

INTEGER SEARCH UNDER LINEAR  
DIOPHANTINE CONSTRAINTS

By

ROBERT PRATT DAVIS

Bachelor of Science  
University of Tennessee  
Knoxville, Tennessee  
1970

Master of Science  
University of Tennessee  
Knoxville, Tennessee  
1971

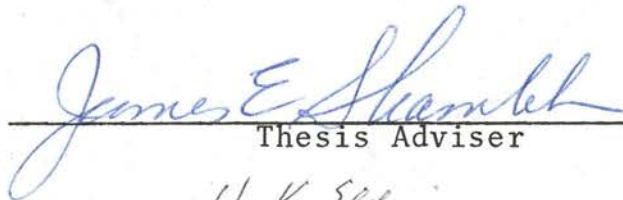
Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
DOCTOR OF PHILOSOPHY  
December, 1973

Thesis  
1973D  
D263i  
Cap. 2

MAR 13 1975

INTEGER SEARCH UNDER LINEAR  
DIOPHANTINE CONSTRAINTS

Thesis Approved:

  
Thesis Adviser







  
Dean of the Graduate College

902059

## ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. James E. Shamblin for his invaluable assistance in the development of this thesis and for his guidance throughout my doctoral studies. To Dr. M. Palmer Terrell and Dr. Donald W. Grace I owe a debt of gratitude for their encouragement and many contributions to my education. I would also like to thank Dr. Hamed K. Eldin for his advice and encouragement.

My deepest gratitude is expressed to my wife, Shirley, without whose understanding, continual sacrifices and support I would never have attained this level of educational development. I thank my parents, Mr. and Mrs. D. G. Davis, Sr., for instilling within me the importance of an education and for their understanding and confidence in me as a student and as a son.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION . . . . .	1
Area of Investigation . . . . .	1
Purpose of the Research . . . . .	3
Format of the Discussion . . . . .	3
Integer Programming Techniques . . . . .	4
II. BASIC IDEAS IN N-DIMENSIONAL GEOMETRY . . . . .	8
Introduction . . . . .	8
Extending the Dimensionality Concept . . . . .	9
Flats or Linear Spaces . . . . .	13
Geometric Configurations--The Simplex . . . . .	17
Geometric Configurations--The Polytope . . . . .	20
Systems of Linear Equations . . . . .	22
Systems of Linear Inequalities . . . . .	25
Nonlinear Functions as Surfaces . . . . .	27
Concluding Remarks . . . . .	30
III. FINDING A FEASIBLE LATTICE POINT . . . . .	32
Lattice Points . . . . .	32
Foundation of the Procedure . . . . .	33
Description of the Procedure . . . . .	34
Cycle and Oscillation . . . . .	36
Example Problem . . . . .	37
Comparison with Other Procedures . . . . .	40
IV. A TIME-SHARED ILP ALGORITHM FOR STRICTLY LIMITED RESOURCE ALLOCATION PROBLEMS . . . . .	43
Foundation of the Procedure . . . . .	44
Description of the Procedure . . . . .	45
Computational Experience . . . . .	47
V. INTEGER SEARCH WITH SIMPLEX DESIGNS . . . . .	52
Integer Nonlinear Programming (INLP) . . . . .	52
Foundation of the Procedure . . . . .	54
Reflecting to Lattice Vertices . . . . .	57
Accelerated Reflections . . . . .	60
Handling Constraints . . . . .	61

Chapter	Page
V. (CONTINUED)	
Stopping Criteria . . . . .	63
Example Problem . . . . .	66
Computational Experience. . . . .	71
VI. CONCLUSIONS AND RECOMMENDATIONS. . . . .	77
Conclusions . . . . .	77
Recommendations . . . . .	80
A SELECTED BIBLIOGRAPHY . . . . .	84
APPENDIX A - CODE FOR FINDING A FEASIBLE LATTICE POINT. . . . .	87
APPENDIX B - CODE FOR IRESAL PROCEDURE. . . . .	91
APPENDIX C - CODE FOR INTEGER SEARCH WITH SIMPLEX DESIGNS. . . . .	96

LIST OF TABLES

Table	Page
I. Corresponding Values of $k$ and $b_k$ . . . . .	48
II. Allocation Test Problem Results . . . . .	49

LIST OF FIGURES

Figure	Page
1. Development of the Dimensionality Concept . . . . .	1
2. Flow Chart for Procedure for Finding a Feasible Lattice Point. . . . .	38
3. Graphical Description of Example Problem. . . . .	39
4. Graphical Description of Modified Example Problem With a Cycle. . . . .	41
5. IRESAL Procedure--Information Flow Chart. . . . .	46
6. Example of Simplex Reflections. . . . .	56
7. Information Flow Chart for Integer Search With Simplex Designs. . . . .	65
8. Integer Search Example. . . . .	67

## CHAPTER I

### INTRODUCTION

#### Area of Investigation

This research is concerned with that class of mathematical programming problems known as integer programming problems. In particular, the class of integer programming problems in which all of the variables are restricted to take on only integral values is examined. Such all-integer programming problems arise from the need for finding an optimum or best policy (solution set) to adopt when confronted with a system model whose variables (or components of the solution set) are meaningful only as integral quantities. For example, if the problem deals with the number of men to be hired or the number of machines to be purchased, then a fractional solution may not be applicable.

