the first objective function value, a new

reduced (for maximization) objective function value is selected for continuing the search.

With the above brief statements as an introduction, the theory of the objective function reduction algorithm will be described fully in the following sections. The step-by-step instructions of the algorithm are provided in Appendix A. This appendix can be used as a reference as the theory is described.

### Stage One

Stage 1 of the objective function reduction algorithm begins by relaxing the integer requirement on the problem variables. Therefore, the variables are temporarily allowed to take on continuous values. With this interim change in the problem, the efficient procedures of the simplex method can be used to find the continuous solution. At this point, the algorithm identifies two elements of information from the simplex solution. First, if it is found that the continuous solution vector,  $\underline{x}^* = (x_1^*, x_2^*, \dots, x_n^*)$ , is all-integer, then the algorithm goes no further. The optimum pure integer solution has been found by the established optimal seeking procedures of the simplex method. The second element of required information is the value of the objective function at the optimum continuous-variable linear programming solution. The key word for this value is ZSIM. It will be used in Stage 2 to establish an integer bound on the objective function.

Integer bounds on each problem variable are an

additional requirement of Stage 1. Patrick D. Korlak (17, 18) and Stanley Zionts (32) have published some recent papers describing their work on integer bounds for all-integer linear programming. Krolak (17, 18) develops an iteration scheme to be used in finding upper and lower integer bounds for each individual variable. Zionts (32) attempts to unify much of the work of integer programming in terms of upper and lower bounds on integer variables.

Krolak (17) suggests one straightforward method of finding the bounds on the variables is to solve the  $2n$  linear programming problems where the objective functions are of the form

$$\text{maximize } z = x_j \quad \text{for } j = 1, 2, \dots, n \quad (4-5)$$

and

$$\text{minimize } z = x_j \quad \text{for } j = 1, 2, \dots, n. \quad (4-6)$$

For a particular problem, the constraints and Equation (4-5) can be used to solve a linear programming problem to find the upper bounds on each variable,  $x_j$ ,  $j = 1, 2, \dots, n$ . Lower bounds can be similarly found using Equation (4-6). The upper integer bound on each variable is identified by taking the value of  $x_j$  found from the linear programming solution using Equation (4-5) and defining

$$x_j^u = [x_j] \quad (4-7)$$

for  $j = 1, 2, \dots, n$ . Using the solution value for each  $x_j$  from the linear programming problem where Equation (4-6) is the

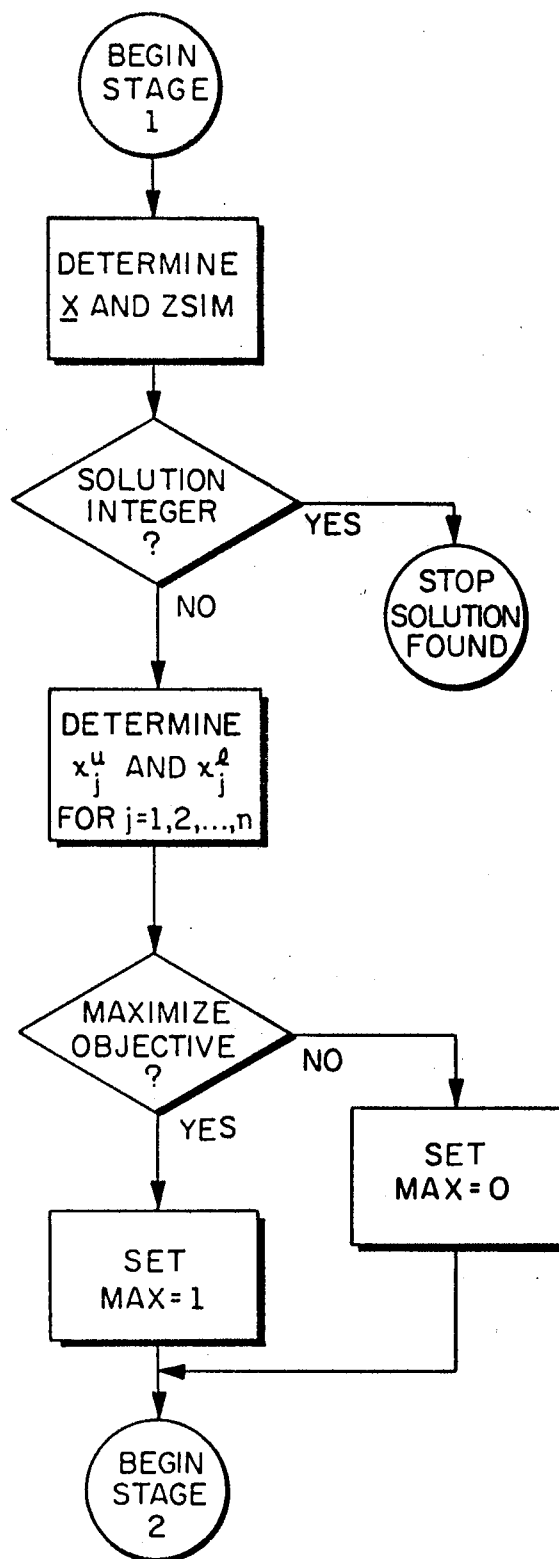


Figure 4. Stage 1 Logic Flow Diagram

objective function, the lower integer bounds are

$$x_j^{\ell} = \langle x_j \rangle \quad (4-8)$$

for  $j = 1, 2, \dots, n$ .

The objective function reduction algorithm requires that the problem variables have finite integer bounds. Therefore, the method described in Equations (4-5) through (4-8) will establish upper and lower integer bounds on each variable.

A final requirement for Stage 1 is an indication whether the objective function is to be maximized or minimized. In both the algorithm of Appendix A and the computer code of Appendix B, the key word MAX is used to indicate maximization or minimization. Maximization is indicated when  $MAX = 1$  and minimization is identified by setting  $MAX = 0$ . Figure 4 shows a flow chart for Stage 1.

### Stage Two

Some of the concepts of linear Diophantine equations and the theory of numbers are used in Stage 2. First, the greatest common divisor,  $g$ , of the objective function coefficients is determined. Given an objective function of the form

$$z = \sum_{j=1}^n c_j x_j, \quad (4-9)$$

then the greatest common divisor of the  $c_j$  coefficients is

defined as

$$(c_1, c_2, \dots, c_n) = g. \quad (4-10)$$

In Chapter III, Euclid's algorithm is described. This algorithm can be used to find the greatest common divisor for Equation (4-9), where  $c_j$  is integer, for  $j = 1, 2, \dots, n$ .

When the greatest common divisor is greater than one, the algorithm takes advantage of the implication of this fact. If  $g > 1$ , then certain parallel hyperplanes can be eliminated from consideration, since integer solutions can only occur when  $g|k$ , where  $k$  is some specific objective function value. This very important observation can significantly reduce the number of combinatorial solutions the algorithm must examine. Very simply, if  $g \nmid k$ , where  $(c_1, c_2, \dots, c_n) = g$ , then no integer solution can lie on the hyperplane

$$\sum_{j=1}^n c_j x_j = k. \quad (4-11)$$

The initial objective function value that is considered is identified by the key word ZOF in the statement of the algorithm in Appendix A and in the computer code. In Stage 1, the optimum continuous-variable simplex solution, ZSIM, was established. For a maximization problem where the greatest common divisor is  $g = 1$ , then the first of objective function value used by the algorithm is

$$ZOF = [ZSIM]. \quad (4-12)$$

If  $g \neq 1$ , then ZOF is set equal to the greatest integer less than ZSIM that has  $g|ZOF$ . Similarly, if minimization problems are to be solved, then

$$ZOF = \langle ZSIM \rangle \quad (4-13)$$

when  $g = 1$ ; and, ZOF is set equal to the smallest integer greater than ZSIM such that  $g|ZOF$  when  $g \neq 1$ .

The logic of beginning at the continuous-variable solution is an important part of the objective function reduction algorithm. In many problems, the optimum integer solution lies on a hyperplane that is very near the hyperplane that contains the optimum noninteger solution. Unfortunately, the set of solutions to the integer programming problem is not convex. If only the space near the optimum noninteger solution is searched, then only a local optimum can be assured with an integer solution. Therefore, the objective function reduction algorithm begins at the hyperplane that contains the optimum noninteger solution because the optimum integer solution is often nearby, but the search method of Stage 4 considers successive hyperplanes and uses procedures to identify a global optimum.

Where  $g = 1$ , for maximization, the first integer value of the objective function, ZOF, was defined to be  $ZOF = [ZSIM]$ . This means the simplex optimum objective function value, ZSIM, is rounded down to the greatest integer less-than-or-equal-to ZSIM. The proof that this is an acceptable place to begin the search will now be considered.

The fact that the simplex method finds the optimum continuous-variable solution is a basic axiom of

mathematical programming. Let the value of the objective function at the optimum continuous-variable solution be defined as

$$w = ZSIM \quad (4-14)$$

for this analysis. Therefore, the optimum objective function to the linear programming problem, where the integer restriction has been relaxed, would be of the form

$$\sum_{j=1}^n c_j x_j = w. \quad (4-15)$$

In Equation (4-15), the coefficients are restricted to integer values. The variables,  $x_j$ ,  $j = 1, 2, \dots, n$ , may be any real number within the limits set by the constraints. Obviously, this means  $w$  may assume a real number for the optimum simplex objective function value. Assuming the objective is to be maximized,  $w$  must set an upper bound on the objective function. Since each coefficient and variable on the left-hand side of Equation (4-15) must be integer in the final optimal all-integer solution, the sum of their products must be less-than-or-equal-to  $w$ .

The above analysis implies that when  $g = 1$ , then  $ZOF = [w]$  provides an upper bound on the objective function. The theorems of linear Diophantine equations require that  $g \mid ZOF$  for an all-integer solution to exist at a particular objective function value. Consequently, when  $g \neq 1$ , then  $[ZSIM]$  must be incremented down in integer amounts until it leads to a  $ZOF$  that has  $g \mid ZOF$ . Similar logic can be used



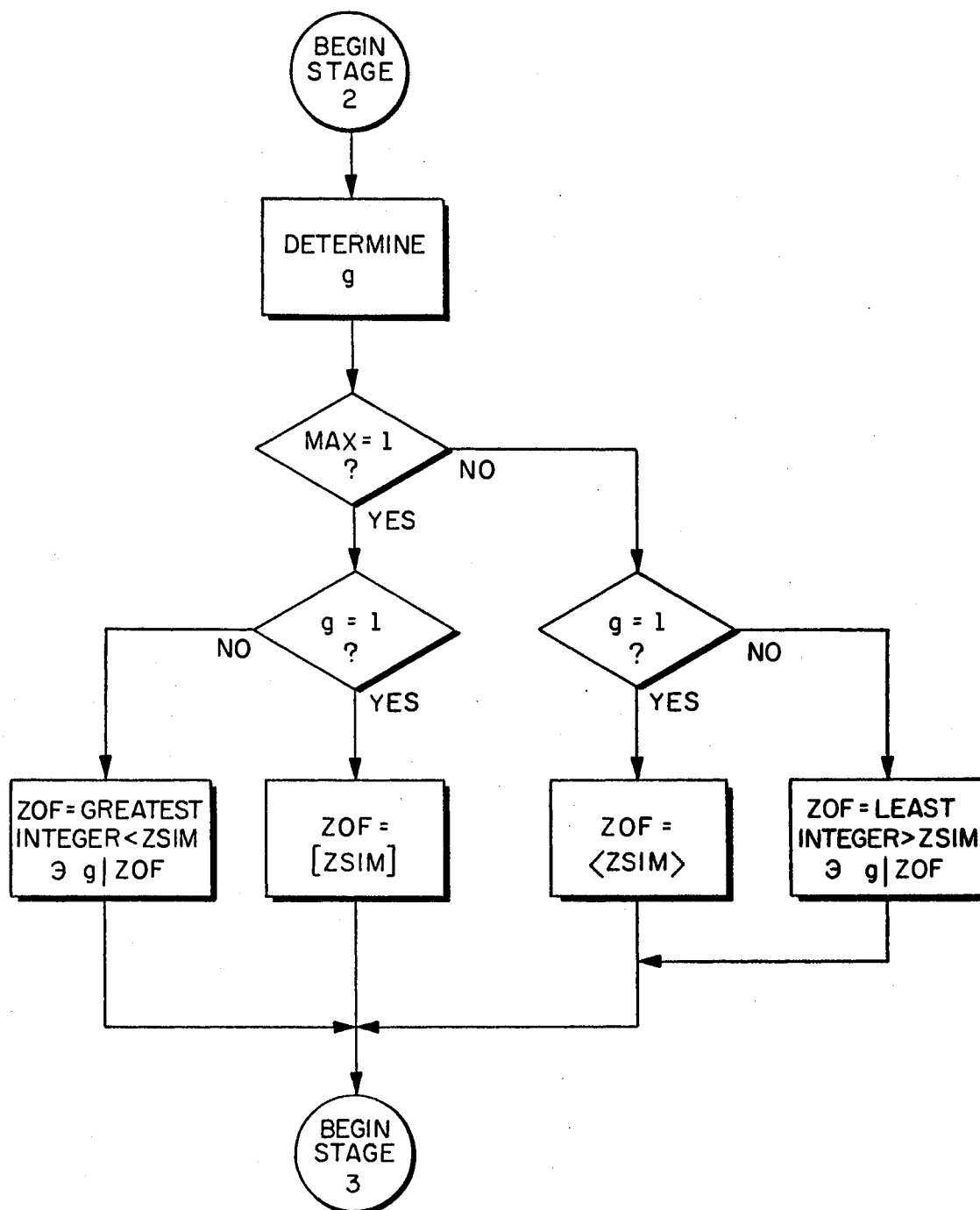


Figure 5. Stage 2 Logic Flow Diagram

to prove that the ZOF described for minimization problems is acceptable.

The logic of Stage 2 is shown in the flow diagram of Figure 5.

### Stage Three

In Stage 3, a ranking scheme for ordering the variables is specified. The objective of ordering the variables is to develop a procedure that will implicitly examine and eliminate several combinatorial solution possibilities. To do this, the bounds on each variable are examined. The upper and lower bounds determine the range of possible integer values for a variable. The range size is the number of integer possibilities for a particular variable. A permutation from the set of problem variables,  $\{x_j | j = 1, 2, \dots, n\}$ , is identified such that the range size of the variables,  $r(x)_j$ , proceeds from smallest to largest. That is, the first variable in the ranking has the smallest range of possible integer values, the second variable the next smallest range, and so on.

The ranked variables are assigned a new symbol,  $y_j$ , for  $j = 1, 2, \dots, n$ . The first variable in the ranking,  $y_1$ , represents the variable with the smallest range of possible integer values. This change of variables process continues so that the ranked variables will correspond to the ordered original problem variables. This will give  $r(y)_1 \leq r(y)_2 \leq \dots \leq r(y)_n$ , where  $r(y)_j$  is the range size of the  $j^{\text{th}}$  ranked

variable.

In some problems, the range size may be the same for two or more variables. The algorithm breaks the tie for ranking position by selecting the variable  $(x_j)$  with the largest objective function coefficient  $(c_j)$  to have the higher ranking. That is, if in some problem  $r(x)_s = r(x)_t$ , where  $s \neq t$ , then  $c_s$  and  $c_t$  must be examined. If  $c_s > c_t$ , then the algorithm would order the variables so  $x_s$  preceded  $x_t$  in the ranking. If the situation should occur such that  $r(x)_s = r(x)_t$  and  $c_s = c_t$ , then the tie for ranking position is arbitrarily broken.

The general philosophy of this ranking scheme is to describe a procedure that will mean that fewer combinatorial solutions will have to be explicitly examined. An additional way to move toward this goal is to use an observation from linear Diophantine equations. As was discussed in Chapter III, if all the coefficients in a Diophantine equation are even, then an odd right-hand side value means there are no integer solutions to that equation. In Stage 4, the objective function reduction algorithm will take advantage of this fact, whenever possible, to truncate the search. Using this concept, the algorithm examines the objective function coefficients  $(\hat{c}_j)$  that correspond to the ranked variables  $(y_j)$ . If a successive series of even coefficients occurs from the  $k^{\text{th}}$  to the  $n^{\text{th}}$  variable in the ranking, the algorithm records this fact. Stage 4 will describe the implicit examination of solutions that can be obtained using

this procedure.

An additional heuristic procedure that blends the ideas of range size and even coefficients has been developed. If it is found that the range sizes on the ranked variables are approximately the same, the speed of convergence can often be improved by purposefully placing any variables with even  $\hat{c}_j$  last in the ranking. Therefore, the variables would be first ranked according to range size. Then, change the ranking to place any variables with even  $\hat{c}_j$  at the end of the ordered variables. Again, this heuristic procedure is only advantageous when the  $r(y)_j$ , for  $j = 1, 2, \dots, n$ , are approximately the same and some  $\hat{c}_j$  are even integers.

One of the goals of Stage 4 will be to attempt to tighten the upper and lower bounds on the ranked variables. Before going to Stage 4, the upper and lower bounds are set equal to variables indicating temporary bounds. This is done by setting

$$y^{(\ell)}_j = y_j^{(\ell)} \quad (4-16)$$

and

$$y^u_j = y_j^u \quad (4-17)$$

for  $j = 1, 2, \dots, n$ . As the algorithm moves to Stage 4, the tightest bounds that are known are the ones given in Equation (4-16) and (4-17).

Figure 6 presents a flow diagram of the logic used in Stage 3.

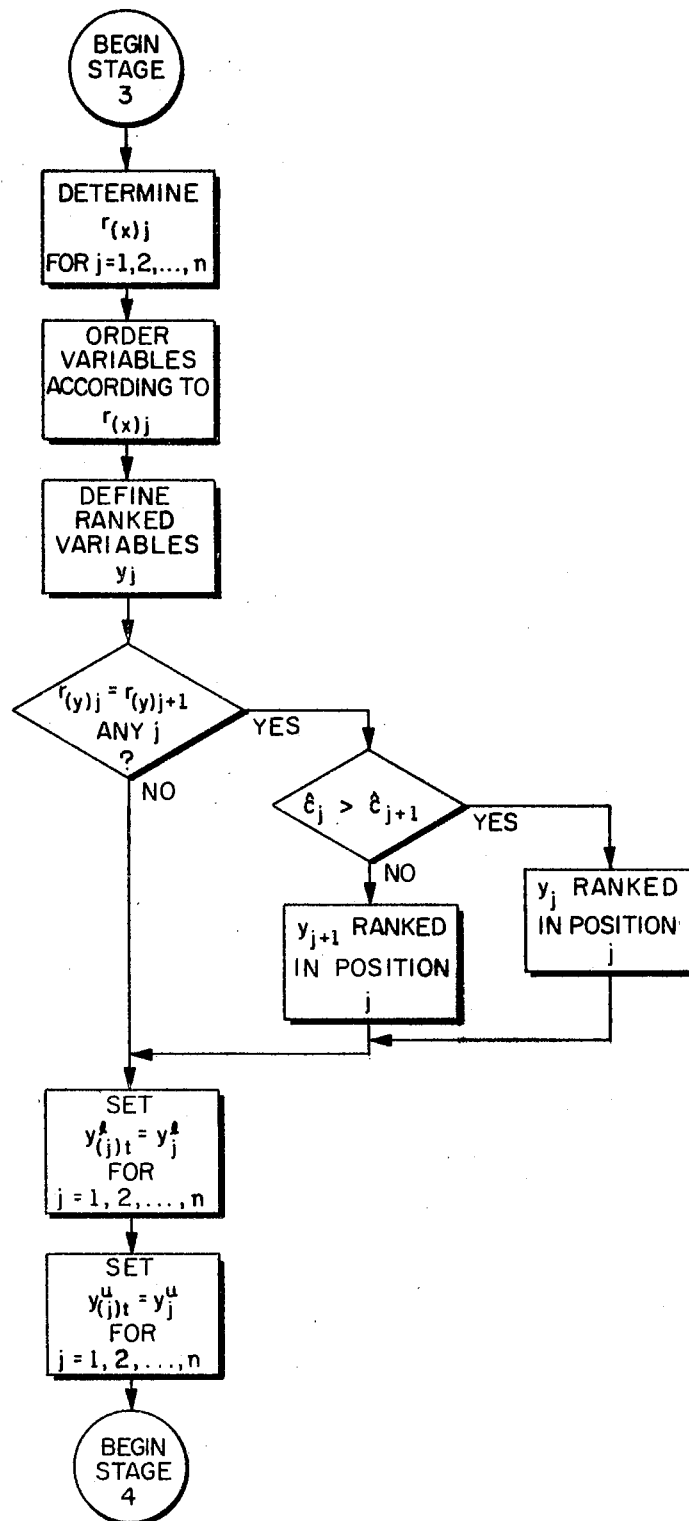


Figure 6. Stage 3 Logic Flow Diagram

### Stage Four

The essential features of Stage 4 will be considered in this section. Appendix A gives detailed, step-by-step instructions for Stage 4. In Stage 4, the search for an optimal integer solution begins. Explicitly or implicitly, all of the integer combinatorial possibilities are examined in the feasible solution space. Beginning at the optimal continuous-variable solution, a family of parallel objective function hyperplanes is searched for the optimal feasible solution.