The specific topics investigated here are more fully defined by the following characteristics. As a unifying characteristic, this research centers around models which have a constraint set formed from linear diophantine inequalities. That is, the system objective function to be optimized is subject to restrictions which may be



expressed as linear inequalities with integral parameters. Mathematically, the restriction on the feasible solution set may be expressed as follows:

$$\sum_{j=1}^N a_{ij} x_j \leq b_i \quad (i = 1, \dots, M) \quad (1-1)$$

$$x_j \geq 0 \quad (j = 1, \dots, N) \quad (1-2)$$

where  $(a_{ij}, x_j, b_i) \in Z^N$  for all  $(i, j)$ .

Based upon this constraint set, two definable classes of objective functions, or function to be optimized, are examined. First, functions which may themselves be classified as linear diophantine objective functions are examined. Specifically, the resultant problem class is further defined to have the form:

$$\text{Maximize } Z = \sum_{j=1}^N c_j x_j \quad (1-3)$$

subject to the restrictions,

$$\sum_{j=1}^N a_{ij} x_j \leq b_i \quad (i = 1, \dots, M) \quad (1-4)$$

where  $c_j, a_{ij}, b_i, x_j$  are non-negative integers for all  $(i, j)$ . These problems are referred to as resource allocation problems and most frequently arise in the context of capital budgeting or knapsack decisions.

The second type of objective function investigated is defined to be convex and nonlinear. The resultant problem is then referred to as an integer nonlinear programming problem, more specifically defined as:

$$\text{Minimize } Z = f(\underline{x}) \quad (1-5)$$

subject to the restrictions,

$$\sum_{j=1}^N a_{ij} x_j \geq b_i \quad (i = 1, \dots, M) \quad (1-6)$$

$$x_j \geq 0 \quad (j = 1, \dots, N) \quad (1-7)$$

where  $a_{ij}$ ,  $b_i$ ,  $x_j$  are integers for all  $(i, j)$ .

Such problems are less frequently encountered than the first class. However, an example of problems taking this form is the constrained economic order quantity model.

#### Purpose of the Research

Simply stated, this research is an attempt to apply some fundamental concepts from N-dimensional geometry to the development of heuristic solution techniques for the problems classified above. These concepts provide a means for making certain useful observations regarding the feasible domain defined by the constraint set. In addition, they also provide the nucleus upon which the iterative techniques which follow are founded.

#### Format of the Discussion

The discussion of this research will proceed as follows. In Chapter II those fundamental concepts from N-dimensional geometry which provide the basis for this research will be developed. The idea of a feasible integer solution point satisfying a system of linear diophantine inequalities will be presented in Chapter III.

In addition, Chapter III also contains a suggested procedure for finding such a point. Next, in Chapter IV, an algorithm will be presented which is specifically designed to solve that class of problems referred to previously as resource allocation problems. This algorithm may be further characterized by the fact that it was developed for use at a time-shared remote telecommunication terminal. In Chapter V an algorithm will be presented for solving integer programming problems with convex, nonlinear objective functions and subject to linear diophantine inequality constraints. This algorithm, formulated in a minimization context, uses the repeated application of simplex patterns to descend on the optimum integral solution point. Finally, Chapter VI contains the conclusions and recommendations drawn from this research. Appendices are provided which contain the computer codes employed in implementing the algorithms developed. The codes in Appendices A and C were run on an IBM 360/65 in the PLAGO partition of PL/I. The code in Appendix B was run on the same machine in the CPS partition of PL/I.

### Integer Programming Techniques

This discussion centers around those techniques of a general nature which are most commonly associated with the topic of integer programming. Omitted from further discussion are the more specialized tabular methods for

solving transportation and assignment problems and the network analysis techniques oriented toward obtaining the maximum flow in a capacitated network system. More specifically, the techniques of discrete dynamic programming, cutting plane algorithms and implicit enumeration are discussed from a conceptual standpoint.