The basic procedure is to let the variables take on integer values within their range of feasible values. When a variable is assigned a specific value, this means potentially tighter bounds can be found on each variable not assigned a specific value. This procedure of finding tighter bounds can often be used to eliminate the need for explicitly considering several integer combinatorial possibilities. The highest ranked variable,  $y_1$ , is set at its bound, such that

$$y_1 = y_{(1)t}^u \quad (4-18)$$

for maximization problems, or

$$y_1 = y_{(1)t}^l \quad (4-19)$$

for minimization problems. Also, initialize a temporary objective function value,  $z_t$ , such that  $z_t = ZOF$ . With  $y_1$  now assigned a value, the initial value of the objective

function,  $z_t = \text{ZOF}$ , can be modified. Therefore, the remaining variables can only take on values such that

$$\hat{c}_2 y_2 + \hat{c}_3 y_3 + \dots + \hat{c}_n y_n = z_t - \hat{c}_1 y_1. \quad (4-20)$$

This fact, alone, may exclude some values of certain variables from being considered for a given  $z_t$  and the held value of  $y_1$ . Still further, the held value of  $y_1$  also offers the possibility of tightening variable bounds due to the functional constraints. Where  $b_i^t$  is a temporary right-hand side value in the  $i^{\text{th}}$  constraint, the  $b_i^t$  are first initialized to the original right-hand side constants, such that  $b_i^t = b_i$  for  $i = 1, 2, \dots, m$ . The constraints are now of the canonical form

$$\sum_{j=1}^n \hat{a}_{ij} y_j \leq b_i^t \quad (4-21)$$

for  $i = 1, 2, \dots, m$ . Since  $y_1$  has been assigned an integer value, each  $b_i^t$ , for  $i = 1, 2, \dots, m$ , can be potentially modified. The new  $b_i^t$  are found from

$$b_i^t = b_i^t - \hat{a}_{i1} y_1 \quad (4-22)$$

for  $i = 1, 2, \dots, m$ .

If  $\hat{a}_{i1} \neq 0$ , a new tighter bound on the next variable in the ranking can now be found. If a maximization problem is being considered, then the new temporary upper bound on  $y_2$

is

$$y^u(z)_t = \min_{i=1,2,\dots,m} \{ [z_t/\hat{c}_2], [b_i^t/\hat{a}_{12}] \}. \quad (4-23)$$

Equation (4-23) imposes a new, temporary upper bound on  $y_2$ , given  $y_1$  is being held at some fixed value. Only coefficients where  $\hat{c}_2 > 0$  and  $\hat{a}_{12} > 0$  are considered when finding a tighter upper bound on  $y_2$ . If  $\hat{c}_2 \leq 0$  and all  $\hat{a}_{12} \leq 0$ , then no tighter bound on  $y_2$  is defined.

Similarly, for a minimization problem, the new tighter lower bound on  $y_2$  is found from

$$y^l(z)_t = \max_{i=1,2,\dots,m} \{ \langle z_t/\hat{c}_2 \rangle, \langle b_i^t/\hat{a}_{12} \rangle \}. \quad (4-24)$$

As before, only coefficients where  $\hat{c}_2 > 0$  and  $\hat{a}_{12} > 0$  are considered when finding a tighter lower bound on  $y_2$ . If  $\hat{c}_2 \leq 0$  and all  $\hat{a}_{12} \leq 0$ , then no tighter bound is defined for  $y_2$ .

Now,  $y_2$  is set at its new bound. If the objective function is to be maximized, then

$$y_2 = y^u(z)_t, \quad (4-25)$$

otherwise, for minimization set

$$y_2 = y^l(z)_t. \quad (4-26)$$

The fact that  $y_2$  has been assigned a value means that a new  $z_t$  and  $b_i^t$ , for  $i = 1, 2, \dots, m$ , can be found. This implies that a new, potentially tighter bound can be determined for



the next variable in the ranking,  $y_3$ . Then,  $y_3$  can be set at its new, temporary bound. This method continues until the algorithm finds it can truncate the process, where further enumeration would lead to infeasible solutions. Some of these methods of implicit examination will now be considered.

If at any time  $z_t$  is an odd integer during the process of assigning values to variables and finding tighter bounds, then the algorithm checks to see if this fact can be used. From the theory of linear Diophantine equations, an equation with even coefficients and an odd right-hand side value is immediately recognized as having no integer solution. Suppose the first  $k-1$  variables have been assigned a value in the ranking and the resulting  $z_t$  is an odd integer. If it is found that all succeeding objective function coefficients from  $\hat{c}_k$  to  $\hat{c}_n$  are even integers, then an equation with no integer solutions has been defined. Therefore, all of the combinations of the remaining variables in the ranking ( $y_k$  to  $y_n$ ) can be eliminated from consideration, given the present held values of  $y_1$  to  $y_{k-1}$ . The algorithm immediately begins to backtrack, reducing  $y_{k-1}$  by one integer amount, finding a new  $z_t$ , determining a new bound for  $y_k$ , and so on. On certain problems, several solutions can be implicitly examined and eliminated with this procedure.

During the calculation of a new  $z_t$ , the value of  $z_t$  can be driven negative. This is an immediate indication that

all of the remaining variables in the ranking can be eliminated from consideration, given the present held values of the preceding variables in the ranking. The algorithm begins to backtrack as described before.

When the truncation methods fail to eliminate a certain combination, the algorithm advances through the ranking until the variable identified as  $y_{n-1}$  is reached. Since all preceding variables in the ranking,  $y_1$  through  $y_{n-2}$ , have been set equal to one of their integer possibilities, only  $y_{n-1}$  need be assigned a value. When this is done,  $y_n$  can be calculated from the objective function equation. If the value calculated for  $y_n$  is noninteger, the algorithm begins backtracking through the combinations. If  $y_n$  is found to be integer, the feasibility of the solution

$$\underline{y} = (y_1, y_2, \dots, y_n) \quad (4-27)$$

is tested in the functional constraints. If it is infeasible, the algorithm backtracks and examines other combinations. In a finite number of iterations, the solution space will be examined explicitly and implicitly. If no integer solution is found for the first value of ZOF, then a new objective value is determined from

$$ZOF = ZOF - g. \quad (4-28)$$

This will assure that the new value of ZOF has  $g|ZOF$ . Only objective function values such that  $g|ZOF$  need be considered as shown in the theory of linear Diophantine equations.

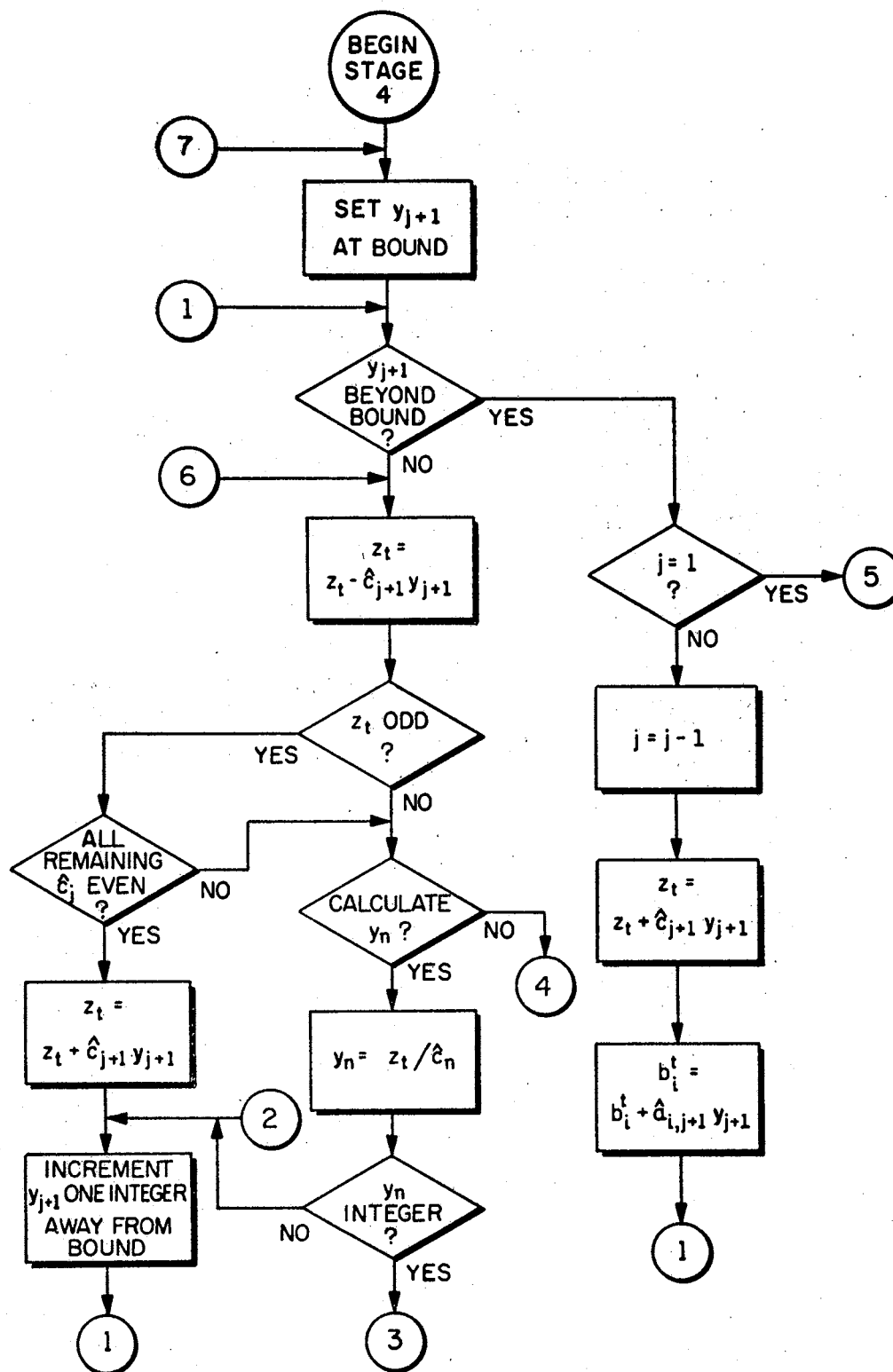
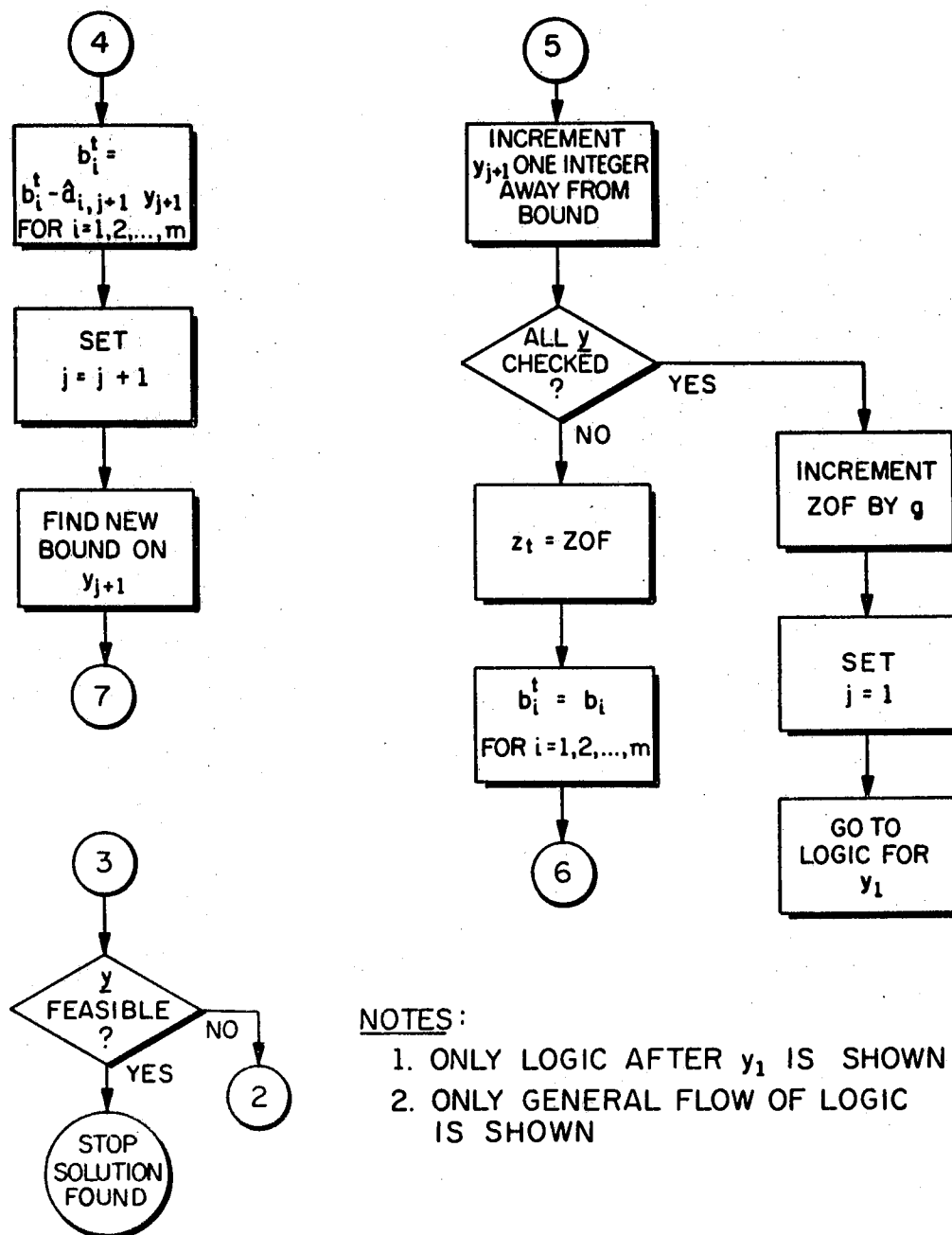


Figure 7. Stage 4 Logic Flow Diagram

NOTES:

1. ONLY LOGIC AFTER  $y_1$  IS SHOWN
2. ONLY GENERAL FLOW OF LOGIC IS SHOWN

Figure 7. (Continued)

If the solution  $\underline{y}$  in Equation (4-27) is found to be feasible, the algorithm stops and indicates the optimal solution has been found. Since the optimal solution must be the extreme point of the set feasible solutions, the first feasible solution found on the family of parallel objective hyperplanes is the optimal solution.

The logic of Stage 4 is shown in Figure 7. A three variable example will be demonstrated in the next section to further explain the concepts of the algorithm.

#### A Three Variable Example

The three variable example presented in this section will be used to demonstrate several of the characteristics of the objective function reduction algorithm. The logic of the algorithm is described in Appendix A. This appendix should be used as a reference while following the step-by-step solution of the example problem. The following example will be used in this section:

$$\text{maximize } z = 6x_1 + 3x_2 + 6x_3 \quad (4-29)$$

$$\text{subject to } -4x_1 + 5x_2 + 2x_3 \leq 4 \quad (4-30)$$

$$-2x_1 + 5x_2 + 0x_3 \leq 5 \quad (4-31)$$

$$3x_1 - 2x_2 + 2x_3 \leq 6 \quad (4-32)$$

$$2x_1 - 5x_2 + 0x_3 \leq 1 \quad (4-33)$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3 \quad (4-34)$$

$$x_j, c_j \text{ INTEGER for } j = 1, 2, 3. \quad (4-35)$$

The solution to this problem with the objective function reduction algorithm begins with Stage 1.

### Stage 1

#### Step 1

Step 1 asks for the simplex linear programming solution to the problem. Using a simple computer code, the solution was found to be

$$ZSIM = 29.2 \quad (4-36)$$

and

$$\underline{x}^* = (3.64, 2.45, 0.0). \quad (4-37)$$

Since this solution is not all integer, the algorithm goes to Step 2.

#### Step 2

The upper and lower integer bounds on each variable,  $x_j$ ,  $j = 1, 2, 3$ , must now be defined. One way of doing this is to use the procedure of solving six linear programming problems. The first three problems will have objective functions of the form

$$\text{maximize} \quad z = x_j \quad (4-38)$$

for  $j = 1, 2, 3$ . The constraints associated with the objective functions of Equation (4-38) are the original problem constraints of Equations (4-30) through (4-34). These three

problems provide upper bounds on each variable. Similarly, three minimization problems with objective functions of the form

$$\text{minimize } z = x_j \quad (4-39)$$

for  $j = 1, 2, 3$ , can be solved to obtain lower bounds on each variable. When this is done, then

$$x_j^u = [x_j^*] \quad (4-40)$$

for  $j = 1, 2, 3$ , where  $x_j^*$  is the value obtained from maximizing the  $j^{\text{th}}$  variable subject to the problem constraints. The lower integer bounds are

$$x_j^l = \langle x_j^* \rangle \quad (4-41)$$

for  $j = 1, 2, 3$ , where  $x_j^*$  is the value found from minimizing the  $j^{\text{th}}$  variable subject to the problem constraints. Using this method, it was found that

$$x_1^u = 3 \quad (4-42)$$

$$x_2^u = 2 \quad (4-43)$$

$$x_3^u = 2 \quad (4-44)$$

and

$$x_j^l = 0 \quad \text{for } j = 1, 2, 3. \quad (4-45)$$

### Step 3

Since the problem is to be maximized, the key word MAX

is set to

$$\text{MAX} = 1. \quad (4-46)$$

## Stage 2

### Step 1

Using Euclid's algorithm or by inspection the greatest common divisor of the objective function coefficients is

$$(6, 6, 3) = 3 \quad (4-47)$$

or,

$$g = 3. \quad (4-48)$$

### Step 2

Now, the initial objective function value must be found. Stage 1 says  $\text{ZSIM} = 29.2$ . Since  $g \neq 1$ ,

$$\text{ZOF} = [\text{ZSIM}] \quad (4-49)$$

$$\text{ZOF} = [29.2] \quad (4-50)$$

$$\text{ZOF} = 29 \quad (4-51)$$

will not lead to an integer solution to the objective function of Equation (4-29). The first integer value below 29 that has  $g = 3$  as a factor is 27. Therefore, with

$$\text{ZOF} = 27 \quad (4-52)$$

the requirement of  $g \mid \text{ZOF}$  is met.



Stage 3Step 1

In Stage 3, the variables are ranked according to the number of integer values they can take on. Using the bounds found in Stage 1, the range size,  $r(x)_j$ , for  $j = 1, 2, 3$ , can be determined. In general, the variables are bounded such that

$$x_j^l \leq x_j \leq x_j^u \quad (4-53)$$

for  $j = 1, 2, \dots, n$ . In this example, the bounds are

$$0 \leq x_1 \leq 3 \quad (4-54)$$

$$0 \leq x_2 \leq 2 \quad (4-55)$$

$$0 \leq x_3 \leq 2. \quad (4-56)$$

Therefore, the range size for each variable is

$$r(x)_1 = 4 \quad (4-57)$$

$$r(x)_2 = 3 \quad (4-58)$$

$$r(x)_3 = 3. \quad (4-59)$$

Although not described in Appendix A or used in the computer code, an interesting and useful heuristic procedure will be pointed out. Given the range sizes are approximately the same, rank the variables so the even objective function coefficients appear last in the ranking. This leads to

$$y_1 = x_2 \quad (4-60)$$

$$y_2 = x_3 \quad (4-61)$$

$$y_3 = x_1 \quad (4-62)$$

as a ranking. This will allow the methods of linear Diophantine equations to be used in truncating part of the search. Also, this implies the bounds in terms of the ranked variables are

$$y_1^u = 2 \quad (4-63)$$

$$y_2^u = 2 \quad (4-64)$$

$$y_3^u = 3 \quad (4-65)$$

and

$$y_j^l = 0 \quad (4-66)$$

for  $j = 1, 2, 3$ .

Notice that using the methods of Appendix A would lead to ranking of  $y_1 = x_3$ ,  $y_2 = x_2$ ,  $y_3 = x_1$ . Unless altered by using the heuristic procedure of ranking, the algorithm would select this ranking scheme. In the special case where the  $r(x)_j$  are approximately the same and some  $c_j$  are even integers, a modified ranking as in Equations (4-60) through (4-62) often speeds convergence.

### Step 2

With the ranking proposed in Equations (4-60) through (4-62), a successive sequence of even objective function coefficients is described. The ordered coefficients are

$$\hat{c}_1 = 3 \quad (4-67)$$

$$\hat{c}_2 = 6 \quad (4-68)$$

$$\hat{c}_3 = 6, \quad (4-69)$$

which has the last two coefficients in the ranking even integers. The key word IFLAG is set equal to the number of the ranking position where the even coefficients begin. That is, all coefficients must be even integers from  $k$  to  $n$  in the set  $\{\hat{c}_j | j = 1, 2, \dots, k, \dots, n\}$ . Therefore, set

$$IFLAG = 2. \quad (4-70)$$

### Step 3

During Stage 4, tighter bounds will be sought for the variables. Temporary bounds will be established and modified at various points in the algorithm. The temporary bounds are first set equal to the over-all bounds for each variable. Using the change of variables and the bounds found earlier, set  $y_{(j)t}^u = y_j^u$  and  $y_{(j)t}^l = y_j^l$  for  $j = 1, 2, \dots, n$ . This implies

$$y_{(1)t}^u = 2 \quad (4-71)$$

$$y_{(2)t}^u = 2 \quad (4-72)$$

$$y_{(3)t}^u = 3 \quad (4-73)$$

and

$$y_{(j)}^l = 0 \quad (4-74)$$

for  $j = 1, 2, 3$ .

Before going to Stage 4, the example problem will be stated in terms of the ranked variables,  $y_j$ ,  $j = 1, 2, 3$ .

$$\text{maximize} \quad z = 3 y_1 + 6 y_2 + 6 y_3 \quad (4-75)$$

$$\text{subject to} \quad 5 y_1 + 2 y_2 - 4 y_3 \leq 4 \quad (4-76)$$

$$5 y_1 + 0 y_2 - 2 y_3 \leq 5 \quad (4-77)$$

$$- 2 y_1 + 2 y_2 + 3 y_3 \leq 6 \quad (4-78)$$

$$- 5 y_1 + 0 y_2 + 2 y_3 \leq 1 \quad (4-79)$$

$$y_j \geq 0 \text{ for } j = 1, 2, 3 \quad (4-80)$$

$$y_j, \hat{c}_j \text{ INTEGER for } j = 1, 2, 3. \quad (4-81)$$

#### Stage 4

##### Step 1

Set  $j = 1$ .

##### Step 2

Since  $\text{MAX} = 1$ , set

$$y_1 = y_1^u = 2. \quad (4-82)$$

A heuristic procedure is to set  $y_1 = y_1^l$  if  $\hat{c}_1$  is negative.

##### Step 3

Set  $\text{ZSUBT} = 27$ ,  $\text{NOTE4} = 0$ , and

$$b_1^t = b_1 = 4 \quad (4-83)$$

$$b_2^t = b_2 = 5 \quad (4-84)$$

$$b_3^t = b_3 = 6 \quad (4-85)$$

$$b_4^t = b_4 = 1. \quad (4-86)$$

The key word NOTE<sup>4</sup> is used in the computer code and the written form of the algorithm as an indicator which equals zero when all right-hand side values of the constraints are positive or zero. It is set equal to one when a right-hand side value has been forced to a negative value.

#### Step 4

Since this is the first time a ZSUBT value has been found that is tighter than the ZOF = 27 value, the following equation is used

$$ZSUBT = ZSUBT - \hat{c}_1 y_1. \quad (4-87)$$

Therefore, set

$$ZSUBT = 27 - (3)(2) = 21. \quad (4-88)$$

This ZSUBT value is  $\geq 0$ , so go to Step 5.

#### Step 5

ZSUBT = 21 is an odd integer, so go to Step 6.

#### Step 6

The key word IFLAG = 2 indicates all successive ranked variables from 2 through n have even objective function coefficients. Therefore, truncate the search and do not consider any further solutions with  $y_1 = 2$ , because none can

produce an integer solution. Go to Step 26, since IFLAG = 2 and  $j = 1$ .

### Step 26

With MAX = 1, set

$$y_1 = y_1 - 1 = 2 - 1 = 1 \quad (4-89)$$

and go to Step 27.

### Step 27

Here, a check is made to see if all solutions for this particular objective function plane have been considered. This would be the case if the algorithm had incremented  $y_1$  below its temporary lower bound. But,  $y_1 = 1$  and  $y_{(1)t}^l = 0$  which says

$$y_1 > y_{(1)t}^l. \quad (4-90)$$

Therefore, set ZSUBT = ZOF; that is,

$$\text{ZSUBT} = 27 \quad (4-91)$$

and go to Step 28.

### Step 28

The value of  $n \neq 2$ , so set

$$b_1^t = b_1 = 4 \quad (4-92)$$

$$b_2^t = b_2 = 5 \quad (4-93)$$

$$b_3^t = b_3 = 6 \quad (4-94)$$

$$b_4^t = b_4 = 1 \quad (4-95)$$

and, go to Step 4.

#### Step 4

Again, as far as the algorithm is concerned, this is the first time a ZSUBT value has been found that is tighter than  $ZOF = 27$ . Therefore, set

$$ZSUBT = ZSUBT - \hat{c}_1 y_1 \quad (4-96)$$

$$ZSUBT = 27 - (3)(1) = 24 \quad (4-97)$$

and,  $ZSUBT \geq 0$ , go to Step 5.

#### Step 5

ZSUBT is an even integer. Go to Step 7.

#### Step 7

This step checks to see if the final variable,  $y_n$ , is to be calculated yet. Since  $n - 1 = 2$  and  $j + 1 = 2$ , therefore

$$n - 1 = j + 1;$$

so the algorithm says go to Step 13, because new right-hand side values have not been calculated for the held value of  $y_1$ , yet.

#### Step 13

The new right-hand side b-values for the constraints

are calculated from

$$b_i^t = b_i^t - \hat{a}_{i1} y_1 \quad (4-98)$$

for  $i = 1, 2, \dots, m$ . This leads to

$$b_1^t = 4 - (5)(1) = -1 \quad (4-99)$$

$$b_2^t = 5 - (5)(1) = 0 \quad (4-100)$$

$$b_3^t = 6 - (-2)(1) = 8 \quad (4-101)$$

$$b_4^t = 1 - (-5)(1) = 6 \quad (4-102)$$

A right-hand side b-value,  $b_1^t$ , has been forced negative.

Therefore, set

$$\text{NOTE4} = 1,$$

implying the methods of Step 18 cannot be used to find tighter bounds on  $y_{j+1}$ . Go to Step 14.

#### Step 14

The subscript  $j = 1$  and tighter bounds have not been found on  $y_{j+1}$ . Go to Step 15.

#### Step 15

The subscript  $j + 1 = 2$ , so go to Step 16.

#### Step 16

Since  $\text{NOTE4} = 1$ , go to Step 17.



Step 17

The key word MAX equals one, so  $y(2)_t^u = y_2^u$  gives

$$y(2)_t^u = 2. \quad (4-103)$$

Go to Step 23.

Step 23

The next variable in the ranking,  $y_{j+1}$ , is set at its bound. With MAX = 1, set

$$y_{j+1} = y(j+1)_t^u \quad (4-104)$$

$$y_2 = y(2)_t^u = 2, \quad (4-105)$$

and go to Step 24.

Step 24

Variable  $y_{j+1}$  has not been incremented below its bound, such that

$$y_{j+1} \geq y(j+1)_t^l. \quad (4-106)$$

Therefore, go to Step 4.

Step 4

This time a ZSUBT has been calculated before, so the new ZSUBT is found from

$$\text{ZSUBT} = \text{ZSUBT} - \hat{c}_{j+1} y_{j+1} \quad (4-107)$$

$$\text{ZSUBT} = 24 - (6)(2) = 12. \quad (4-108)$$

Since  $ZSUBT \geq 0$ , go to Step 5.

#### Step 5

$ZSUBT$  is an even integer. Go to Step 7.

#### Step 7

Now, the final variable is to be calculated,  $n - 1 = j + 1$ , and right-hand side  $b$ -values have been found for  $y_1$ . Go to Step 8.

#### Step 8

Check to see if  $y_n$  is an integer from

$$y_n = ZSUBT / \hat{c}_n \quad (4-109)$$

$$y_n = 12/6 = 2. \quad (4-110)$$

The variable  $y_n$  is integer so go to Step 9.

#### Step 9

Check to see if  $y_n > y_n^u$ . The value of  $y_3^u = 3$ , so  $y_3 < y_3^u$ . Go to Step 11.

#### Step 11

The feasibility of the solution

$$\underline{y} = (1, 2, 2) \quad (4-111)$$

is now tested in the functional constraints, Equations (4-76) through (4-79). This solution is found to be

infeasible in the third constraint, Equation (4-78). Go to Step 12.

### Step 12

A solution has been found infeasible, so  $y_{j+1} = y_2$  is incremented down one integer. With  $\text{MAX} = 1$ ,

$$y_{j+1} = y_{j+1} - 1 \quad (4-112)$$

$$y_2 = y_2 - 1 \quad (4-113)$$

$$y_2 = 2 - 1 = 1. \quad (4-114)$$

The algorithm continues attempting to find tighter bounds and searching for a feasible solution on the plane

$$z = 3 y_1 + 6 y_2 + 6 y_3 = 27. \quad (4-115)$$

No feasible solution is found on this plane. After several steps similar to the ones described above, the algorithm reaches Step 27 with  $y_1$  incremented to a value of  $y_1 = -1$ . This is below  $y_1^{\ell}$  which indicates all solutions on the plane of Equation (4-115) have been implicitly or explicitly considered. The objective function is incremented down by the amount of the greatest common divisor to get

$$\text{ZOF} = \text{ZOF} - g \quad (4-116)$$

$$\text{ZOF} = 27 - 3 = 24. \quad (4-117)$$

Now, the plane

$$z = 3 y_1 + 6 y_2 + 6 y_3 = 24 \quad (4-118)$$

will be searched for a feasible solution with the methods of Stage 4. A feasible solution, therefore, the optimal solution, is found on this plane, such that

$$\underline{y}^* = (2, 0, 3). \quad (4-119)$$

The optimal solution in terms of the ranked variables can be changed to the original variables to give

$$\underline{x}^* = (3, 2, 0). \quad (4-120)$$

This example demonstrates many of the essential features of the objective function reduction algorithm.

Chapter V will discuss the computer code and some of the implications of the algorithm.

## CHAPTER V

### SOME IMPLICATIONS OF THE ALGORITHM

This chapter discusses some of the implications of the algorithm developed in this research. The computer code that performs the step-by-step process of the objective function reduction algorithm is described. The proposed method of ranking the problem variables is discussed more fully. Because of the importance of finding good approximate solutions to the pure integer linear programming problem, a heuristic procedure for establishing a lower bound on the maximand is presented. First, the computer code will be considered.

#### The Computer Code

The computer program used for the objective function reduction algorithm is written in the FORTRAN IV language. The code is composed of a main driver test program and three subroutines. The main driver test program is shown in Appendix B. This main program performs several functions, the first of which is initialization of certain parameters. Some parameters are set to their initial or normal values early in the main program. Later, tests are made to see if violations of limits or anticipated errors have been found

as the program searches for the solution.

The main driver test program reads in the values that describe the integer linear program to be solved. These include the number of constraints, the number of real variables, whether the problem is maximization or minimization, the constraint coefficients, the right-hand side b-values, and the objective function coefficients. Also, the Stage 1 information of the upper and lower integer bounds on the variables and the objective function value at the continuous variable solution are read into computer memory. For reference and analysis, the main program writes out the input data.

In Appendix C, SUBROUTINE GCD is presented. This subroutine calculates the greatest common divisor for a set of objective function coefficients. The greatest common divisor is calculated using Euclid's algorithm that was described in Chapter III. In addition, this subroutine calculates the first objective function value that is searched while looking for the optimal integer solution. Since it must have the greatest common divisor as a factor, the initial objective function value is calculated after the greatest common divisor has been determined.

The FORTRAN IV code for SUBROUTINE RANK is listed in Appendix D. This subroutine ranks the variables in the objective function according to their range of possible values. The ranking procedure was demonstrated and explained in Chapter IV, but will be discussed again later

in this chapter. The subroutine first calculates the range of possible values that each real variable can take on. Using a Shell sorting technique, the variable with the smallest range size is ranked first in the ranking. The next part of the subroutine tests for the situation that some variables have equal range sizes in the ranking. When this occurs, the variable with the larger objective function coefficient is given the ranking position nearest the variable ranked first. Also, this subroutine calculates the value for the even sequence indicator, IFLAG. The subroutine scans the objective function coefficients of the ranked variables, looking for an even sequence of coefficients from  $\hat{c}_k$  to  $\hat{c}_n$ . If there is no such even sequence, the program defaults to setting IFLAG at one greater than the number of real variables. This indicates to the program that no even sequence exists.

The main part of the explicit and implicit search for the optimal integer solution is conducted in SUBROUTINE SEARCH, which is reproduced in Appendix E. This subroutine takes the information from the main program and subroutines RANK and GCD as it begins examining the solution space. The combinatorial search starts by holding the highest ranked variable at its upper limit for a maximization problem. SUBROUTINE SEARCH corresponds directly to Stage 4 of the analysis presented in Chapter IV and Appendix A. The steps of Stage 4 in Appendix A provide a direct reference to the step-by-step analysis of SUBROUTINE SEARCH.

Several integer linear programming test problems that have been solved by the computer code are collected in Appendix F. Many of the problems are examples used in the literature. The majority of the examples are small in size, but they have sufficient variety to test the computer code. The computer code and all test problems were run on an IBM System/360 Model 65 computer. The method of ranking the variables used in the objective function reduction algorithm will now be discussed.

#### The Scheme of Ranking the Variables

The objective function reduction algorithm searches for the optimal integer solution by considering the feasible integer combinatorial possibilities. Any practical algorithm must examine the solution space in such a manner that many solutions need not be explicitly considered. One technique used in the algorithm presented in this thesis is a ranking scheme for ordering the variables. The objective of ordering the variables is to develop a procedure that will implicitly examine and eliminate several combinatorial solution possibilities. Appendix A gives a written description of the ranking procedure used in the algorithm. In Appendix D, SUBROUTINE RANK shows how the computer code ranks the variables.

Briefly, the ranking scheme will be reviewed again. The integer bounds on each variable are examined and used to



determine the range of possible integer values for each variable. The range size is the number of integer values a particular variable can take on within the feasible solution space. From the set of problem variables, a permutation is developed such that the range size of the variables,  $r(x)_j$ , will proceed from smallest to largest. The ranked variables are assigned a new symbol,  $y_j$ , for  $j = 1, 2, \dots, n$ .

When the range size is the same for two or more variables, the tie for ranking position is broken by selecting the variable  $(x_j)$  with the largest  $c_j$  to have the ranking position nearest  $y_1$ . If two or more variables have the same range size and the same objective function coefficients, the tie for ranking position is broken arbitrarily.

The canonical form of a maximization problem will be assumed throughout the remainder of this discussion. The algorithm begins its search for the optimal solution by setting the variable ranked as  $y_1$  at its upper limit. This value of  $y_1$  is set at

$$y_1 = y_1^u, \quad (5-1)$$

the upper limit on the first variable in the ranking. From this held value of  $y_1$ , a new, reduced temporary objective function value can be found such that

$$\hat{c}_2 y_2 + \hat{c}_3 y_3 + \dots + \hat{c}_n y_n = z_t - \hat{c}_1 y_1, \quad (5-2)$$

if  $\hat{c}_1 > 0$ . As described earlier, this process of tightening the value of  $z_t$  for the held values of variables continues

until the algorithm can truncate the search or  $y_{n-1}$  is reached. If the algorithm must proceed through the combinatorial possibilities to  $y_{n-1}$  without truncating the search, the ranking scheme does not increase the speed of the algorithm. When a method can be used to implicitly examine the remainder of the combinatorial possibilities, the ranking method is beneficial in eliminating solution combinations.

Table I shows the feasible integer values of each variable in an example with 2535 feasible integer combinations. The variables have been ranked according to their range size, such that,  $r(y)_1 \leq r(y)_2 \leq r(y)_3 \leq r(y)_4$ .

TABLE I  
AN EXAMPLE WITH THE VARIABLES RANKED  
ACCORDING TO RANGE SIZE

$y_1$	$y_2$	$y_3$	$y_4$
2	4	12	12
1	3	11	11
0	2	10	10
	1	9	9
	0	8	8
		7	7
		6	6
		5	5
		4	4
		3	3
		2	2
		1	1
		0	0

The preceding table shows the advantage the ranking scheme offers when a specific situation is considered. Suppose the algorithm sets  $y_1$  at one of its feasible values. If the algorithm now finds it can truncate the search, the pyramid effect of the ranking scheme provides a maximum of implied examination of the solution space.

The ranked variables  $y_3$  and  $y_4$  in Table I both have the same range size,

$$r(y)_3 = r(y)_4 = 13. \quad (5-3)$$

In the particular problem selected as an example for Table I, the associated objective function coefficients are  $\hat{c}_3 = 7$  and  $\hat{c}_4 = 3$ . When the original problem variables ( $x_j$ ) were being examined to determine ranking positions, it was found that two variables had the same range size. The tie for which variable should be given ranking position  $y_3$  was broken by assigning the variable with the greater  $\hat{c}_j$  the  $y_3$  position. The reason for selecting that variable is because it potentially allows the algorithm to take advantage of one of its truncation methods. If the algorithm ever causes the temporary  $z_t$  value to be driven negative, it can be immediately implied that all succeeding combinations of ranked variables are infeasible for the held value of all preceding variables. Therefore, the ranking scheme should maximize the feature whenever possible. By selecting the variable with the largest coefficient, the value of  $z_t$  can be reduced most quickly. This may lead to truncation early

in the search.