Dynamic programming is an approach to problem solving. In particular, dynamic programming employs the technique of partitioning the problem into a series of subproblems. These subproblems are then sequentially optimized and ultimately yield the optimum solution to the entire problem. More specifically, the subproblems are referred to as stages. In discrete dynamic programming, the stages are most usually associated with a particular problem variable. At each stage a feasible set of states are given which represent the feasible domain described by the constraints. For each of these states a decision, or stage variable assignment, is made which optimizes the return, or objective, given that input state. A new stage is then added to the problem and the same procedure employed, with the exception that after the first stage, consideration must be made for the state which results from a decision and the corresponding returns so defined from previous stages. The advantage gained by the dynamic programming approach is that once a decision is reached, at a particular stage and for a particular state, the decisions and

returns from previous stages are also established. Unfortunately, dynamic programming suffers from what is referred to as the curse of dimensionality. That is, the number of feasible states and decisions at a given stage can often exceed the capacity of even a large computer.

The cutting plane algorithms are more easily described. In effect, these algorithms begin by solving the linear programming problem while allowing the variables to take on continuous values. If the resultant solution satisfies the integrality restrictions, then the procedure, of course, terminates. Otherwise a new constraint, developed from the original constraints, is appended to the problem and a new solution obtained. The effect of this constraint is to cut off that part of the feasible domain containing the current non-integral solution, but not cutting off an optimum integral solution. The process of adding new constraints, or cutting planes, is repeated until the optimum integral solution is obtained.

The concept of implicit enumeration or branch-and-bound is essentially a tree search methodology. This approach begins by building a tree constituted from branches associated with the feasible integral values of the problem variables. In this context, a solution set is said to be fathomed when a feasible solution value is established for each problem variable and the resultant

objective function value thereby determined. In effect, all possible combinations combinations of solution values are implicitly examined by either determining them to be infeasible or yielding an objective function value less favorable than an explicitly fathomed solution set. In other words, various possible solution sets are strategically fathomed until one is found which cannot be improved upon by the further explicit determination of any other possible set. Of the techniques here described, implicit enumeration has been found to demonstrate, in general, computational superiority when applied to large problems (9).

## CHAPTER II

### BASIC IDEAS IN N-DIMENSIONAL GEOMETRY

#### Introduction

From the standpoint of the individual observer, the concept of geometry arises through an intuitive development of the senses of sight and touch. The fundamental ideas of shape, bulk and length are analysed and refined, leading eventually to the conception of geometric figures. In like fashion, the history of the study of geometric concepts developed. First came the concept of a solid and from this, the abstractions of surfaces and lines, without solidity, developed. These abstractions from solidity led, after much refinement, to the development of plane geometry. As a consequence of the recognition of the point, line and plane as existing entities in a three dimensional universe, the concept of "dimensionality" itself arose. However, many centuries had passed before the plane and solid geometry of the Greeks and Egyptians gave way to an upward extension of the dimensionality concept to N-dimensional space.

There are essentially two approaches to the development of an understanding of the geometry of higher

dimensions. On the one hand, there is the approach of extending the elements of point, line and plane in a serial fashion to higher dimensions. On the other hand, there is the approach of interpreting algebraic expressions through geometric concepts. Initially here, the first approach will be used to formulate an intuitive understanding of the fundamental concepts. These concepts will then be reinforced and extended by utilizing them as a basis for interpreting and understanding algebraic relationships in a geometric context. The purpose of this discussion is to enhance the ability of the individual observer to accomplish this interpretation and understanding.

#### Extending the Dimensionality Concept

As is customary, certain undefinable entities are taken to exist and from these the fundamental axioms (unproved propositions) of geometry are stated. These undefinables are the point, line and plane. The awareness and recognition of these being supposed, the following axioms are given:

Axiom 1: Any two distinct points uniquely determine a straight line.

Axiom 2: If two distinct points determine a straight line, then a third point exists which does not lie on this line.

Axiom 3: Any three non-collinear points determine a plane.



Axiom 4: If two distinct points both belong to a plane, then every point on the line determined by these points also lies in the plane.

Axiom 5: If three non-collinear points exist to determine a plane, there also exists a fourth point not in that plane.

Axiom 6: The intersection of any two distinct lines in a plane uniquely determines a point at that intersection.

Axiom 7: If two distinct planes have a point in common, they have a second point in common and consequently intersect in a straight line.

These axioms provide the foundation for the concept of three dimensional space and thereby solid geometry.

Observe that Axiom 1 postulates the existence of one dimensional space, Axiom 2 postulates two dimensional space and finally Axiom 5 postulates the existence of three dimensional (solid) space. This foundation being established, an attempt is now made to extend these concepts straightaway to higher dimensions.

Given four non-coplanar points, it may be observed that all of the points, lines and planes therefrom determined constitute a three dimensional region, or the familiar three-space of solid geometry. Assume now that a point exists which is not in this region, and consequently the preceding three dimensional region is not now the whole of space. The region constituted by

any four of the five points now postulated is called a "hyperplane" lying in a "hyperspace," or space whose dimensionality exceeds three (in this case, a four dimensional hyperspace). Taking these five points and all the lines, planes and hyperplanes thereby constituted, the following statements may be shown to be true in this four dimensional region.

1. Two hyperplanes intersect in a plane.
2. Three hyperplanes intersect in a line.
3. Four hyperplanes intersect in a point.
4. In general, five hyperplanes do not necessarily have any point in common.
5. A hyperplane intersects a plane (the intersection of two hyperplanes) in a line.
6. A hyperplane intersects a line in a point.
7. Two planes, each the intersection of two hyperplanes, have in general only one point in common.
8. A plane and a line have in general no point in common.
9. A hyperplane is constituted by not only four distinct non-coplanar points, but also by a plane and a point not in the plane or by two skew lines.

In an attempt to develop a better intuitive feel for the dimensionality concept it is often beneficial to relate dimensionality to the "degree of freedom" of a

point in a region of specified dimension. By way of example, consider some point in a hyperspace of four dimensions. If the point is required to lie on a given line, a one dimensional entity, the point is said to possess one degree of freedom in that it may lie anywhere on that line. Similarly, a point required to lie on a given plane, a two dimensional entity, is said to have two degrees of freedom. Continuing upward, a point required to lie in a particular hyperplane has three degrees of freedom; and in the four dimensional hyperspace, four degrees of freedom are available.

Considering the concept of degrees of freedom now from the restrictive standpoint, it may be observed that, in a four dimensional hyperspace, no conditions or constraints are required for a point to exist in that space (i.e., the point, as before, has four degrees of freedom). However, for a point to lie in a given hyperplane, one degree of constraint or one condition is required thereby reducing the degrees of freedom to three. Similarly, for a point to lie in a given plane two conditions or degrees of constraint are necessary thereby reducing the degrees of freedom to two. Finally, as one might suspect, if the point is required to lie in two distinct planes simultaneously, four conditions are required and the point is thereby uniquely determined (i.e., it now possesses no degrees of freedom).

## Flats or Linear Spaces

Having now made the extension from a three dimensional space to space of four dimensions, the succeeding extension to higher dimensions follows directly. In an attempt to achieve greater generality, the term "hyperplane" will be extended to include not only the three dimensional analogue of a plane in four space but any space of "n" dimensions where n is greater than or equal to three. Note that this will require a means of explicitly distinguishing, for example, a three dimensional hyperplane from one of four dimensions (see Figure 1). This distinction is accomplished by referring to a hyperplane of "p" dimensions as a "p-flat." A "flat" space is also referred to as a "linear space."

The series of regions point, line, plane, three-flat, . . . ., n-flat are then determined respectively by one, two, three, four, . . . ., n+1 distinct points; and have correspondingly zero, one, two, three, . . . ., n dimensions. Consequently, given n+1 points which determine an n-flat, there exists a p-flat which lies entirely in the n-flat and is determined by any p+1 of the n+1 given points (for  $p < n$ ).

The points which have been taken to uniquely determine a region exhibit a specific characteristic, that being "linear independence." Observe that if n+1 points are to uniquely determine an n-flat, then they