In certain problems, it may be beneficial to override the ranking scheme used in SUBROUTINE RANK. A heuristic procedure has been developed that reduces the number of iterations on certain types of problems. When the range sizes on the ranked variables are approximately the same, the number of iterations can often be reduced by modifying the ranking so any variables with even  $\hat{c}_j$  appear last in the ranking. After the variables have been ranked according to their range size, the ranking is changed so any variables with even objective function coefficients appear at the end of the ordered variables. This will allow the algorithm to set the even sequence indicator, IFLAG, at a value that will increase the number of variables truncated. The newly formed sequence of even objective function coefficients allows the algorithm to truncate the search each time  $z_t$  is found to be an odd integer, when only the even sequence of coefficients is being considered. This ranking modification procedure is only beneficial when the variable range sizes are approximately the same and some of the objective function coefficients are even integers.

#### Modifying the Coefficients in the Objective Function

In many integer linear programming problems, the size or configuration of the solution space make finding the optimal solution difficult, even with high speed computing

equipment. These situations can make the computing time impractical for finding the optimal integer solution. A good approximate solution can be of great value when the optimal solution cannot be easily obtained. Hillier (14) has pointed out this fact in his work in developing an efficient heuristic procedure for integer linear programming.

The objective function reduction algorithm has some features that allow it to converge most rapidly on some types of problems. Conversely, convergence speed is limited when certain situations exist. In the case where the greatest common divisor ( $g$ ) of the objective function coefficients is one, the algorithm must search each succeeding hyperplane beyond the continuous variable solution. Also, if each objective function coefficient is an odd integer, the even sequence indicator, IFLAG, can never be used to truncate the search and implicitly examine some combinatorial possibilities. As with other linear programming algorithms, the objective function reduction algorithm converges increasingly slower as the number of variables and constraints enlarges. A heuristic procedure will now be discussed that will modify the objective function coefficients to take advantage of some of the algorithm's methods of speeding convergence. The canonical form of a maximization problem will be assumed throughout the remainder of this discussion.

When the algorithm has determined that the greatest

common divisor ( $g$ ) is greater than one, only certain hyperplanes are considered in the search for the optimal solution. As previously mentioned, for an integer solution to be possible,  $g$  must be a factor of the value of the particular hyperplane being considered. Using the earlier notation,  $g|k$  must be true for an integer solution to be possible in the objective function

$$\hat{c}_1 y_1 + \hat{c}_2 y_2 + \dots + \hat{c}_n y_n = k, \quad (5-4)$$

where  $\hat{c}_j$ , for  $j = 1, 2, \dots, n$ , and  $k$  are integers. Using this concept, an approximate solution procedure that modifies the objective function coefficients to get  $g > 1$  can potentially reduce the number of hyperplanes the algorithm must examine.

Consider an objective function of the form

$$z = 501 x_1 + 98 x_2 - 296 x_3 + 705 x_4. \quad (5-5)$$

The objective function reduction algorithm would find that the greatest common divisor is one. With small changes in each coefficient, a new, approximate objective function could have the form

$$z = 500 x_1 + 100 x_2 - 300 x_3 + 700 x_4. \quad (5-6)$$

The algorithm would now conclude that  $g = 100$  should be used for the greatest common divisor. As described in Chapter III on linear Diophantine equations, objective function values in increments of  $g$  are the only ones that need be

searched for an integer solution. Again, this comes from the fact that the greatest common divisor must be a factor of the objective function value. The number of hyperplanes the algorithm must search has been significantly reduced. Also, Theorem 3.3 in Chapter III can be used to set a lower bound on the search. For a feasible integer solution to exist, the value of the objective function must be greater-than-or-equal-to  $g$ .

Another consideration would be to try to modify the objective function coefficients to obtain a sequence of even integers from  $k$  to  $n$  in the set  $\{\hat{c}_j \mid j = 1, 2, \dots, k, \dots, n\}$ . This will allow the algorithm to use the even sequence indicator, IFLAG, to truncate the search. The number of iterations can be reduced appreciably when IFLAG can be used to implicitly examine some solution combinations.

Finally, modifying the objective function to obtain an approximate solution offers still other advantages. Many primal integer programming algorithms benefit from having a good approximate solution to begin their search. An approximate solution that is obtained rapidly can reduce the computing time for many optimal algorithms. Also, this approximate solution can be used to describe a lower bound on the maximand.

Using the techniques described above for modifying the objective function coefficients, an approximate solution can be obtained with methods that speed the algorithm. Tests on several example problems have shown that minor changes in

objective function coefficients can be made and still lead to the optimal solution. Obviously, a modified objective function gives no direct assurance the optimal solution has been found. Nevertheless, the solution found with the modified objective function,

$$\underline{y} = (y_1, y_2, \dots, y_n), \quad (5-7)$$

is potentially useful. Even if the objective function slope is changed sufficiently so that  $\underline{y} \neq \underline{y}^*$ , the  $\underline{y}$  solution can be used to set a lower bound on the maximand. If the solution  $\underline{y}$  is substituted in the unmodified objective function,

$$z = \hat{c}_1 y_1 + \hat{c}_2 y_2 + \dots + \hat{c}_n y_n, \quad (5-8)$$

then a new constraint can be formed. If the value of Equation (5-8) with the approximate solution substituted in for  $y_j$ ,  $j = 1, 2, \dots, n$  is called  $k$ , then a new lower bound can be made with the constraint

$$\hat{c}_1 y_1 + \hat{c}_2 y_2 + \dots + \hat{c}_n y_n \geq k. \quad (5-9)$$

The new constraint defined in Equation (5-9) can be added to the original problem constraints to describe a new integer linear programming problem. The solution space has been reduced and some solutions eliminated from further consideration. This approach offers an approximate solution and tighter bound on the optimal feasible solution.

The next and final chapter of this thesis will briefly



state the problem, summarize the important findings, and suggest areas of further research, investigation, and study.

## CHAPTER VI

### SUMMARY AND CONCLUSIONS

The primary problem considered in this research is an extension of the existing theory of solution procedures for pure integer linear programming. The objective is to provide a new algorithm for solving the pure integer linear programming problem. Secondary problems approached are (1) to identify any heuristic procedures that will speed the convergence of the algorithm, (2) develop a procedure for finding a good approximate solution to the problem, (3) write a computer code to evaluate the algorithm.

#### Important Findings

Several important techniques for integer linear programming have been identified in this research. A new algorithm has been developed. The objective function reduction algorithm presented in this thesis uses a combinatorial search procedure to implicitly and explicitly search the solution space. This algorithm uses the concept of examining a family of objective function hyperplanes until an integer solution is found. Beginning at the optimum non-integer solution, the algorithm inspects parallel objective function hyperplanes in the feasible solution space.

The basic structure of the algorithm developed in this research is divided into four stages. Stage 1 identifies the optimum continuous-variable solution, defines the overall bounds on each variable, and determines whether the objective function is to be maximized or minimized. Stage 2 calculates the greatest common divisor of the objective function coefficients and determines the first hyperplane to be searched. In Stage 3, a ranking scheme is selected for the variables. It also examines the objective function coefficients of the ranked variables and defines an even sequence indicator used to truncate the search for implicit consideration of solution combinations. Stage 4, the main section of the algorithm, carries out the implicit and explicit enumeration of the feasible integer solution space.

The ranking scheme proposed in this research shows how the variables can be ordered to potentially eliminate many combinatorial solution possibilities from explicit consideration. The ranking method examines the range of possible integer values for each variable. A range size is defined as the number of integer possibilities for a particular variable. Using the range sizes, a permutation from the set of problem variables,  $\{x_j \mid j = 1, 2, \dots, n\}$ , is identified such that the range size of the variables proceeds from the smallest to the largest. The search of the algorithm proceeds from the variables with the smallest range size to those with the largest range size. This ordering of the variables allows the algorithm to use truncation procedures

to maximize the number of solution combinations that are examined implicitly.

A heuristic method of ranking has been developed for a certain class of problems. For the situation where the variable range sizes are approximately the same and some of the objective function coefficients are even integers, a modification of the ranking scheme can often speed convergence. When the variables with even objective function coefficients are positioned last in the ranking, the truncation method of the even sequence indicator (IFLAG) can be used. Since the range sizes are approximately the same, any additional truncation method and the even sequence indicator will be used to greatest advantage.

Another important result of this research is associated with finding an approximate solution to the pure integer linear programming problem. Some of the concepts of linear Diophantine equations allow the algorithm to implicitly consider certain combinatorial solution possibilities. In order to take advantage of these concepts, the objective function coefficients can be modified to produce a new, approximate objective function that can be handled more rapidly by the algorithm. This procedure can be used to establish a lower bound on the maximand, as described in Chapter V. The lower bound can potentially be used to tighten the solution space.

The computer code in Appendixes B, C, D, and E is a valuable tool for further evaluation of the algorithm. It

is a practical necessity for solving problems of moderate or large size. The computer code can be used to experiment with refinements and additions to the algorithm.

#### Areas for Further Investigations

During this research into integer linear programming, some topics were found for future study and investigation. First, a review of the literature identified the need for a thorough survey of the recent literature. From the late 1950's through the middle of the 1960's, surveys such as those by Balinski (3) and Beale (4) adequately describe the work in integer linear programming. Nevertheless, in recent years no comprehensive survey has appeared to unify and update this area of study. It is needed and, hopefully, this void in the literature will be filled soon.

As a companion of a survey of the recent literature, additional work should be published evaluating the recent integer programming computer codes. Although some of the published literature does list experimental results with individual computer codes, no computational efficiency survey such as the 1967 work of Trauth and Woolsey (28) has appeared recently. From a practical point of view, the ability of existing computer codes to solve problems is very important. Further investigations are needed.

Additional testing of the algorithm developed in this research should be considered. Larger, more difficult problems offer a severe test to any integer programming

algorithm. These problems can consume an enormous amount of computer time and should be studied and evaluated carefully. The objective function reduction algorithm should be compared for computational efficiency with other procedures.

Further investigation of several heuristic procedures associated with this research should be evaluated and refined. Some of these techniques have the potential of becoming an integral part of the algorithm. For example, the heuristic ranking scheme that can be used when the range sizes are approximately the same offers the possibility of being quantified. The heuristic procedures for obtaining a good approximate solution can be developed further. The speed and efficiency of these concepts should be considered.

The study of other problems in integer programming has become an area of increasing interest in recent years. Some of the concepts of this thesis may suggest a new approach to a mixed-integer programming algorithm. Also, further research could consider the integer nonlinear programming problem and see if some of the theory of nonlinear Diophantine equations can be used.

### Conclusions

Many of the conclusions of this research are discussed and analyzed in the earlier chapters of this thesis. The dominant conclusions are as follows:

- (1) A new algorithm for integer linear programming

has been developed. This algorithm uses a combinatorial search procedure to implicitly and explicitly search the solution space.

- (2) Some heuristic procedures have been identified that speed the convergence of the algorithm. A technique for ranking the variables in certain classes of problems has been developed.
- (3) A ranking scheme for the variables has been defined as a part of the algorithm. This ranking method produces an ordered set of variables to potentially eliminate many combinatorial solution possibilities.
- (4) A procedure for finding a good approximate solution has been outlined. Also, this technique can be used to establish a lower bound on the maximand.
- (5) Finally, a computer code has been written to provide an additional method of evaluating the algorithm. Also, it can be used to test modifications and refinements of the algorithm.

The area of integer linear programming cannot be considered complete. No practical algorithm comparable in efficiency to the simplex method has yet been discovered. This research has continued the search for such an algorithm and has proposed some new, useful techniques for solving the integer linear programming problem.

## SELECTED BIBLIOGRAPHY

- (1) Agin, Norman. "Optimum Seeking Methods With Branch and Bound." Management Science, Vol. 13, No. 4 (December, 1966).
- (2) Balas, Egon. "A Note on the Branch-and-Bound Principle." Operations Research, Vol. 16 (1968), 442-445.
- (3) Balinski, M. L. "Integer Programming: Methods, Uses, Computation." Management Science, Vol. 12 (1965), 253-313.
- (4) Beale, E. M. L. "Survey of Integer Programming." Operational Research Quarterly, Vol. 16 (1965), 219-228.
- (5) Cook, R. "An Algorithm for Integer Linear Programming." (Unpub. Doctoral Dissertation, Sever Institute of Technology, Washington University, 1966.)
- (6) Forsythe, G. E., and C. B. Moler. Computer Solution of Linear Algebraic Systems. Englewood Cliffs: Prentice-Hall, 1967.
- (7) Geoffrion, A. M. "An Improved Enumeration Approach for Integer Programming." Operations Research, Vol. 17 (1969), 437-454.
- (8) Glover, Fred. "A New Foundation for a Simplified Primal Integer Programming Algorithm." Operations Research, Vol. 16 (1968), 727-740.
- (9) Glover, Fred. "A Note on Linear Programming and Integer Feasibility." Operations Research, Vol. 16 (1968), 1212-1216.
- (10) Gomory, Ralph E. "All-Integer Integer Programming Algorithm." Research Report RC-189. Yorktown Heights: IBM Corp., 1960.
- (11) Gomory, Ralph E., and William J. Baumol. "Integer Programming and Pricing." Econometrica, Vol. 28 (1960), 521-550.



- (12) Gomory, Ralph E., and Ellis L. Johnson. "Some Continuous Functions Related to Corner Polyhedra." Research Report RC-3311. Yorktown Heights: IBM Corp., 1971.
- (13) Graves, G., and A. Whinston. "A New Approach to Discrete Mathematical Programming." Management Science, Vol. 15, No. 3 (1968), 177-190.
- (14) Hillier, Frederick S. "Efficient Heuristic Procedures for Integer Linear Programming With an Interior." Operations Research, Vol. 17 (1969), 600-637.
- (15) Hillier, Frederick S. "A Bound-and-Scan Algorithm for Pure Integer Linear Programming With General Variables." Operations Research, Vol. 17 (1969), 638-679.
- (16) Hillier, Frederick S., and Gerald J. Lieberman. Introduction to Operations Research. San Francisco: Holden-Day, 1969.
- (17) Krolak, Patrick D. "The Bounded Variable Algorithm for Solving Integer Linear Programming Problems." (Unpub. Doctoral Dissertation, Sever Institute of Technology, Washington University, 1968.)
- (18) Krolak, Patrick D. "Computational Results of an Integer Programming Algorithm." Operations Research, Vol. 17 (1969), 743-749.
- (19) Kunzi, H. P., H. G. Tzsach, and C. A. Zehnder. Translated by Werner C. Rheinboldt and Cornelia J. Rheinboldt. Numerical Methods of Mathematical Optimization. New York: Academic Press, 1971.
- (20) Land, A. H., and A. G. Doig. "An Automatic Method of Solving Discrete Programming Problems." Econometrica, Vol. 28 (1960), 497-520.
- (21) Lawler, E. L., and D. E. Wood. "Branch-and-Bound Methods: A Survey." Operations Research, Vol. 16 (1966), 699-719.
- (22) McCracken, D. D., and W. S. Dorn. Numerical Methods and Fortran Programming. New York: John Wiley and Sons, 1964.
- (23) Mitten, L. G. "Branch-and-Bound Methods: General Formulation and Properties." Operations Research, Vol. 18 (1970), 24-34.

- (24) Niven, Ivan, and H. S. Zuckerman. An Introduction to the Theory of Numbers. New York: John Wiley and Sons, 1966.
- (25) Petersen, Clifford C. "Integer Linear Programming." Journal of Industrial Engineering, Vol. 18 (1967), 456-464.
- (26) Pettofrezzo, A. J., and D. R. Byrkit. Elements of Number Theory. Englewood Cliffs: Prentice-Hall, 1970.
- (27) Senju, S., and T. Yoshiaki. "An Approach to Linear Programming With 0-1 Variables." Management Science, Vol. 15 (1968), 196-207.
- (28) Trauth, C. A. Jr., and R. E. Woolsey. "Integer Linear Programming: A Study in Computational Efficiency." Management Science, Vol. 15 (1969), 481-493.
- (29) Unger, V. E. Jr. "Capital Budgeting and Mixed Zero-One Integer Programming." A.I.I.E. Transactions, Vol. 2 (1970), 28-36.
- (30) Wagner, Harvey M. Principles of Operations Research. Englewood Cliffs: Prentice-Hall, 1969.
- (31) Young, R. D. "A Simplified Primal (all-integer) Integer Programming Algorithm." Operations Research, Vol. 16 (1968), 750-782.
- (32) Zionts, Stanley. "Toward a Unifying Theory for Integer Linear Programming." Operations Research, Vol. 17 (1969), 359-366.

## APPENDIX A

### FLOW OF LOGIC IN THE ALGORITHM

## The Objective Function Reduction Algorithm

The following statements describe the flow of logic used in the objective function reduction algorithm. The logic is divided into four stages. The first three stages are preparatory, while the Stage 4 carries out the implicit and explicit examination of the solution space.

### Stage 1

1. Find the simplex linear programming solution,  $\underline{x}$  and ZSIM.
  - a. Is the solution,  $\underline{x}$ , all integer?
    - (1) If yes, stop, solution found.
    - (2) If no, go to step 2.
2. Determine the over-all integer bound on each variable.
  - a. Set  $x_j^l$  equal to the lower integer bound for  $j = 1, 2, \dots, n$ .
  - b. Set  $x_j^u$  equal to the upper integer bound for  $j = 1, 2, \dots, n$ .
3. Set MAX = 1 for maximization or set MAX = 0 for minimization of the objective function.

Stage 2

1. Find the greatest common divisor,  $g$ , of the objective function coefficients such that

$$(c_1, c_2, \dots, c_n) = g. \quad (A-1)$$

2. Find the initial objective function value,  $ZOF$ , to be considered.
  - a. If  $MAX = 1$ , then  $ZOF = [ZSIM]$  where  $g = 1$ ; otherwise,  $ZOF$  equals the greatest integer less than  $ZSIM$  such that  $g \mid ZOF$ .
  - b. If  $MAX = 0$ , then  $ZOF = \langle ZSIM \rangle$  where  $g = 1$ ; otherwise,  $ZOF$  equals the smallest integer greater than  $ZSIM$  such that  $g \mid ZOF$ .

Stage 3

1. Rank the variables according to the number of feasible integer values they can take on.
  - a. Identify a permutation from the set of problem variables  $\{x_j \mid j = 1, 2, \dots, n\}$  such that the variable range size,  $r(x)_j$ , proceeds from smallest to largest in permutation of  $x_j$  variables. Set the ranked variables,  $y_j$ ,  $j = 1, 2, \dots, n$ , equal to the ordered  $x_j$  variables in the permutation, such that  $y_1$  equals the first ordered  $x_j$  variable,  $y_2$  equals the second ordered  $x_j$  variable, and so on, for  $j = 1, 2, \dots, n$ , giving  $r(y)_1 \leq r(y)_2 \leq \dots \leq r(y)_n$ .
  - b. Break any ties in the ranking scheme where  $r(y)_j = r(y)_{j+1}$  by selecting the variable with the largest objective function coefficient to have the  $j^{\text{th}}$  ranking position. When  $\hat{c}_j = \hat{c}_{j+1}$  and  $r(y)_j = r(y)_{j+1}$ , arbitrarily break the tie.
2. Identify any successive sequence of objective function coefficients corresponding to the ordered set  $\{y_j \mid j = 1, 2, \dots, n\}$ , such that all coefficients are even integers from  $k$  to  $n$  in the set  $\{\hat{c}_j \mid j = 1, 2, \dots, k, \dots, n\}$ .
  - a. Set IFLAG =  $k$ .