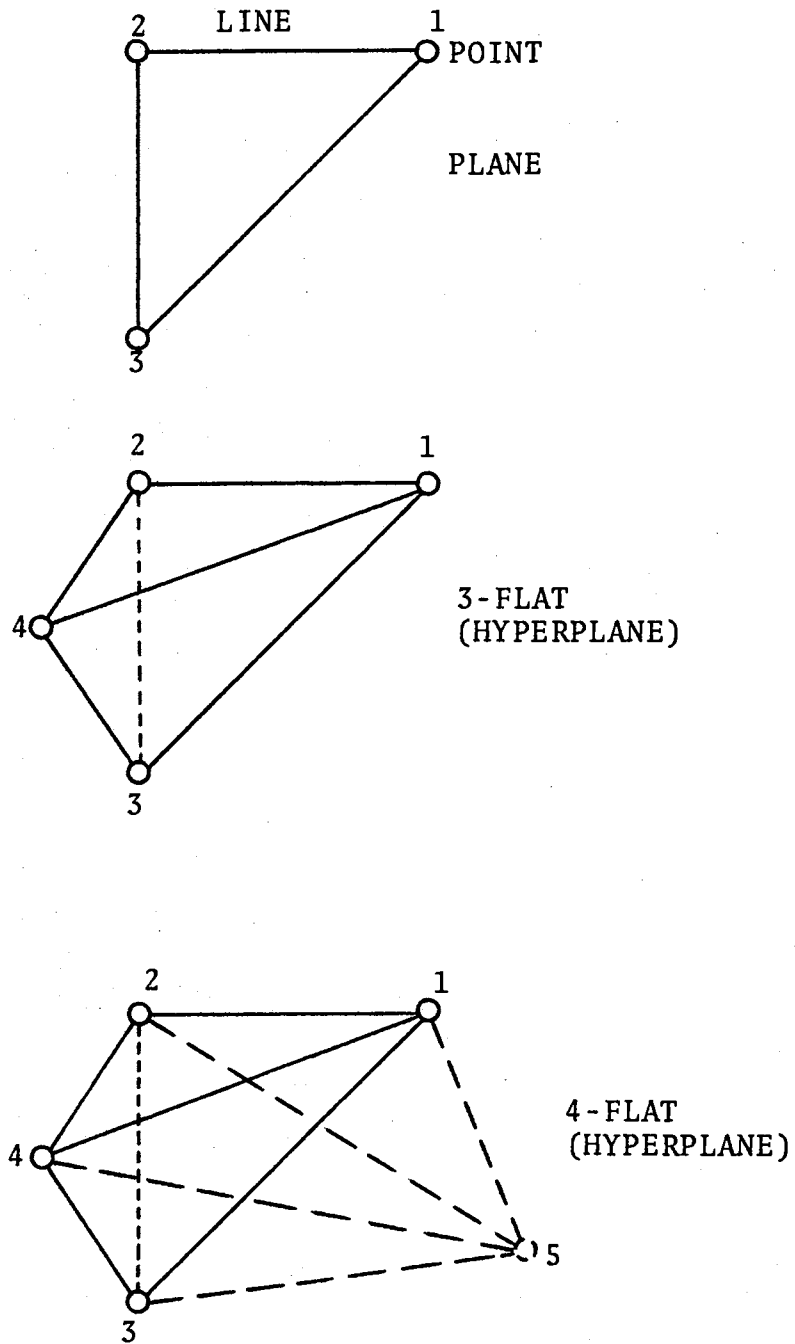


Figure 1. Development of the Dimensionality Concept

must not be contained in the same  $(n-1)$ -flat. In addition, no " $p$ " of these points ( $p \leq n$ ) may be contained in the same  $(p-2)$ -flat. If this were not true, then the  $p-1$  points required to determine the  $(p-2)$ -flat, together with the remaining  $n+1-p$  points would uniquely determine an  $(n-1)$ -flat. A system of  $n+1$  points, no  $p$  of which lie in the same  $(p-2)$ -flat, is referred to as a system of linearly independent points. It may further be stated that any  $n+1$  points of an  $n$ -flat, if they are linearly independent, can be taken to uniquely determine the  $n$ -flat.

This characteristic of linear independence leads directly to the following important observations. Given a  $p$ -flat and an  $r$ -flat which are determined respectively by  $p+1$  and  $r+1$  points, if they have no points in common then there are in total  $p+r+2$  independent points which determine a  $(p+r+1)$ -flat. Therefore, any  $p$ -flat and  $r$ -flat taken arbitrarily must lie in the same  $(p+r+1)$ -flat. However, if these flats lie in an  $n$  dimensional space and  $p+r+1$  is greater than  $n$ , then the two flats must have a region in common. Assume that this common region is of dimension " $s$ ". It may then be stated that, a  $p$ -flat and an  $r$ -flat having in common an  $s$ -flat are both then contained in a  $(p+r-s)$ -flat. Furthermore, a  $p$ -flat and an  $r$ -flat which are both contained in an  $n$ -flat (where  $p+r > n-1$ ) have in common a  $(p+r-n)$ -flat; and if  $p+r < n$ , they have in general no point in common.

It now remains to tie the concept of degrees of freedom to that of a linear space. Recall that a  $p$ -flat requires  $p+1$  independent points to determine it, and each of these points requires  $n$  conditions to determine it in an " $n$  dimensional" space. Observe, however, that  $p$  degrees of freedom are available in the selection of each point. It may then be concluded that the number of conditions required to determine a  $p$ -flat in a space of  $n$ -dimensions is  $(p+1)(n-p)$ . In other words, the number of degrees of freedom available to a  $p$ -flat lying in an  $n$ -flat is  $(p+1)(n-p)$  where  $n > p$ . This number is referred to as the "constant number" of the  $p$ -flat. To further extend this concept, observe that if a  $p$ -flat has  $r$  points already determined, then  $p+1-r$  points are required to uniquely determine it; and consequently, the number of degrees of freedom is  $(n-p)(p+1-r)$ . Therefore, the number of degrees of freedom of a  $p$ -flat lying in a given  $n$ -flat and passing through a given  $r$ -flat is  $(n-p)(p+1-r)$ . From this is obtained a fractional representation of the "degree of incidence" of a  $p$ -flat and an  $s$ -flat, where it is assumed that  $p > s$ . When there is complete incidence, or enclosure, the  $s$ -flat lies entirely in the  $p$ -flat and the fraction is unity. Conversely, skewness, or no points in common is represented by zero. Intermediately, if the  $p$ -flat and the  $s$ -flat have in common an  $r$ -flat, then the degree of incidence is given by the fraction  $(r+1)/(s+1)$ .

Recall that in an  $n$  dimensional space, a  $p$ -flat has  $(n-p)(p+1)$  degrees of freedom; but if it passes through a given  $r$ -flat it has only  $(n-p)(p-r)$  degrees of freedom. From this it is observed that the number of conditions required for a  $p$ -flat in  $n$  dimensional space to pass through a given  $r$ -flat is  $(n-p)(r+1)$ , where  $n > p > r$ . Consequently, if the  $r$ -flat is free to move in a given  $s$ -flat, it has  $(r+1)(s-r)$  degrees of freedom. Therefore, the number of conditions necessary for a  $p$ -flat and an  $s$ -flat in  $n$  dimensional space to intersect in an  $r$ -flat is  $(r+1)(n-p-s+r)$ , provided that  $p+s \leq n+r$ . If  $p+s > n+r$  the  $p$ -flat and  $s$ -flat intersect in a region of dimension  $p+s-n$  which is greater than  $r$ .

### Geometric Configurations -- The Simplex

Having developed a fundamental understanding of dimensionality concepts, attention will now be turned to configurations which exist in higher dimensional space. To begin, the following observations are made:

1. A point on a line divides the line into two segments but will not divide a plane in which the line lies.
2. A line in a plane will divide the plane but will not divide a three-flat in which the plane lies.
3. A plane in a three-flat will divide the three-flat but will not divide a four-flat in which



the three flat lies.

These observations may be extended in like fashion to higher dimensions and lead directly to the idea of geometric order.

If  $A_1$  and  $A_2$  are two distinct points on a given line, they constitute a "line segment" consisting of all points  $P$  such that their order is  $A_1PA_2$ . In addition,  $A_1$  and  $A_2$  also divide the given line into three distinct segments having respectively the order  $PA_1A_2$ ,  $A_1PA_2$  and  $A_1A_2P$ .

If  $A_1$ ,  $A_2$  and  $A_3$  are three non-collinear points in a plane, they constitute three distinct lines which form a triangle. Taking a particular one of these lines, say  $A_2A_3$ , and a point on the line segment thereby determined,  $P_{23}$ , it may be observed that all points  $P$  in the interior of the triangle have the order  $A_1PP_{23}$ . By extension it is observed that these three lines will divide the plane into seven regions as follows:

1. the interior of the triangle:  $A_1PP_{23}$ ,  
 $A_2PP_{13}$ ,  $A_3PP_{12}$ .
2. three regions on the edges:  $A_1P_{23}P$ ,  
 $A_2P_{13}P$ ,  $A_3P_{12}P$ .
3. three regions at the vertices:  $PA_1P_{23}$ ,  
 $PA_2P_{13}$ ,  $PA_3P_{12}$ .

Similarly four non-coplanar points  $A_1$ ,  $A_2$ ,  $A_3$ , and  $A_4$  determine four planes and six lines which constitute a tetrahedron. The faces of the tetrahedron are the triangles formed by any three of the four points. The

four planes corresponding to these triangles divide the space into fifteen regions consisting of points  $P$  having the following orders (here for example,  $P_{123}$  denotes any point on the face  $A_1A_2A_3$ ):

1. the interior of the tetrahedron:  $A_1PP_{234}$ , etc.
2. four regions on the faces:  $A_1P_{234}P$ , etc.
3. four regions on the vertices:  $P_{234}A_1P$ , etc.
4. six regions on the edges:  $P_{12}P_{34}P$ , etc.

Extending to a space of four dimensions, there are then five points which determine five hyperplanes, ten planes and ten edges which collectively form a four dimensional "simplex" dividing space into thirty-one regions:

1. the interior of the simplex.
2. five regions on the three dimensional boundaries.
3. ten regions on the two dimensional boundaries.
4. ten regions on the edges.
5. five regions at the vertices.