4. Set  $y_{(j)t}^{\ell} = y_j^{\ell}$  and  $y_{(j)t}^u = y_j^u$  for  $j = 1, 2, \dots, n$ .

Stage 4

1. Set  $j = 1$ .
2. Set highest ranked variable at its upper bound,  
 $y_1 = y_1^u$ , if  $MAX = 1$ ; otherwise, set  $y_1 = y_1^l$ .
3. Set  $ZSUBT = ZOF$ ,  $NOTE4 = 0$ , and  $b_i^t = b_i$  for  
 $i = 1, 2, \dots, m$ .
4. Find a new modified objective function value,  
 $ZSUBT$ , based on the held value of the variables.
  - a. If  $ZSUBT$  has not been found before, then

$$ZSUBT = ZSUBT - \hat{c}_1 y_1; \quad (A-2)$$

otherwise,

$$ZSUBT = ZSUBT - \hat{c}_{j+1} y_{j+1}. \quad (A-3)$$

- b. If  $ZSUBT \geq 0$ , go to step 5; otherwise, if  
 $\hat{c}_{j+1} < 0$ , go to step 5; otherwise, go to  
 step 12.
5. Check to see if  $ZSUBT$  is an odd integer.
  - a. If yes, go to step 6.
  - b. If no, go to step 7.
6. Check to see if all succeeding objective  
 function coefficients in the ranking are  
 even integers.
  - a. If  $IFLAG = 1$ , go to step 26.
  - b. If  $IFLAG \neq j + 1$ , go to step 7; otherwise,  
 if  $j = 1$ , go to step 26; otherwise, set  
 $ZSUBT = ZSUBT + \hat{c}_{j+1} y_{j+1}$ ; and, if  $MAX = 1$ ,  
 set  $y_{j+1} = y_{j+1} - 1$ ; otherwise, set



$$y_{j+1} = y_{j+1} + 1, \text{ and} \quad (\text{A-4})$$

go to step 24.

7. Determine if the final variable is to be calculated.

- a. If  $n = 2$ , go to step 8; otherwise,  
check if  $n - 1 \neq j + 1$ .
  - (1) If yes, go to step 13.
  - (2) If no, go to step 8, unless b-values  
not found for  $y_1$ , go to step 13.

8. Check to see if  $y_n$  is integer from

$$y_n = \text{ZSUBT}/\hat{c}_n. \quad (\text{A-5})$$

- a. If yes, to to step 9.
- b. If no, go to step 12.

9. Check to see if  $y_n$  is greater than  $y_n^u$ .

- a. If yes, go to step 10.
- b. If no, go to step 11.

10. Check if  $n = 2$ .

- a. If yes, go to step 26.
- b. If no, go to step 12.

11. Test the feasibility of the solution in the functional constraints.

- a. If the solution is feasible, the optimum  
feasible integer solution has been found.
- b. If the solution is infeasible in any func-  
tional constraint, go to step 12.

12. A test solution has been found infeasible.

Increment  $y_{j+1}$  one integer amount.

- a. If  $\text{MAX} = 1$ , set

$$y_{j+1} = y_{j+1} - 1; \quad (\text{A-6})$$

otherwise, set

$$y_{j+1} = y_{j+1} + 1. \quad (\text{A-7})$$

- b. Go to step 24.

13. Calculate new right-hand side b-values.

- a. If new b-values have not been calculated for the held value of  $y_1$ , then

$$b_i^t = b_i^t - a_{i1} y_1 \quad (\text{A-8})$$

for  $i = 1, 2, \dots, m$ .

- b. If new b-values have been calculated for the held value of  $y_1$ , then

$$b_i^t = b_i^t - \hat{a}_{i,j+1} y_{j+1} \quad (\text{A-9})$$

for  $i = 1, 2, \dots, m$ .

- c. If any  $b_i^t < 0$ , for  $i = 1, 2, \dots, m$ , then set the flag  $\text{NOTE4} = 1$ .

- d. Go to step 14.

14. Determine if  $j$  should be incremented.

- a. If  $j \neq 1$ , set  $j = j + 1$  and go to step 15; otherwise, check if tighter bounds have been found on  $y_{j+1}$ .

- (1) If yes, set  $j = j + 1$  and go to step 15.
- (2) If no, go to step 15.
15. If  $j + 1 \neq 2$ , go to step 19; otherwise, go to step 16.
16. If  $\text{NOTE}_4 = 1$ , go to step 17; otherwise, go to step 18.
17. Check if  $\text{MAX} = 1$ .
- a. If yes, set  $y_{(2)t}^u = y_2^u$ , and go to step 23.
- b. If no, set  $y_{(2)t}^l = y_2^l$ , and go to step 23.
18. Find a new, tighter bound on the next variable in the ranking,  $y_2$ .

a. If  $\text{MAX} = 1$ , considering  $\hat{c}_2 > 0$  and  $\hat{a}_{12} > 0$ , adding  $-\hat{a}_{1j} y_j^u$  to  $b_1^t$  for all  $\hat{a}_{1j} < 0$ ,

$$y_{(2)t}^u = \min_{i=1,2,\dots,m} \{ [\text{ZSUBT}/\hat{c}_2], [b_1^t/\hat{a}_{i2}] \}$$

(A-10)

and, check if  $y_{(2)t}^u \leq y_2^u$ .

(1) If yes, go to step 23.

(2) If no, set  $y_{(2)t}^u = y_2^u$ , and go to step 23.

b. If  $\text{MAX} = 0$ , considering  $\hat{c}_2 > 0$  and  $\hat{a}_{12} > 0$ , adding  $-\hat{a}_{1j} y_j^l$  to  $b_1^t$  for all  $\hat{a}_{1j} < 0$ ,

$$y_{(2)t}^l = \max_{i=1,2,\dots,m} \{ \langle \text{ZSUBT}/\hat{c}_2 \rangle, \langle b_1^t/\hat{a}_{i2} \rangle \}$$

(A-11)

and check if  $y_{(2)t}^l \geq y_2^l$ .

- (1) If yes, go to step 23.
- (2) If no, set  $y_{(2)t}^{\ell} = y_2^{\ell}$ , and go to step 23.
19. If  $MAX = 0$ , go to step 22; if not, go to step 20.
20. If  $NOTE4 = 1$ , set  $y_{(j+1)t}^u = y_{j+1}^u$ , and go to step 23; otherwise, go to step 21.
21. Considering only  $\hat{c}_{j+1} > 0$  and  $\hat{a}_{i,j+1} > 0$ , adding  $-\hat{a}_{i,j+1} y_{j+1}^u$  to  $b_i^t$  for all  $\hat{a}_{i,j+1} < 0$ ,  

$$y_{(j+1)t}^u = \min_{i=1,2,\dots,m} \{ [ZSUBT/\hat{c}_{j+1}], [b_i^t/\hat{a}_{i,j+1}] \}$$
(A-12)  
and, check if  $y_{(j+1)t}^u \leq y_{j+1}^u$ .  
a. If yes, go to step 23.  
b. If no, set  $y_{(j+1)t}^u = y_{j+1}^u$ , and go to step 23.
22. Considering only  $\hat{c}_{j+1} > 0$  and  $\hat{a}_{i,j+1} > 0$ , adding  $-\hat{a}_{i,j+1} y_{j+1}^{\ell}$  to  $b_i^t$  for all  $\hat{a}_{i,j+1} < 0$ ,  

$$y_{(j+1)t}^{\ell} = \max_{i=1,2,\dots,m} \{ \langle ZSUBT/\hat{c}_{j+1} \rangle, \langle b_i^t/\hat{a}_{i,j+1} \rangle \}$$
(A-13)  
and, check if  $y_{(j+1)t}^{\ell} \geq y_{j+1}^{\ell}$ .  
a. If yes, go to step 23.  
b. If no, set  $y_{(j+1)t}^{\ell} = y_{j+1}^{\ell}$ , and go to step 23.
23. Set next variable in ranking at its new bound.  
a. If  $MAX = 1$ , set  

$$y_{j+1} = y_{(j+1)t}^u ;$$
(A-14)  
otherwise, set  

$$y_{j+1} = y_{(j+1)t}^{\ell}$$
(A-15)  
and, go to step 24.
24. Check to see if a variable has been incremented

beyond its bound.

a. If  $MAX = 0$ , check if  $y_{j+1} \leq y_{(j+1)t}^u$ .

(1) If yes, go to step 4.

(2) If no, set  $y_{j+1} = y_{(j+1)t}^u$ , and go to step 25.

b. If  $MAX = 1$ , check if  $y_{j+1} \geq y_{(j+1)t}^l$ .

(1) If yes, go to step 4.

(2) If no, set  $y_{j+1} = y_{(j+1)t}^l$ , and go to step 25.

25. If  $j = 1$ , go to step 26; otherwise, set  $j = j - 1$ ,

$$ZSUBT = ZSUBT + \hat{c}_{j+1} y_{j+1}, \quad (A-16)$$

and

$$b_i^t = b_i^t + \hat{a}_{i,j+1} y_{j+1}, \quad (A-17)$$

for  $i = 1, 2, \dots, m$ ; also, set  $NOTE4 = 0$ , and check to see if  $MAX = 1$ .

(1) If yes, set  $y_{j+1} = y_{j+1} - 1$ , and go to step 24.

(2) If no, set  $y_{j+1} = y_{j+1} + 1$ , and go to step 24.

26. Increment first variable in ranking,  $y_1$ , one integer.

a. If  $MAX = 1$ , set

$$y_1 = y_1 - 1, \quad (A-18)$$

and go to step 27; otherwise, set

$$y_1 = y_1 + 1, \quad (A-19)$$

and go to step 27.

27. Check to see if all solutions have been examined.

a. If  $MAX = 1$ , check if  $y_1 < y_{(1)t}^{\ell}$ .

(1) If yes, set  $ZOF = ZOF - g$ , and go to step 1.

(2) If no, set  $ZSUBT = ZOF$ , and go to step 28.

b. If  $MAX = 0$ , check if  $y_1 > y_{(1)t}^u$ .

(1) If yes, check  $ZOF \leq 0$ .

(a) If yes, set  $ZOF = ZOF - g$ , and go to step 1.

(b) If no, set  $ZOF = ZOF + g$ , and go to step 1.

(2) If no, set  $ZSUBT = ZOF$ , and go to step 28.

28. Check if  $n = 2$ .

a. If yes, go to step 4.

b. If no, set  $b_i^t = b_i$  for  $i = 1, 2, \dots, m$ , and go to step 4.

## APPENDIX B

### MAIN TEST PROGRAM

```

$JOB 12217,441-38-2562,REGION=40K,TIME=30 JAMES M. SHIRLEY
C *****
C *
C * THE OBJECTIVE FUNCTION
C * REDUCTION ALGORITHM
C *
C * JAMES M. SHIRLEY
C *
C * SCHOOL OF INDUSTRIAL ENGINEERING
C * AND MANAGEMENT
C * OKLAHOMA STATE UNIVERSITY
C * MAY 14,1972
C *
C * THIS PROGRAM IS THE MAIN DRIVER TEST PROGRAM TO CALL
C * INTEGER LINEAR PROGRAM SUBROUTINES. THE SUBROUTINES USE AN
C * OBJECTIVE FUNCTION REDUCTION ALGORITHM DEVELOPED DURING RESEARCH
C * INTO INTEGER LINEAR PROGRAMMING. FOR FURTHER INFORMATION SEE
C * THE DOCTORAL THESIS "AN OBJECTIVE FUNCTION REDUCTION ALGORITHM
C * FOR INTEGER LINEAR PROGRAMMING."
C *
C * THE USER MUST SUPPLY THE FOLLOWING INFORMATION ON CONTROL
C * CARDS AS INPUT DATA:
C * ORDER CONSTRAINTS SO ALL GREATER-THAN-OR-EQUAL TO
C * CONSTRAINTS APPEAR LAST.
C *
C * CARD 1 - LM,KM,LN,MAX: FORMAT(4I5)
C *
C * LM = THE NUMBER OF LESS-THAN-OR-EQUAL-TO
C * CONSTRAINTS
C * KM = THE NUMBER OF GREATER-THAN-OR-EQUAL-TO
C * CONSTRAINTS
C * LN = THE NUMBER OF REAL PROBLEM VARIABLES
C * MAX = SET EQUAL 1 FOR MAXIMIZATION
C * SET EQUAL 0 FOR MINIMIZATION
C *
C * CARD 2 - ZSIM: FORMAT(F10.2)
C *
C * ZSIM = OBJECTIVE FUNCTION VALUE FOR CONTINUOUS
C * VARIABLE SOLUTION
C *
C * CARD 3 - AND SUCCESSIVE CARDS REQUIRED TO DEFINE THE
C * CONSTRAINT COEFFICIENTS,A(I,J). ONE COEFFICIENT
C * PER CARD,READ BY THE ROWS: FORMAT(10.2)
C *
C * A(I,J) = CONSTRAINT COEFFICIENT OF THE I TH ROW
C * AND J TH COLUMN
C *
C * CARD J - AND SUCCESSIVE CARDS REQUIRED TO DEFINE THE
C * RIGHT-HAND SIDE CONSTRAINT B-VALUES. ONE VALUE
C * PER CARD: FORMAT(F10.2)
C *
C * B(I) = CONSTRAINT RIGHT-HAND SIDE VALUES
C *
C * CARD K - AND SUCCESSIVE CARDS REQUIRED TO DEFINE THE

```

```

* OFR0001
* OFR0002
* OFR0003
* OFR0004
* OFR0005
* OFR0006
* OFR0007
* OFR0008
* OFR0009
* OFR0010
* OFR0011
* OFR0012
* OFR0013
* OFR0014
* OFR0015
* OFR0016
* OFR0017
* OFR0018
* OFR0019
* OFR0020
* OFR0021
* OFR0022
* OFR0023
* OFR0024
* OFR0025
* OFR0026
* OFR0027
* OFR0028
* OFR0029
* OFR0030
* OFR0031
* OFR0032
* OFR0033
* OFR0034
* OFR0035
* OFR0036
* OFR0037
* OFR0038
* OFR0039
* OFR0040
* OFR0041
* OFR0042
* OFR0043
* OFR0044
* OFR0045
* OFR0046
* OFR0047
* OFR0048
* OFR0049
* OFR0050
* OFR0051
* OFR0052
* OFR0053
* OFR0054
* OFR0055
* OFR0056
* OFR0057
* OFR0058

```



C	*	OBJECTIVE FUNCTION COEFFICIENTS. ONE VALUE PER	* OFR0059
C	*	CARD: FORMAT(I10)	* OFR0060
C	*		* OFR0061
C	*	C(J) = OBJECTIVE FUNCTION COEFFICIENTS	* OFR0062
C	*		* OFR0063
C	*	CARD L - AND SUCCESSIVE CARDS REQUIRED TO DEFINE THE	* OFR0064
C	*	UPPER INTEGER LIMIT ON EACH VARIABLE. ONE VALUE	* OFR0065
C	*	PER CARD: FORMAT(I10)	* OFR0066
C	*		* OFR0067
C	*	XU(J) = UPPER INTEGER LIMIT ON VARIABLE X(J)	* OFR0068
C	*		* OFR0069
C	*	CARD M - AND SUCCESSIVE CARDS REQUIRED TO DEFINE THE	* OFR0070
C	*	LOWER INTEGER LIMIT ON EACH VARIABLE. ONE VALUE	* OFR0071
C	*	PER CARD: FORMAT(I10)	* OFR0072
C	*		* OFR0073
C	*	XL(J) = LOWER INTEGER LIMIT ON VARIABLE X(J)	* OFR0074
C	*		* OFR0075
C	*		* OFR0076
C	*		* OFR0077
C	*		* OFR0078
C	*	PARAMETERS FOR PROGRAM:	* OFR0079
C	*		* OFR0080
C	*		* OFR0081
C	*	KI = INPUT DEVICE NUMBER ASSOCIATED WITH READ STATEMENT	* OFR0082
C	*	KO = OUTPUT DEVICE NUMBER ASSOCIATED WITH WRITE STATEMENT	* OFR0083
C	*	MCOUNT = A COUNTER KEEPING TRACK OF THE NUMBER OF ITERATIONS	* OFR0084
C	*	REQUIRED	* OFR0085
C	*	NCOUNT = A COUNTER LIMIT SET ON THE UPPER LIMIT ON THE NUMBER OF	* OFR0086
C	*	ITERATIONS	* OFR0087
C	*	NOFLAG = A COUNTER TO RECORD THE NUMBER OF TIMES IFLAG TRUNCATES	* OFR0088
C	*	THE SEARCH	* OFR0089
C	*	NITER = NUMBER OF ITERATIONS ALLOWED	* OFR0090
C	*	NCNT1 = A FLAG EQUALING 1 WHEN NUMBER OF ITERATIONS EXCEEDED;	* OFR0091
C	*	OTHERWISE, IT EQUALS ZERO	* OFR0092
C	*	ERR1 = AN INDICATOR WHICH EQUALS 1 WHEN AN OBJECTIVE FUNCTION	* OFR0093
C	*	COEFFICIENT EQUALS ZERO; OTHERWISE, IT EQUALS ZERO	* OFR0094
C	*	ERR2 = AN INDICATOR WHICH EQUALS 1 WHEN AN ERROR IN THE	* OFR0095
C	*	EUCLID'S ALGORITHM CAUSED A NEGATIVE REMAINDER	* OFR0096
C	*	EPS = ERROR TEST LIMIT	* OFR0097
C	*	LM = THE NUMBER OF LESS THAN OR EQUAL TO CONSTRAINTS	* OFR0098
C	*	KM = THE NUMBER OF GREATER THAN OR EQUAL TO CONSTRAINTS	* OFR0099
C	*	LN = THE NUMBER OF REAL VARIABLES IN THE PROBLEM	* OFR0100
C	*	MAX = AN INDICATOR WHICH EQUALS 1 WHEN OBJECTIVE FUNCTION IS TO	* OFR0101
C	*	BE MAXIMIZED; EQUALS ZERO FOR MINIMIZATION	* OFR0102
C	*	A = MATRIX OF CONSTRAINT COEFFICIENTS	* OFR0103
C	*	B = COLUMN VECTOR OF CONSTRAINT RIGHT-HAND SIDE VALUES	* OFR0104
C	*	C = THE COEFFICIENT VECTOR FOR THE OBJECTIVE FUNCTION	* OFR0105
C	*	XU = COLUMN VECTOR GIVING THE UPPER LIMIT ON EACH REAL VARIABLE	* OFR0106
C	*	XL = COLUMN VECTOR GIVING THE LOWER LIMIT ON EACH REAL VARIABLE	* OFR0107
C	*	X = SOLUTION VECTOR OF INTEGER VALUES	* OFR0108
C	*	KY = THE CONSTRAINT ROW WHERE THE GREATER-THAN-OR-EQUAL-TO	* OFR0109
C	*	CONSTRAINTS BEGIN	* OFR0110
C	*	KZ = THE TOTAL NUMBER OF CONSTRAINT ROWS	* OFR0111
C	*	ZSIM = THE VALUE OF THE OBJECTIVE FUNCTION AT THE CONTINUOUS	* OFR0112
C	*	VARIABLE SOLUTION	* OFR0113
C	*	ZOF = THE FIRST INTEGER OBJECTIVE FUNCTION VALUE SEARCHED	* OFR0114
C	*	G = THE GREATEST COMMON DIVISOR OF THE OBJECTIVE FUNCTION	* OFR0115
C	*	COEFFICIENTS	* OFR0116
C	*	IFLAG = INDICATES ALL RANKED OBJECTIVE FUNCTION COEFFICIENTS	* OFR0117

C	C	* SUBSCRIPT FROM SUBSCRIPT IFLAG TO N ARE EVEN	* QFR0118
C	C	* Y = COLUMN VECTOR INDICATING THE RANKING OF THE REAL VARIABLES BY*	* QFR0119
C	C	* RECORDING THE SUBSCRIPTS OF THE VARIABLES	* QFR0120
C	C	* RX = A VECTOR DESCRIBING THE NUMBER OF INTEGER VALUES THE J TH	* QFR0121
C	C	* ELEMENT CAN TAKE ON	* QFR0122
C	C	*	* QFR0123
C	C	*****	* QFR0124
C	C		QFR0125
C	C		QFR0126
C	C	INTEGER C(10),XL(10),XU(10),Y(10),X(10),ZOF,G,ERR1,ERR2,RX(10)	QFR0127
C	C	DIMENSION A(10,10),B(10)	QFR0128
C	C		QFR0129
C	C		QFR0130
C	C		QFR0131
C	C	KI = 5	QFR0132
C	C	KO = 6	QFR0133
C	C	MCOUNT = 0	QFR0134
C	C	NCOUNT = 0	QFR0135
C	C	NOFLAG = 0	QFR0136
C	C	NCNT1 = 0	QFR0137
C	C	NITER = 4	QFR0138
C	C	ERR1 = 0	QFR0139
C	C	ERR2 = 0	QFR0140
C	C	EPS = 0.1E-04	QFR0141
C	C		QFR0142
C	C	READ IN REQUIRED INPUT INFORMATION	QFR0143
C	C		QFR0144
C	C	READ(KI,10) LM,KM,LN,MAX	QFR0145
C	C	10 FORMAT(4I5)	QFR0146
C	C	KY = LM + 1	QFR0147
C	C	KZ = LM + KM	QFR0148
C	C		QFR0149
C	C		QFR0150
C	C		QFR0151
C	C	READ(KI,15) ZSIM	QFR0152
C	C	15 FORMAT(F10.2)	QFR0153
C	C	READ(KI,20)((A(LX,LY),LY=1,LN),LX=1,KZ)	QFR0154
C	C	20 FORMAT(F10.2)	QFR0155
C	C	READ(KI,20)(B(LZ),LZ=1,KZ)	QFR0156
C	C	READ(KI,30)(C(LV),LV=1,LN)	QFR0157
C	C	30 FORMAT(I10)	QFR0158
C	C	READ(KI,30)(XU(LU),LU=1,LN)	QFR0159
C	C	READ(KI,30)(XL(LT),LT=1,LN)	QFR0160
C	C		QFR0161
C	C	WRITE OUT THE INPUT DATA	QFR0162
C	C		QFR0163
C	C	IF(MAX.EQ.1) GO TO 50	QFR0164
C	C	WRITE(KO,40)	QFR0165
C	C	40 FORMAT(1H1,10X,21H MINIMIZATION PROBLEM)	QFR0166
C	C	GO TO 65	QFR0167
C	C	50 WRITE(KO,60)	QFR0168
C	C	60 FORMAT(1H1,10X,21H MAXIMIZATION PROBLEM)	QFR0169
C	C	65 WRITE(KO,70)	QFR0170
C	C	70 FORMAT(1H0,10X,37H OBJECTIVE FUNCTION COEFFICIENTS : )	QFR0171
C	C	WRITE(KO,80)(C(LV),LV=1,LN)	QFR0172
C	C	80 FORMAT(1H0,19X,10I7)	QFR0173
C	C	IF(LM.EQ.0) GO TO 120	QFR0174
C	C	WRITE(KO,90)	QFR0175
C	C	90 FORMAT(1H0,10X,47H LESS-THAN-OR-EQUAL-TO CONSTRAINT COEFFICIENTS:)	QFR0176

```

DO 110 JA=1,LM
  WRITE(KO,100)(A(JA,JB),JB=1,LN)
100  FORMAT(1H0,20X,10F7.2)
110  CONTINUE
  IF(KM.EQ.0) GO TO 150
120  WRITE(KO,130)
130  FORMAT(1H0,10X,50H GREATER-THAN-OR-EQUAL-TO CONSTRAINT COEFFICIENT
  #S:)
  DO 140 JC=KY,KZ
    WRITE(KO,100)(A(JC,JD),JD=1,LN)
140  CONTINUE
150  WRITE(KO,155)
155  FORMAT(1H0,10X,41H RIGHT-HAND SIDE B-VALUES OF CONSTRAINTS:)
  DO 170 JE=1,KZ
    BJ=B(JE)
    WRITE(KO,160)JE,BJ
160  FORMAT(1H0,22X,3H B(,11,3H) =,F7.2)
170  CONTINUE
    WRITE(KO,175)
175  FORMAT(1H0,10X,23H VARIABLE LOWER BOUNDS:)
  DO 185 JF = 1,LN
    MXL = XL(JF)
    WRITE(KO,180)JF,MXL
180  FORMAT(1H0,22X,4H XL(,11,3H) =,15)
185  CONTINUE
    WRITE(KO,190)
190  FORMAT(1H0,10X,23H VARIABLE UPPER BOUNDS:)
  DO 200 JG = 1,LN
    MXU = XU(JG)
    WRITE(KO,195)JG,MXU
195  FORMAT(1H0,22X,4H XU(,11,3H) =,15)
200  CONTINUE
C
C  CALCULATE THE GREATEST COMMON DIVISOR AND ZOF WITH SUBROUTINE GCD
C
  CALL GCD(C,G,LN,ZSIM,ZOF,MAX,ERR1,ERR2)
  IF(ERR1.EQ.0) GO TO 210
  WRITE(KO,205)
205  FORMAT(1H1,50H * AN OBJECTIVE FUNCTION COEFFICIENT EQUALS ZERO *)
  GO TO 1000
210  IF(ERR2.EQ.0) GO TO 220
  WRITE(KO,215)
215  FORMAT(1H1,59H** A REMAINDER WAS FORCED NEGATIVE IN EUCLID'S ALGOR
  #ITHM **)
  GO TO 1000
220  WRITE(KO,225) G
225  FORMAT(1H0,10X,27H GREATEST COMMON DIVISOR: ,3HG =,15)
  WRITE(KO,230) ZOF
230  FORMAT(1H0,10X,44H INITIAL INTEGER OBJECTIVE FUNCTION VALUE: ,5HZ
  #CF =,15)
C
C  RANK THE VARIABLES WITH SUBROUTINE RANK
C
  CALL RANK(C,XL,XU,LN,Y,IFLAG,RX)
  WRITE(KO,235)
235  FORMAT(1H0,10X,22H VARIABLE RANGE SIZES:)
  DO 245 JH = 1,LN
    MRX = RX(JH)
    WRITE(KO,240) JH,MRX

```

OFR0177  
 OFR0178  
 OFR0179  
 OFR0180  
 OFR0181  
 OFR0182  
 OFR0183  
 OFR0184  
 OFR0185  
 OFR0186  
 OFR0187  
 OFR0188  
 OFR0189  
 OFR0190  
 OFR0191  
 OFR0192  
 OFR0193  
 OFR0194  
 OFR0195  
 OFR0196  
 OFR0197  
 OFR0198  
 OFR0199  
 OFR0200  
 OFR0201  
 OFR0202  
 OFR0203  
 OFR0204  
 OFR0205  
 OFR0206  
 OFR0207  
 OFR0208  
 OFR0209  
 OFR0210  
 OFR0211  
 OFR0212  
 OFR0213  
 OFR0214  
 OFR0215  
 OFR0216  
 OFR0217  
 OFR0218  
 OFR0219  
 OFR0220  
 OFR0221  
 OFR0222  
 OFR0223  
 OFR0224  
 OFR0225  
 OFR0226  
 OFR0227  
 OFR0228  
 OFR0229  
 OFR0230  
 OFR0236  
 OFR0237  
 OFR0238  
 OFR0239  
 OFR0240

240	FORMAT(1H0,22X,4H RX(,11,3H) =,15)	OFR0241
245	CONTINUE	OFR0242
	WRITE(KO,250)	OFR0243
250	FORMAT(1H0,10X,36H COLUMN VECTOR OF RANKED SUBSCRIPTS:)	OFR0244
	DO 260 JK =1, LN	OFR0245
	MY = Y(JK)	OFR0246
	WRITE(KO,255) JK, MY	OFR0247
255	FORMAT(1H0,22X,3H Y(,11,3H) =,15)	OFR0248
260	CONTINUE	OFR0249
	WRITE(KO,265) IFLAG	OFR0250
265	FORMAT(1H0,10X,46H VALUE OF EVEN COEFFICIENT INDICATOR: IFLAG =,I	OFR0251
	#5)	OFR0252
C		OFR0253
C	BEGIN EXAMINING SOLUTION SPACE WITH SUBROUTINE SEARCH	OFR0254
C		OFR0255
	CALL SEARCH(A,B,C,Y,XU,XL,X,EPS,IFLAG,G,LM,KM,LN,MAX,NCOUNT,ZOF,NC	OFR0256
	#NT1,NITER,MCOUNT,NOFLAG)	OFR0257
	IF(NCNT1.EQ.0) GO TO 520	OFR0258
	WRITE(KO,345)	OFR0259
345	FORMAT(1H0,' *** NUMBER OF ITERATIONS EXCEEDED ***')	OFR0260
	GO TO 899	OFR0261
C		OFR0262
C	WRITE SOLUTION INFORMATION	OFR0263
C		OFR0264
520	WRITE(KO,525)	OFR0265
525	FORMAT(1H1)	OFR0266
	DO 510 JC = 1, LN	OFR0267
	JX = X(JC)	OFR0268
	WRITE(KO,505) JC, JX	OFR0269
505	FORMAT(1H ,40X,'***** X(,11,) = ',15)	OFR0270
510	CONTINUE	OFR0271
	WRITE(KO,515) ZOF	OFR0272
515	FORMAT(1H0,40X,'***** Z = ',15)	OFR0273
	WRITE(KO,500) MCOUNT	OFR0274
500	FORMAT(1H-,40X,'Y(N) WAS CALCULATED',15,3X,'TIMES')	OFR0275
	WRITE(KO,501) NOFLAG	OFR0276
501	FORMAT(1H-,40X,'IFLAG WAS USED ',15,3X,'TIMES TO TRUNCATE SEARCH')	OFR0277
899	WRITE(KO,900)	OFR0278
900	FORMAT(1H1)	OFR0279
1000	STOP	OFR0280
	END	OFR0281

## APPENDIX C

### SUBROUTINE GCD

```

SUBROUTINE GCD(C,G,LN,ZSIM,ZOF,MAX,ERR1,ERR2)
*****
*
*          SUBROUTINE GCD
*
*          JAMES M. SHIRLEY
*
*          SCHOOL OF INDUSTRIAL ENGINEERING
*          AND MANAGEMENT
*          OKLAHOMA STATE UNIVERSITY
*          MAY 14,1972
*
*          THIS SUBROUTINE CALCULATES THE GREATEST COMMON DIVISOR FOR
*          A SET OF OBJECTIVE FUNCTION COEFFICIENTS. IT ALSO DETERMINES
*          THE FIRST OBJECTIVE FUNCTION VALUE USED IN SUBROUTINE SEARCH.
*          FOR FURTHER INFORMATION SEE THE DOCTORAL THESIS "AN OBJECTIVE
*          FUNCTION REDUCTION ALGORITHM FOR INTEGER LINEAR PROGRAMMING."
*
*          PARAMETERS FOR PROGRAM:
*
*          C = THE COEFFICIENT VECTOR FOR THE OBJECTIVE FUNCTION
*          G = THE GREATEST COMMON DIVISOR OF THE OBJECTIVE FUNCTION
*              COEFFICIENTS
*          LN = THE NUMBER OF REAL VARIABLES IN THE PROBLEM
*          ZSIM= THE VALUE OF THE OBJECTIVE FUNCTION AT THE CONTINUOUS
*              VARIABLE SOLUTION
*          ZOF = THE FIRST INTEGER OBJECTIVE FUNCTION VALUE SEARCHED
*          MAX = AN INDICATOR WHICH EQUALS 1 WHEN OBJECTIVE FUNCTION IS TO
*              BE MAXIMIZED; IT EQUALS ZERO FOR MINIMIZATION
*          ERR1 = AN INDICATOR WHICH EQUALS 1 WHEN AN OBJECTIVE FUNCTION
*              COEFFICIENT EQUALS ZERO; OTHERWISE, IT EQUALS ZERO
*          ERR2 = AN INDICATOR WHICH EQUALS 1 WHEN AN ERROR IN THE
*              CALCULATION OF EUCLID'S ALGORITHM CAUSED A NEGATIVE
*              REMAINDER TO BE FORMED; OTHERWISE, IT EQUALS ZERO
*          GC = A MODIFIED COEFFICIENT VECTOR WHERE ALL VALUES ARE POSITIVE
*
*****
INTEGER C(10),GC(10),D,E,Q,R,TEMP,ZOF,ERR1,ERR2,G
EPS = 0.1E-04

ESTABLISH THE VECTOR GC WHICH HAS ALL POSITIVE ELEMENTS

DO 50 JA =1,LN
  IF(C(JA))20,30,40
  NCJA = C(JA)
  GC(JA) = ABS(NCJA)
  GO TO 50
20  ERR1 = 1
30  GO TO 300

```

40	GC(JA) = C(JA)	
50	CONTINUE	GCD0061
C		GCD0062
C	DETERMINE GREATEST COMMON DIVISOR, G, USING EUCLID'S ALGORITHM	GCD0063
C		GCD0064
	TEMP = GC(1)	GCD0065
	DO 145 L = 1, LN	GCD0066
	IF(TEMP - GC(L)) 70, 140, 90	GCD0067
70	D = GC(L)	GCD0068
	E = TEMP	GCD0069
	GO TO 100	GCD0070
90	D = TEMP	GCD0071
	E = GC(L)	GCD0072
100	Q = D/E	GCD0073
	R = D - (Q*E)	GCD0074
	IF(R) 110, 120, 130	GCD0075
110	ERR2 = 1	GCD0076
	GO TO 300	GCD0077
120	TEMP = E	GCD0078
	GO TO 140	GCD0079
130	D = E	GCD0080
	E = R	GCD0081
	GO TO 100	GCD0082
140	IF(TEMP.EQ.1) GO TO 150	GCD0083
	IF(L.EQ.LN) GO TO 150	GCD0084
145	CONTINUE	GCD0085
C		GCD0086
C	CALCULATE FIRST OBJECTIVE FUNCTION VALUE	GCD0087
C		GCD0088
150	G = TEMP	GCD0089
	ZOF = ZSIM	GCD0090
	TEMPG = G	GCD0091
160	TMPZOF = ZOF	GCD0092
	TEMP1 = TMPZOF/TEMPG	GCD0093
	NTEMP1 = TEMP1	GCD0094
	TEMP2 = NTEMP1	GCD0095
	DIFF = TEMP1 - TEMP2	GCD0096
	ABDIFF = ABS(DIFF)	GCD0097
	IF(ABDIFF - EPS) 300, 300, 170	GCD0098
170	IF(ZSIM) 180, 190, 190	GCD0099
180	IF(MAX - 1) 210, 200, 200	GCD0100
190	IF(MAX.EQ.1) GO TO 210	GCD0101
200	ZOF = ZOF + 1	GCD0102
	GO TO 160	GCD0103
210	ZOF = ZOF - 1	GCD0104
	GO TO 160	GCD0105
300	RETURN	GCD0106
	END	GCD0107

## APPENDIX D

### SUBROUTINE RANK



```

SUBROUTINE RANK(C,XL,XU,LN,Y,IFLAG,RX)
*****
*
* SUBROUTINE RANK
*
* JAMES M. SHIRLEY
*
* SCHOOL OF INDUSTRIAL ENGINEERING
* AND MANAGEMENT
* OKLAHOMA STATE UNIVERSITY
* MAY 14,1972
*
* THIS SUBROUTINE RANKS THE VARIABLES IN THE OBJECTIVE
* FUNCTION ACCORDING TO THEIR RANGE OF POSSIBLE VALUES. THE
* VARIABLE WITH THE SMALLEST RANGE IS RANKED FIRST. FOR FURTHER
* INFORMATION SEE THE DOCTORAL THESIS "AN OBJECTIVE FUNCTION
* REDUCTION ALGORITHM FOR INTEGER LINEAR PROGRAMMING."
*
*
* PARAMETERS FOR PROGRAM:
*
* C = THE COEFFICIENT VECTOR FOR THE OBJECTIVE FUNCTION
* XU = COLUMN VECTOR GIVING THE UPPER LIMIT ON EACH REAL VARIABLE
* XL = COLUMN VECTOR GIVING THE LOWER LIMIT ON EACH REAL VARIABLE
* IFLAG = INDICATES ALL RANKED OBJECTIVE FUNCTION COEFFICIENT
* SUBSCRIPTS FROM SUBSCRIPT IFLAG TO N ARE EVEN SUBSCRIPTS
* LN = THE NUMBER OF REAL VARIABLES IN THE PROBLEM
* Y = COLUMN VECTOR INDICATING THE RANKING OF THE REAL VARIABLES BY
* RECORDING THE SUBSCRIPTS OF THE VARIABLES
* RX = A VECTOR DESCRIBING THE NUMBER OF INTEGER VALUES THE J TH
* ELEMENT CAN TAKE ON
* EPS = ERROR TEST LIMIT
* RXT = TEMPORARY RX VALUES
* INDEX = A VECTOR OF THE SUBSCRIPTS OF THE RANKED VARIABLES
* JFLAG = AN INDICATOR WHICH CHECKS TO BE SURE ALL POSSIBLE EQUAL
* RANGE SIZES HAVE BEEN CONSIDERED
*****
INTEGER C(10),XL(10),XU(10),RX(10),Y(10),INDEX(10),RXT(10),CJA,CJA
#P1,CJE

EPS = 0.1E-04
LNH1 = LN - 1

DO 30 JA = 1,LN
  RX(JA) = XU(JA) - XL(JA) + 1
  RXT(JA) = RX(JA)
  INDEX(JA) = JA

```

	Y(JA) = 0	RNK0060
	30 CONTINUE	RNK0061
C		RNK0062
C	SORT RANGE SIZES WITH SHELL SORT	RNK0063
C		RNK0064
	M = LN	RNK0065
40	M = M/2	RNK0066
	IF(M.LE.EPS) GO TO 75	RNK0067
	K = LN - M	RNK0068
	J = 1	RNK0069
50	L = J	RNK0070
60	IF(RXT(L).LE.RXT(L+M)) GO TO 70	RNK0071
	TEMP = RXT(L)	RNK0072
	RXT(L) = RXT(L+M)	RNK0073
	RXT(L+M) = TEMP	RNK0074
	ITEMP = INDEX(L)	RNK0075
	INDEX(L) = INDEX(L+M)	RNK0076
	INDEX(L+M) = ITEMP	RNK0077
	L = L - M	RNK0078
	IF(L.GT.0) GO TO 60	RNK0079
70	J = J + 1	RNK0080
	IF(J-K)50,50,40	RNK0081
C		RNK0082
C	MODIFY RANKING FOR EQUAL RANGE SIZES	RNK0083
C		RNK0084
	75 JFLAG = 0	RNK0085
80	DO 100 JA = 1, LNM1	RNK0086
	JAP1 = JA + 1	RNK0087
	IF(RXT(JA).NE.RXT(JAP1)) GO TO 100	RNK0088
	CJA = C(INDEX(JA))	RNK0089
	CJAP1 = C(INDEX(JAP1))	RNK0090
	IF(CJA - CJAP1)90,100,100	RNK0091
90	ITEMP = INDEX(JA)	RNK0092
	INDEX(JA) = INDEX(JAP1)	RNK0093
	INDEX(JAP1) = ITEMP	RNK0094
	JFLAG = 1	RNK0095
100	CONTINUE	RNK0096
	IF(JFLAG.EQ.1) GO TO 75	RNK0097
	DO 105 JB = 1, LN	RNK0098
	Y(JB) = INDEX(JB)	RNK0099
105	CONTINUE	RNK0100
C		RNK0101
C	CHECK FOR EVEN INTEGER SEQUENCE AND SET IFLAG	RNK0102
C		RNK0103
	IFLAG = LN + 1	RNK0104
	DO 110 JD = 1, LN	RNK0105
	JE = LN - JD + 1	RNK0106
	CJE = C(INDEX(JE))	RNK0107
	TCJE = CJE	RNK0108
	DIV = TCJE/2.0	RNK0109
	NDIV = DIV	RNK0110
	TNDIV = NDIV	RNK0111
	TMUL = TNDIV * 2.0	RNK0112
	DIFF = TCJE - TMUL	RNK0113
	IF(DIFF.GT.EPS) GO TO 120	RNK0114
	IFLAG = JE	RNK0115
110	CONTINUE	RNK0116
120	RETURN	RNK0117
	END	RNK0118