Finally the extension to  $n$  dimensions is as follows. The configuration formed by  $n+1$  independent points and the lines, planes and hyperplanes thereby determined is called a "simplex of  $n$  dimensions," denoted  $S_{(n+1)}$ . The lines, planes and hyperplanes determined by these points are called the boundaries of the simplex, and are of one, two, . . . .,  $n-1$  dimensions (with the constituting points being referred to as its vertices). It may then be generalized that a simplex,  $S_{(n+1)}$ ,

has the following number of boundaries of  $r$  dimension,  
 $n+1 \binom{C}{r+1}$ .

### Geometric Configurations--The Polytope

Having now introduced the idea of a geometric configuration bounded by linear spaces, the question now arises as to configurations which may be bounded by several other configurations of smaller dimension. For example, consider the polygon in two dimensions which is bounded by several lines and has consequently several vertices. Analogously, the polyhedron in three dimensions is bounded by several lines and planes. As before, the analogy continues to higher dimensions with the configuration being referred to as a "polytope" bounded by hyperplanes, planes and lines. The following properties may now be stated as regards a polytope:

1. Adjacent hyperplanes meet in boundaries of  $n-2$  dimensions and in general only two hyperplanes meet in each boundary of  $n-2$  dimensions.
2. Three or more  $(n-1)$ -flats meet in boundaries of  $n-3$  dimensions.
3.  $p$  or more  $(n-1)$ -flats meet in boundaries of  $n-p$  dimensions.
4.  $n$  or more  $(n-1)$ -flats meet at a point, one of the vertices of the polytope.
5. A boundary of  $r$  dimension is referred to as

an  $r$ -boundary.

It should now be apparent that the simplest polytope that can exist in an  $n$  dimensional space is the simplex,  $S_{(n+1)}$ , which is bounded by  $n+1$  hyperplanes of dimension  $n-1$ . To clarify this point consider the triangle in two dimensions which is bounded by three lines, the tetrahedron in three dimensions bounded by four planes and finally the four dimensional simplex bounded by five three dimensional tetrahedrons.

Although there exist several classifications or characteristics of polytopes, this discussion will address that particular class of polytopes known as "simple, convex polytopes." A "simple" polytope is characterized by the fact that two and only two boundaries of dimension  $(n-1)$  meet at each boundary of dimension  $(n-2)$ ; and in general within any boundary of dimension  $p$ , two and only two boundaries of dimension  $(p-1)$  meet at each boundary of dimension  $(p-2)$ . Furthermore, a polytope is said to be "convex" if it lies entirely to one side of each of its boundaries having dimension  $(n-1)$ . In other words the polytope and only the polytope is entirely closed within these  $(n-1)$  dimensional boundaries. This characteristic is also true for each of its boundary configurations of any dimension.

For simplicity a simple, convex polytope of  $n$  dimensions is denoted  $(Po)_n$ . The configurations which

form the exterior boundaries of  $(Po)_n$  (i.e.,  $(Po)_{n-1}$ ,  $(Po)_{n-2}$ , . . . ., planes) are termed the "face constituents" of the polytope. As an example, consider the four dimensional simplex,  $(Po)_4$ . It is observed that this simplex is bounded by four three dimensional tetrahedrons and six bounding planes which collectively form the face constituents of the polytope,  $(Po)_4$ .

Having now discussed, at some length, the abstracted but fundamental ideas of N-dimensional geometry, the attention now turns to the second approach for understanding these concepts. Recall that this approach is one of attempting to interpret and understand algebraic expressions through the geometric concepts which have been discussed.

### Systems of Linear Equations

Following a similar approach to preceding sections, the interpretation of systems of linear equations will begin with the point definition and proceed to build upon this to equations of higher dimensionality. This succession is by no means obtuse, but is a necessary prerequisite to the ultimate goal of interpreting systems of linear equations and inequalities.

There are two fundamental concepts which provide the basis upon which the interpretation of linear equations is founded. First, the dimensionality of the space under consideration is given by the number of

unique variables contained in the equation. Second, an equation represents a condition which must be satisfied by a point in the defined space and as such reduces the "degrees of freedom" of a point in that space by one.

Consider the following three equations:

$$x_1 = k_1 \quad (2-1)$$

$$x_1 + x_2 = k_2 \quad (2-2)$$

$$x_1 + x_2 + x_3 + \dots + x_N = k_N \quad (2-3)$$

where  $k_1$ ,  $k_2$ ,  $k_N$  and  $N$  are known constants.