## APPENDIX E

### SUBROUTINE SEARCH

```

SUBROUTINE SEARCH(A,B,C,Y,XU,XL,X,EPS,IFLAG,G,LM,KM,LN,MAX,NCOUNT, SEAO001
#ZDF,NCNT1,NITER,MCOUNT,NOFLAG) SEAO002
***** SEAO003
C * SEAO004
C * SEAO005
C * SEAO006
C * SEAO007
C * SEAO008
C * SEAO009
C * SEAO010
C * SEAO011
C * SEAO012
C * SEAO013
C * SEAO014
C * SEAO015
C * SEAO016
C * SEAO017
C * THIS SUBROUTINE SEARCHES FOR A FEASIBLE SOLUTION FOR A FIXED* SEAO018
C * VALUE OF THE OBJECTIVE FUNCTION IN AN INTEGER LINEAR PROGRAMMING * SEAO019
C * PROBLEM. IT ACCEPTS THE RANKING DETERMINED IN SUBROUTINE RANK. * SEAO020
C * IT BEGINS A COMBINATORIAL SEARCH BY HOLDING THE LOWEST RANKED * SEAO021
C * VARIABLE AT ITS UPPER LIMIT THEN EXPLICITLY OR IMPLICITLY * SEAO022
C * EXAMINES THE POSSIBLE RANGE OF ALL OTHER VARIABLES. FOR FURTHER * SEAO023
C * INFORMATION SEE THE DOCTORAL THESIS "AN OBJECTIVE FUNCTION * SEAO024
C * REDUCTION ALGORITHM FOR INTEGER LINEAR PROGRAMMING." * SEAO025
C * SEAO026
C * SEAO027
C * SEAO028
C * SEAO029
C * SEAO030
C * SEAO031
C * SEAO032
C * SEAO033
C * A = MATRIX OF CONSTRAINT COEFFICIENTS * SEAO034
C * B = COLUMN VECTOR OF CONSTRAINT RIGHT-HAND SIDE VALUES * SEAO035
C * BT = COLUMN VECTOR OF TEMPORARILY MODIFIED RIGHT-HAND SIDE VALUES* SEAO036
C * C = THE COEFFICIENT VECTOR FOR THE OBJECTIVE FUNCTION * SEAO037
C * ZSUBT = THE TEMPORARY VALUE OF THE OBJECTIVE FUNCTION * SEAO038
C * Y = COLUMN VECTOR INDICATING THE RANKING OF THE REAL VARIABLES BY* SEAO039
C * RECORDING THE SUBSCRIPTS OF THE VARIABLES * SEAO040
C * XU = COLUMN VECTOR GIVING THE UPPER LIMIT ON EACH REAL VARIABLE * SEAO041
C * XL = COLUMN VECTOR GIVING THE LOWER LIMIT ON EACH REAL VARIABLE * SEAO042
C * XUT = COLUMN VECTOR OF TEMPORARY UPPER LIMITS ON A VARIABLE * SEAO043
C * XLT = COLUMN VECTOR OF TEMPORARY LOWER LIMITS ON A VARIABLE * SEAO044
C * X = SOLUTION VECTOR OF INTEGER VALUES * SEAO045
C * XTEMP = COLUMN VECTOR OF TEMPORARY INTEGER VALUES OF VARIABLES * SEAO046
C * WHICH ARE AT A HELD VALUE DURING THE COMBINATORIAL SEARCH* SEAO047
C * EPS = ERROR TEST LIMIT * SEAO048
C * IFLAG = INDICATES ALL RANKED OBJECTIVE FUNCTION COEFFICIENTS * SEAO049
C * SUBSCRIPT FROM SUBSCRIPT IFLAG TO N ARE EVEN * SEAO050
C * G = GREATEST COMMON DIVISOR * SEAO051
C * NOTE1 = AN INDICATOR WHICH EQUALS 1 WHEN B-VALUES HAVE BEEN * SEAO052
C * CALCULATED FOR HELD VALUE OF Y SUB 1, ZERO OTHERWISE * SEAO053
C * NOTE2 = AN INDICATOR WHICH EQUALS 1 WHEN TIGHTER BOUNDS HAVE BEEN* SEAO054
C * FOUND ON VARIABLE Y SUB L+1,ZERO OTHERWISE * SEAO055
C * NOTE3 = AN INDICATOR WHICH EQUALS 1 WHEN ZSUBT HAS BEEN * SEAO056
C * NOTE4 = AN INDICATOR WHICH EQUALS ZERO WHILE ALL RIGHT-HAND SIDE * SEAO057
C * CALCULATED THE FIRST TIME USING Y SUB 1, ZERO OTHERWISE * SEAO058
C * VALUES ARE POSITIVE OR ZERO; IT EQUALS ONE WHEN A * SEAO059
C * RIGHT-HAND SIDE VALUE HAS BEEN FORCED TO A NEGATIVE VALUE*

```

```

C * LM = THE NUMBER OF LESS THAN OR EQUAL TO CONSTRAINTS * SEA0060
C * KM = THE NUMBER OF GREATER THAN OR EQUAL TO CONSTRAINTS * SEA0061
C * LN = THE NUMBER OF REAL VARIABLES IN THE PROBLEM * SEA0062
C * MAX = AN INDICATOR WHICH EQUALS 1 WHEN OBJECTIVE FUNCTION IS TO * SEA0063
C * BE MAXIMIZED; EQUALS ZERO FOR MINIMIZATION * SEA0064
C * NCOUNT = A COUNTER LIMIT SET ON THE UPPER LIMIT ON THE NUMBER OF * SEA0065
C * ITERATIONS * SEA0066
C * NITER = NUMBER OF ITERATIONS ALLOWED * SEA0067
C * NCNT1 = A FLAG EQUALING 1 WHEN NUMBER OF ITERATIONS EXCEEDED; * SEA0068
C * OTHERWISE, IT EQUALS ZERO * SEA0069
C * ZOF = AN INTEGER OBJECTIVE FUNCTION VALUE BEING SEARCHED * SEA0070
C * MCOUNT = A COUNTER KEEPING TRACK OF THE NUMBER OF ITERATIONS * SEA0071
C * REQUIRED * SEA0072
C * NOFLAG = A COUNTER TO RECORD THE NUMBER OF TIMES IFLAG TRUNCATED * SEA0073
C * THE SEARCH * SEA0074
C * ROW = AN INDICATOR VECTOR IDENTIFYING CONSTRAINT ROWS WITH * SEA0075
C * NEGATIVE COEFFICIENTS; ZERO EQUALS ALL POSITIVE, ONE EQUALS * SEA0076
C * ONE OR MORE NEGATIVE * SEA0077
C * * SEA0078
C ***** SEA0079
C SEA0080
C SEA0081
C INTEGER C(10),XL(10),XU(10),Y(10),X(10),XTEMP(10),XLT(10),XUT(10), SEA0082
C #ZOF,ZSUBT,G,Y1,YLP1,YN,YZ,ROW(10) SEA0083
C DIMENSION A(10,10),B(10),BT(10) SEA0084
C SEA0085
C SEA0086
C DO 1 KA = 1,LN SEA0087
C XUT(KA) = XU(KA) SEA0088
C XLT(KA) = XL(KA) SEA0089
C 1 CONTINUE SEA0090
C KY = LM + 1 SEA0091
C KZ = LM + KM SEA0092
C DO 3 NA = 1,KZ SEA0093
C DO 2 NB = 1,LN SEA0094
C ROW(NA) = 0 SEA0095
C IF(A(NA,NB).GE.0.0) GO TO 2 SEA0096
C ROW(NA) = 1 SEA0097
C GO TO 3 SEA0098
C 2 CONTINUE SEA0099
C 3 CONTINUE SEA0100
C SEA0101
C STEP NUMBER 1 SEA0102
C SEA0103
C INITIALIZATION TO BEGIN RECURSION SEA0104
C SEA0105
C 4 L = 1 SEA0106
C SEA0107
C CHECK IF NUMBER OF ITERATIONS EXCEEDED SEA0108
C SEA0109
C NCOUNT = NCOUNT + 1 SEA0110
C IF(NCOUNT.LE.NITER) GO TO 5 SEA0111
C NCNT1 = 1 SEA0112
C GO TO 1000 SEA0113
C 5 NOTE1 = 0 SEA0114
C NOTE2 = 0 SEA0115
C DO 6 KB = 1,LN SEA0116
C XTEMP(KB) = 0 SEA0117
C 6 CONTINUE SEA0118

```

C		SEA0119
C	STEP NUMBER 2	SEA0120
C		SEA0121
C	SET HIGHEST RANKED VARIABLE AT ITS BOUND	SEA0122
C		SEA0123
	7 Y1 = Y(1)	SEA0124
	IF(MAX.EQ.1) GO TO 8	SEA0125
	XTEMP(Y1) = XL(Y1)	SEA0126
	GO TO 9	SEA0127
	8 XTEMP(Y1) = XU(Y1)	SEA0128
	9 NOTE3 = 0	SEA0129
C		SEA0130
C	STEP NUMBER 3	SEA0131
C		SEA0132
C	INITIALIZE ZSUBT AND BT VECTOR	SEA0133
C		SEA0134
	ZSUBT = ZOF	SEA0135
	10 DO 11 KA = 1,KZ	SEA0136
	BT(KA) = B(KA)	SEA0137
	11 CONTINUE	SEA0138
	NOTE4 = 0	SEA0139
C		SEA0140
C	STEP NUMBER 4	SEA0141
C		SEA0142
C	FIND NEW MODIFIED OBJECTIVE FUNCTION VALUE, ZSUBT	SEA0143
C		SEA0144
	12 Y1 = Y(1)	SEA0145
	YLP1 = Y(L+1)	SEA0146
	IF(NOTE3.EQ.1) GO TO 14	SEA0147
	ZSUBT = ZSUBT - C(Y1)*XTEMP(Y1)	SEA0148
	NOTE3 = 1	SEA0149
	IF(ZSUBT)13,15,15	SEA0150
	13 IF(C(YLP1))15,75,75	SEA0151
	14 ZSUBT = ZSUBT - C(YLP1)*XTEMP(YLP1)	SEA0152
	IF(ZSUBT)13,15,15	SEA0153
C		SEA0154
C	STEP NUMBER 5	SEA0155
C		SEA0156
C	CHECK TO SEE IF ZSUBT IS AN ODD INTEGER	SEA0157
C		SEA0158
	15 YLP1 = Y(L+1)	SEA0159
	LPI = L+1	SEA0160
	TEMP = ZSUBT	SEA0161
	ALPHA = TEMP/2.0	SEA0162
	NALPHA = ALPHA	SEA0163
	BETA = NALPHA	SEA0164
	DIFF = ALPHA - BETA	SEA0165
	ABDIFF = ABS(DIFF)	SEA0166
	IF(ABDIFF-EPS)20,20,16	SEA0167
C		SEA0168
C	STEP NUMBER 6	SEA0169
C		SEA0170
C	CHECK TO SEE IF ALL SUCCEEDING OBJECTIVE FUNCTION COEFFICIENTS ARE	SEA0171
C		SEA0172
	16 IF(IFLAG.EQ.1) GO TO 350	SEA0173
	IF(LPI.NE.IFLAG) GO TO 20	SEA0174
C		SEA0175
C	NOFLAG KEEPS A RECORD OF THE NUMBER OF TIMES IFLAG IS USED TO	SEA0176
C	TRUNCATE THE SEARCH	SEA0177

C	NOFLAG = NOFLAG + 1	SEA0178
	IF(L.EQ.1) GO TO 350	SEA0179
	ZSUBT = ZSUBT + C(YLP1)*XTEMP(YLP1)	SEA0180
	IF(MAX.EQ.1) GO TO 18	SEA0181
	XTEMP(YLP1) = XTEMP(YLP1) + 1	SEA0182
	GO TO 340	SEA0183
18	XTEMP(YLP1) = XTEMP(YLP1) - 1	SEA0184
	GO TO 340	SEA0185
C		SEA0186
C	STEP NUMBER 7	SEA0187
C		SEA0188
C	DETERMINE IF FINAL VARIABLE IS TO BE CALCULATED	SEA0189
C		SEA0190
		SEA0191
20	LP1 = L+1	SEA0192
	YLP1 = Y(L+1)	SEA0193
	IF(LN.EQ.2) GO TO 22	SEA0194
	IF(LN-1.NE.LP1) GO TO 80	SEA0195
	IF(NOTE1.EQ.0) GO TO 80	SEA0196
C		SEA0197
C	STEP NUMBER 8	SEA0198
C		SEA0199
C	CHECK XTEMP(Y(N)) INTEGER	SEA0200
C		SEA0201
22	YN = Y(LN)	SEA0202
C		SEA0203
C	MCDUNT KEEPS A RECORD OF THE NUMBER OF TIMES Y(N) IS CALCULATED	SEA0204
C		SEA0205
	MCDUNT = MCDUNT + 1	SEA0206
	RZSUBT = ZSUBT	SEA0207
	RCYN = C(YN)	SEA0208
	RXTPYN = RZSUBT/RCYN	SEA0209
	NUM = RXTPYN	SEA0210
	DELTA = NUM	SEA0211
	GAMMA = RXTPYN - DELTA	SEA0212
	IF(GAMMA - EPS)30,30,75	SEA0213
C		SEA0214
C	STEP NUMBER 9	SEA0215
C		SEA0216
C	CHECK TO SEE IF XTEMP(Y(N)) IS GREATER THAN ITS UPPER BOUND	SEA0217
C		SEA0218
30	XTEMP(YN) = RXTPYN	SEA0219
	IF(XTEMP(YN))75,31,31	SEA0220
31	XT1 = XTEMP(YN)	SEA0221
	XT2 = XU(YN)	SEA0222
	IF(XT1 - XT2)50,50,40	SEA0223
C		SEA0224
C	STEP NUMBER 10	SEA0225
C		SEA0226
40	IF(LN - 2)350,350,75	SEA0227
C		SEA0228
C	STEP NUMBER 11	SEA0229
C		SEA0230
C	TEST SOLUTION FEASIBILITY IN FUNCTIONAL CONSTRAINTS	SEA0231
C		SEA0232
50	DO 55 JB = 1,LM	SEA0233
	VALUE = 0.0	SEA0234
	DO 54 JA = 1,LN	SEA0235
	VALUE = VALUE + A(JB,Y(JA)) * XTEMP(Y(JA))	SEA0236

54	CONTINUE	SEA0237
	TEST = B(JB) - VALUE	SEA0238
	IF(TEST)75,55,55	SEA0239
55	CONTINUE	SEA0240
56	IF(KM.LE.0) GO TO 63	SEA0241
	DO 60 KX = KY,KZ	SEA0242
	AMOUNT = 0.0	SEA0243
	DO 57 KW = 1, LN	SEA0244
	AMOUNT = AMOUNT + A(KX,Y(KW)) * XTEMP(Y(KW))	SEA0245
57	CONTINUE	SEA0246
	CHECK = B(KX) - AMOUNT	SEA0247
	IF(CHECK)60,60,75	SEA0248
60	CONTINUE	SEA0249
63	DO 65 JC = 1, LN	SEA0250
	X(JC) = XTEMP(JC)	SEA0251
65	CONTINUE	SEA0252
	GO TO 1000	SEA0253
C		SEA0254
C	STEP NUMBER 12	SEA0255
C		SEA0256
C	SOLUTION INFEASIBLE; INCREMENT XTEMP(Y(LP1)) ONE INTEGER	SEA0257
C		SEA0258
	75 IF(LN - 2)350,350,77	SEA0259
	77 ZSUBT = ZSUBT + C(YLP1)*XTEMP(YLP1)	SEA0260
	IF(MAX.EQ.1) GO TO 78	SEA0261
	XTEMP(YLP1) = XUT(YLP1) + 1	SEA0262
	GO TO 340	SEA0263
	78 XTEMP(YLP1) = XTEMP(YLP1) - 1	SEA0264
	GO TO 340	SEA0265
C		SEA0266
C	STEP NUMBER 13	SEA0267
C		SEA0268
C	CALCULATE NEW RIGHT-HAND SIDE B-VALUES	SEA0269
C		SEA0270
	80 LP1 = L+1	SEA0271
	YLP1 = Y(L+1)	SEA0272
	IF(NOTE1.EQ.1) GO TO 90	SEA0273
	DO 85 JD = 1,KZ	SEA0274
	BT(JD) = BT(JD) - A(JD,Y(1))*XTEMP(Y(1))	SEA0275
	IF(BT(JD))84,85,85	SEA0276
	84 NOTE4 = 1	SEA0277
	85 CONTINUE	SEA0278
	NOTE1 = 1	SEA0279
	GO TO 96	SEA0280
	90 RXTEMP = XTEMP(YLP1)	SEA0281
	DO 95 JE = 1,KZ	SEA0282
	BT(JE) = BT(JE) - A(JE,YLP1)*RXTEMP	SEA0283
	IF(BT(JE))94,95,95	SEA0284
	94 NOTE4 = 1	SEA0285
	95 CONTINUE	SEA0286
C		SEA0287
C	STEP NUMBER 14	SEA0288
C		SEA0289
C	DETERMINE IF L SHOULD BE INCREMENTED	SEA0290
C		SEA0291
	96 IF(L.EQ.1) GO TO 98	SEA0292
	97 L = L+1	SEA0293
	GO TO 99	SEA0294
	98 IF(NOTE2 - 1)99,97,97	SEA0295



C		SEA0296
C	FIND NEW BOUND ON NEXT VARIABLE IN RANKING	SEA0297
C		SEA0298
	99 LP1 = L+1	SEA0299
	YLP1 = Y(L+1)	SEA0300
	Y2 = Y(2)	SEA0301
C		SEA0302
C	STEP NUMBER 15	SEA0303
C		SEA0304
	IF(LP1.NE.2) GO TO 124	SEA0305
C		SEA0306
C	STEP NUMBER 16	SEA0307
C		SEA0308
	IF(NOTE4 - 1)102,100,100	SEA0309
C		SEA0310
C	STEP NUMBER 17	SEA0311
C		SEA0312
	100 IF(MAX.EQ.1) GO TO 101	SEA0313
	XLTY2 = XLTY2	SEA0314
	GO TO 330	SEA0315
	101 XLTY2 = XLTY2	SEA0316
	GO TO 330	SEA0317
C		SEA0318
C	STEP NUMBER 18	SEA0319
C		SEA0320
	102 NOTE2 = 1	SEA0321
	RZSUBT = ZSUBT	SEA0322
	RCY2 = C(Y2)	SEA0323
	Y1 = Y(1)	SEA0324
	IF(MAX.EQ.0) GO TO 107	SEA0325
	XLTY2 = RZSUBT/RCY2	SEA0326
	DO 106 JF = 1, LN	SEA0327
	IF(A(JF,Y2).LE.EPS) GO TO 106	SEA0328
	IF(ROW(JF).EQ.0) GO TO 104	SEA0329
	SUM = 0.0	SEA0330
	DO 103 NC = 1, LN	SEA0331
	IF(A(JF,NC).GE.0.0) GO TO 103	SEA0332
	SUM = SUM + A(JF,NC) * XU(NC)	SEA0333
	103 CONTINUE	SEA0334
	BT(JF) = BT(JF) - SUM	SEA0335
	104 MTEMP = BT(JF)/A(JF,Y2)	SEA0336
	IF(ROW(JF).EQ.0) GO TO 105	SEA0337
	BT(JF) = BT(JF) + SUM	SEA0338
	105 IF(MTEMP.GE.XLTY2) GO TO 106	SEA0339
	XLTY2 = MTEMP	SEA0340
	106 CONTINUE	SEA0341
	IF(XLTY2.LE.XU(Y2)) GO TO 330	SEA0342
	XLTY2 = XU(Y2)	SEA0343
	GO TO 330	SEA0344
	107 XLTY2 = RZSUBT/RCY2	SEA0345
	NUM2 = XLTY2	SEA0346
	DELTA1 = NUM2	SEA0347
	GAMMA1 = XLTY2 - DELTA1	SEA0348
	IF(GAMMA1 - EPS)108,108,109	SEA0349
	108 XLTY2 = XLTY2	SEA0350
	GO TO 110	SEA0351
	109 XLTY2T = XLTY2 + 1.0	SEA0352
	XLTY2T = XLTY2T	SEA0353
	110 DO 123 JK = KY,KZ	SEA0354

IF(A(JK,Y2).LE.EPS) GO TO 123	SEA0355
IF(ROW(JK).EQ.0) GO TO 112	SEA0356
SUM = 0.0	SEA0357
DO 111 ND = 1, LN	SEA0358
IF(A(JK,ND).GE.0.0) GO TO 111	SEA0359
SUM = SUM + A(JK,ND) * XL(ND)	SEA0360
111 CONTINUE	SEA0361
BT(JK) = BT(JK) - SUM	SEA0362
112 TEMP1 = BT(JK)/A(JK,Y2)	SEA0363
IF(ROW(JK).EQ.0) GO TO 113	SEA0364
BT(JK) = BT(JK) + SUM	SEA0365
113 NUM3 = TEMP1	SEA0366
DELTA3 = NUM3	SEA0367
GAMMA3 = TEMP1 - DELTA3	SEA0368
IF(GAMMA3 - EPS)120,120,121	SEA0369
120 NTEMP = TEMP1	SEA0370
GO TO 122	SEA0371
121 TEMP2 = TEMP1 + 1.0	SEA0372
NTEMP = TEMP2	SEA0373
122 IF(NTEMP.LE.XLT(Y2)) GO TO 123	SEA0374
XLT(Y2) = NTEMP	SEA0375
123 CONTINUE	SEA0376
IF(XLT(Y2).GE.XL(Y2)) GO TO 330	SEA0377
XLT(Y2) = XL(Y2)	SEA0378
GO TO 330	SEA0379
C	SEA0380
C STEP NUMBER 19	SEA0381
C	SEA0382
124 IF(MAX.EQ.0) GO TO 200	SEA0383
C	SEA0384
C STEP NUMBER 20	SEA0385
C	SEA0386
125 IF(NOTE4 - 1)127,126,126	SEA0387
126 XUT(YLP1) = XU(YLP1)	SEA0388
GO TO 330	SEA0389
C	SEA0390
C STEP NUMBER 21	SEA0391
C	SEA0392
127 RZSUBT = ZSUBT	SEA0393
RCYLP1 = C(YLP1)	SEA0394
XUT(YLP1) = RZSUBT/RCYLP1	SEA0395
DO 145 JG = 1, LM	SEA0396
IF(A(JG,YLP1).LE.EPS) GO TO 145	SEA0397
IF(ROW(JG).EQ.0) GO TO 135	SEA0398
SUM = 0.0	SEA0399
DO 130 NE = 1, LN	SEA0400
IF(A(JG,NE).GE.0.0) GO TO 130	SEA0401
SUM = SUM + A(JG,NE) * XU(Y(NE))	SEA0402
130 CONTINUE	SEA0403
BT(JG) = BT(JG) - SUM	SEA0404
135 MTEMP = BT(JG)/A(JG,YLP1)	SEA0405
IF(ROW(JG).EQ.0) GO TO 140	SEA0406
BT(JG) = BT(JG) + SUM	SEA0407
140 IF(MTEMP.GE.XUT(YLP1)) GO TO 145	SEA0408
XUT(YLP1) = MTEMP	SEA0409
145 CONTINUE	SEA0410
IF(XUT(YLP1).LE.XU(YLP1)) GO TO 330	SEA0411
XUT(YLP1) = XU(YLP1)	SEA0412
GO TO 330	SEA0413

C		SEA0414
C	STEP NUMBER 22	SEA0415
C		SEA0416
	200 IF(NOTE4 - 1)210,220,230	SEA0417
	210 XLT(YLP1) = XL(YLP1)	SEA0418
	GO TO 330	SEA0419
	220 RZSUBT = ZSUBT	SEA0420
	RCYLP1 = C(YLP1)	SEA0421
	XL YLP1 = RZSUBT/RCYLP1	SEA0422
	NUM4 = XL YLP1	SEA0423
	DELTA4 = NUM4	SEA0424
	GAMMA4 = XL YLP1 - DELTA4	SEA0425
	IF(GAMMA4 - EPS)225,225,230	SEA0426
	225 XLT(YLP1) = XL YLP1	SEA0427
	GO TO 235	SEA0428
	230 TEMXLT = XL YLP1 + 1.0	SEA0429
	XLT(YLP1) = TEMXLT	SEA0430
	235 DO 245 JL = KY,KZ	SEA0431
	IF(A(JL,Y2).LE.EPS) GO TO 245	SEA0432
	IF(ROW(JL).EQ.0) GO TO 237	SEA0433
	SUM = 0.0	SEA0434
	DO 236 NF = 1, LN	SEA0435
	IF(A(JL,NF).GE.0.0) GO TO 236	SEA0436
	SUM = SUM + A(JL,NF) * XL(Y(NF))	SEA0437
	236 CONTINUE	SEA0438
	BT(JL) = BT(JL) - SUM	SEA0439
	237 TEMP5 = BT(JL)/A(JL,YLP1)	SEA0440
	IF(ROW(JL).EQ.0) GO TO 238	SEA0441
	BT(JL) = BT(JL) + SUM	SEA0442
	238 NUM5 = TEMP5	SEA0443
	DELTA5 = NUM5	SEA0444
	GAMMA5 = TEMP5 - DELTA5	SEA0445
	IF(GAMMA5 - EPS)240,240,241	SEA0446
	240 NTEMP5 = TEMP5	SEA0447
	GO TO 242	SEA0448
	241 TEMPY = TEMP5 + 1.0	SEA0449
	NTEMP5 = TEMPY	SEA0450
	242 IF(NTEMP5.LE.XLT(YLP1)) GO TO 245	SEA0451
	XLT(YLP1) = NTEMP5	SEA0452
	245 CONTINUE	SEA0453
	IF(XLT(YLP1).GE.XL(YLP1)) GO TO 330	SEA0454
	XLT(YLP1) = XL(YLP1)	SEA0455
C		SEA0456
C	STEP NUMBER 23	SEA0457
C		SEA0458
C	SET NEXT VARIABLE IN RANKING AT ITS NEW BOUND	SEA0459
C		SEA0460
	330 IF(MAX.EQ.1) GO TO 331	SEA0461
	XTEMP(YLP1) = XLT(YLP1)	SEA0462
	GO TO 340	SEA0463
	331 XTEMP(YLP1) = XUT(YLP1)	SEA0464
C		SEA0465
C	STEP NUMBER 24	SEA0466
C		SEA0467
C	CHECK TO SEE IF A VARIABLE HAS BEEN INCREMENTED BEYOND ITS BOUND	SEA0468
C		SEA0469
	340 LPI = L+1	SEA0470
	YLP1 = Y(L+1)	SEA0471
	IF(MAX.EQ.1) GO TO 341	SEA0472

IF(XTEMP(YLP1).LE.XUT(YLP1)) GO TO 12	SEA0473
XTEMP(YLP1) = XUT(YLP1)	SEA0474
GO TO 342	SEA0475
341 IF(XTEMP(YLP1).GE.XLT(YLP1)) GO TO 12	SEA0476
XTEMP(YLP1) = XLT(YLP1)	SEA0477
C	SEA0478
C STEP NUMBER 25	SEA0479
C	SEA0480
342 IF(L.EQ.1) GO TO 350	SEA0481
L = L-1	SEA0482
YLP1 = Y(L+1)	SEA0483
ZSUBT = ZSUBT + C(YLP1)*XTEMP(YLP1)	SEA0484
RXTEMP = XTEMP(YLP1)	SEA0485
DO 345 JH = 1,LM	SEA0486
BT(JH) = BT(JH) + A(JH,YLP1)*RXTEMP	SEA0487
345 CONTINUE	SEA0488
NOTE4 = 0	SEA0489
LPI = L+1	SEA0490
IF(MAX.EQ.1) GO TO 346	SEA0491
XTEMP(YLP1) = XTEMP(YLP1) + 1	SEA0492
GO TO 340	SEA0493
346 XTEMP(YLP1) = XTEMP(YLP1) - 1	SEA0494
GO TO 340	SEA0495
C	SEA0496
C STEP NUMBER 26	SEA0497
C	SEA0498
C INCREMENT FIRST VARIABLE IN RANKING ONE INTEGER	SEA0499
C	SEA0500
350 IF(MAX.EQ.1) GO TO 351	SEA0501
XTEMP(Y(1)) = XTEMP(Y(1)) + 1	SEA0502
GO TO 352	SEA0503
351 XTEMP(Y(1)) = XTEMP(Y(1)) - 1	SEA0504
352 NOTE1 = 0	SEA0505
NOTE2 = 0	SEA0506
C	SEA0507
C STEP NUMBER 27	SEA0508
C	SEA0509
C CHECK TO SEE IF ALL SOLUTIONS HAVE BEEN EXAMINED	SEA0510
C	SEA0511
IF(MAX.EQ.0) GO TO 365	SEA0512
IF(XTEMP(Y(1)) - XLT(Y(1))) 355,360,360	SEA0513
355 ZOF = ZOF - G	SEA0514
GO TO 4	SEA0515
360 ZSUBT = ZOF	SEA0516
NOTE3 = 0	SEA0517
C	SEA0518
C STEP NUMBER 28	SEA0519
C	SEA0520
IF(LN-2) 12,12,10	SEA0521
365 IF(XTEMP(Y(1)) - XUT(Y(1))) 380,380,375	SEA0522
375 IF(ZOF) 355,355,376	SEA0523
376 ZOF = ZOF + G	SEA0524
GO TO 4	SEA0525
380 ZSUBT = ZOF	SEA0526
NOTE3 = 0	SEA0527
IF(LN-2) 12,12,10	SEA0528
1000 RETURN	SEA0529
END	SEA0530

## APPENDIX F

### TEST PROBLEMS AND SOLUTIONS

$$1. \quad \text{maximize} \quad z = 15 x_1 + 6 x_2 + 9 x_3 + 2 x_4 \quad (\text{F-1})$$

$$\text{subject to} \quad 2 x_1 + 1 x_2 + 5 x_3 + 0.6 x_4 \leq 12.5 \quad (\text{F-2})$$

$$3 x_1 + 1 x_2 + 3 x_3 + 0.25 x_4 \leq 12.6 \quad (\text{F-3})$$

$$7 x_1 + 0 x_2 + 0 x_3 + 1 x_4 \leq 35 \quad (\text{F-4})$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3, 4 \quad (\text{F-5})$$

$$x_j \text{ INTEGER for } j = 1, 2, 3, 4 \quad (\text{F-6})$$

$$\underline{x}^* = (1, 9, 0, 2) \quad (\text{F-7})$$

$$z^* = 73. \quad (\text{F-8})$$

2. From Wagner (30):

$$\text{maximize} \quad z = 3 x_1 + 3 x_2 + 13 x_3 \quad (\text{F-9})$$

$$\text{subject to} \quad -3 x_1 + 6 x_2 + 7 x_3 \leq 8 \quad (\text{F-10})$$

$$6 x_1 - 3 x_2 + 7 x_3 \leq 8 \quad (\text{F-11})$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3 \quad (\text{F-12})$$

$$x_j \text{ INTEGER for } j = 1, 2, 3 \quad (\text{F-13})$$

$$\underline{x}^* = (0, 0, 1) \quad (\text{F-14})$$

$$z^* = 13. \quad (\text{F-15})$$

3. From Gomory (11):

$$\text{maximize} \quad z = 4 x_1 + 5 x_2 + x_3 \quad (\text{F-16})$$

$$3 x_1 + 2 x_2 + 0 x_3 \leq 10 \quad (\text{F-17})$$

$$1 x_1 + 2 x_2 + 0 x_3 \leq 11 \quad (\text{F-18})$$

$$3 x_1 + 3 x_2 + 1 x_3 \leq 13 \quad (\text{F-19})$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3 \quad (\text{F-20})$$

$$x_j \text{ INTEGER for } j = 1, 2, 3 \quad (\text{F-21})$$

$$\underline{x}^* = (2, 2, 1) \quad (\text{F-22})$$

$$z^* = 19. \quad (\text{F-23})$$

4. From Young (31):

$$\text{maximize } z = x_1 + x_2 + x_3 \quad (\text{F-24})$$

$$\text{subject to } -4x_1 + 5x_2 + 2x_3 \leq 4 \quad (\text{F-25})$$

$$-2x_1 + 5x_2 + 0x_3 \leq 5 \quad (\text{F-26})$$

$$3x_1 - 2x_2 + 2x_3 \leq 6 \quad (\text{F-27})$$

$$2x_1 - 5x_2 + 0x_3 \leq 1 \quad (\text{F-28})$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3 \quad (\text{F-29})$$

$$x_j \text{ INTEGER for } j = 1, 2, 3 \quad (\text{F-30})$$

$$\underline{x}^* = (3, 2, 0) \quad (\text{F-31})$$

$$z^* = 5. \quad (\text{F-32})$$

$$5. \text{ minimize } z = 10x_1 + 14x_2 + 21x_3 \quad (\text{F-33})$$

$$\text{subject to } 4x_1 + 4x_2 + 7x_3 \leq 28 \quad (\text{F-34})$$

$$8x_1 + 11x_2 + 9x_3 \geq 12 \quad (\text{F-35})$$

$$2x_1 + 2x_2 + 7x_3 \geq 14 \quad (\text{F-36})$$

$$9x_1 + 6x_2 + 3x_3 \geq 10 \quad (\text{F-37})$$

$$\underline{x}^* = (1, 0, 2) \quad (\text{F-38})$$

$$z^* = 52. \quad (\text{F-39})$$

6. From Cook (5):

$$\text{maximize } z = 1 x_1 - 3 x_2 + 3 x_3 \quad (\text{F-40})$$

$$\text{subject to } 2 x_1 + 1 x_2 - 1 x_3 \leq 4 \quad (\text{F-41})$$

$$4 x_1 - 3 x_2 + 0 x_3 \leq 2 \quad (\text{F-42})$$

$$- 3 x_1 + 2 x_2 + 1 x_3 \leq 3 \quad (\text{F-43})$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3 \quad (\text{F-44})$$

$$x_j \text{ INTEGER for } j = 1, 2, 3 \quad (\text{F-45})$$

$$\underline{x}^* = (2, 2, 5) \quad (\text{F-46})$$

$$z^* = 11. \quad (\text{F-47})$$

7. From Cook (5):

$$\text{maximize } z = 1 x_1 + 2 x_2 + 3 x_3 + 1 x_4 + 1 x_5 \quad (\text{F-48})$$

$$\text{subject to } 1 x_1 + 0 x_2 + 4 x_3 + 2 x_4 + 1 x_5 \leq 41 \quad (\text{F-49})$$

$$4 x_1 + 3 x_2 + 1 x_3 + 0 x_4 - 1 x_5 \leq 147 \quad (\text{F-50})$$

$$x_j \geq 0 \text{ for } j = 1, 2, 3, 4, 5 \quad (\text{F-51})$$

$$x_j \text{ INTEGER for } j = 1, 2, 3, 4, 5 \quad (\text{F-52})$$

$$\underline{x}^* = (0, 42, 0, 19, 3) \quad (\text{F-53})$$

$$z^* = 106. \quad (\text{F-54})$$



8. From Trauth and Woolsey (28):

$$\begin{aligned} \text{maximize } z = & 20 x_1 + 18 x_2 + 17 x_3 + 15 x_4 + 15 x_5 \\ & + 10 x_6 + 5 x_7 + 3 x_8 + x_9 + x_{10} \end{aligned} \quad (\text{F-55})$$

$$\begin{aligned} \text{subject to } & 30 x_1 + 25 x_2 + 20 x_3 + 18 x_4 + 17 x_5 \\ & + 11 x_6 + 5 x_7 + 2 x_8 + x_9 + x_{10} \\ & \leq 55 \end{aligned} \quad (\text{F-56})$$

$$x_j \leq 1 \text{ for } j = 1, 2, \dots, 10 \quad (\text{F-57})$$

$$x_j \geq 0 \text{ for } j = 1, 2, \dots, 10 \quad (\text{F-58})$$

$$x_j \text{ INTEGER for } j = 1, 2, \dots, 10 \quad (\text{F-59})$$

$$\underline{x}^* = (0, 0, 0, 1, 1, 1, 1, 1, 1, 1) \quad \text{or} \quad (\text{F-60})$$

$$\underline{x}^* = (0, 0, 1, 0, 1, 1, 1, 1, 0, 0) \quad (\text{F-61})$$

$$z^* = 50. \quad (\text{F-62})$$

VITA

James Melvin Shirley

Candidate for the Degree of

Doctor of Philosophy

Thesis: AN OBJECTIVE FUNCTION REDUCTION ALGORITHM FOR  
INTEGER LINEAR PROGRAMMING

Major Field: Engineering

Biographical:

Personal Data: Born in Oklahoma City, Oklahoma,  
June 22, 1940, the son of Mr. and Mrs. M. F.  
Shirley.

Education: Attended Central State College, Edmond,  
Oklahoma, from September, 1958 to January, 1961;  
transferred to Oklahoma State University in  
January, 1961; received a Bachelor of Science de-  
gree from Oklahoma State University in May, 1963,  
with a major in Electrical Engineering; attended  
and completed courses at the Federal Aviation  
Administration FAA Academy during 1965, studying  
basic radar techniques and radar bright display  
equipment; attended the University of Oklahoma  
during 1966 and 1967; completed the requirements  
for a Master of Science degree at Oklahoma State  
University in May, 1968; completed requirements  
for Registered Professional Engineer in the State  
of Oklahoma, January, 1971; completed requirements  
for the Doctor of Philosophy degree at Oklahoma  
State University, with a major in Industrial Engi-  
neering and Management, in May, 1972.

Professional Experience: Student engineering trainee,  
Oklahoma Gas and Electric Company, summers 1961  
and 1962; electronic engineer, Federal Aviation  
Administration, communication engineering, 1963,  
radar engineering, 1964 to June, 1968; test set  
engineer, Western Electric Company, Inc., June,  
1968 to September, 1969; research graduate  
assistant, Oklahoma State University, September,

1969 to September, 1971; teaching graduate assistant, Oklahoma State University, September, 1971 to May, 1972.