Utilizing these two concepts, it is observed that equation (2-1) represents a point (or region of zero dimension), equation (2-2) a line (or region of one dimension) and finally equation (2-3) represents an  $N-1$ -flat (or region of  $N-1$  dimension). Note that in each instance, as indicated previously, the dimensionality of the space was defined by the number of unique variables and the region defined by each expression had for its dimensionality one less than that of the space to reflect the remaining degrees of freedom available. As a result, three distinct regions have been defined each lying in a space of known dimension and consequently defining a condition which must be met by any admissible point in that space.

It now remains to combine each of these equations into a system or collection of linear equations. Taken collectively, it is observed that  $x_1$  and  $x_2$  are not now uniquely represented in the system but are indicated

three and two times respectively. Consequently, the system defined represents in total a space of  $N$  dimension. Observe that three conditions are now specified and therefore the region defined is of dimension  $N-3$  (i.e., an  $N-3$ -flat). This is more readily observed if the following steps are taken. First,  $x_1$  may be removed from the system by noting that its value is fixed at  $k_1$ . Similarly, by removing  $x_1$  the value of  $x_2$  is fixed and it too may be removed. What remains is then equation (2-3) with two variables removed and a new constant value on the right hand side of the equation,

$$x_3 + \dots + x_N = (k_N - k_2 + k_1 - k_1) = k_p \quad (2-4)$$

The adjustment from  $k_N$  to  $k_p$  and elimination of  $x_1$  and  $x_2$  from consideration forces equations (2-1) and (2-2) to be satisfied. Equation (2-4) is readily observed to define a space of  $N-2$  dimensions with one condition specified. Again, a region of  $N-3$  dimensions is defined.

To generalize, it may be stated that a system of linear equations containing  $N$  unique variables and  $P$  distinct equations represents a region of dimension  $N-P$  (i.e., an  $N-P$ -flat) lying in a space of dimension  $N$ ; and further, that as each variable is allowed to take on a specific value (as with equation (2-1)) the dimensionality of the space is reduced by the specific removal of one of its constituents.

## Systems of Linear Inequalities

A linear inequality is represented in similar form to a linear equation with the exception that the equality sign is replaced by one of the following:

1. "<" -- strictly less than
2. " $\leq$ " -- less than or equal to
3. ">" -- strictly greater than
4. " $\geq$ " -- greater than or equal to
5. " $\neq$ " -- not equal to

The relationships to be discussed here are (2.) and (4.), that is, the "loose" inequalities. Such relationships are treated in the same fashion as equalities after one simple modification.

The inequalities (2.) and (4.) may be made to appear as equalities by the addition of "dummy" or pseudo-variables, so named because they are mathematical conveniences and do not specifically identify with the real space defined by the system (even though they are treated as if they do). This is accomplished as follows.

For " $\leq$ " inequalities a non-negative dummy variable is appended to form an equation as,

$$\sum_{j=1}^N a_j x_j \leq k \quad (2-5)$$

becomes

$$\sum_{j=1}^N a_j x_j + s = k \quad (2-6)$$



where  $s$  is the dummy or "slack variable." For ">" inequalities,

$$\sum_{j=1}^N a_j x_j \geq k \quad (2-7)$$

becomes

$$\sum_{j=1}^N a_j x_j - s = k \quad (2-8)$$

where the dummy variable,  $s$ , represents a "surplus variable." Again,  $S$  is non-negative.

It is apparent that, when confronted with such a loose inequality, the number of degrees of freedom in the defined space is unchanged. That is, one condition is specified by each resultant equation but this is offset by the corresponding addition of another constituent to the defined space. Therefore, inequalities such as (2-5) and (2-7) represent a space of  $N+1$  dimensions in which one condition is specified; and consequently a region of dimension  $N$  is defined (i.e.,  $N$  real space dimensions + one pseudo space dimension less the one specified condition).

This approach may be extended directly to systems of linear inequalities. Considering the following such system, it may be observed that it represents a space of  $N$  real plus  $P$  pseudo dimensions in which an  $N$ -flat has been defined.

$$\begin{aligned}
 \sum_{j=1}^N a_{1j} x_j + s_1 &= k_1 \\
 &\vdots \\
 &\vdots \\
 &\vdots \\
 \sum_{j=1}^N a_{pj} x_j + s_p &= k_p
 \end{aligned}
 \tag{2-9}$$

To generalize, it may be stated that a system of  $M$  linear equations and  $P$  linear inequalities (of the form (b) or (d)) in  $N$  uniquely represented variables constitutes a space of  $N+P$  dimensions in which a region of dimension  $N-M+P$  has been defined (i.e., an  $N-M+P$ -flat).

#### Nonlinear Functions as Surfaces

Conspicuously absent from the discussion thus far has been the topic of nonlinear functions and equations. There are two reasons for postponing a discussion of this topic till now. First, it is believed that the interpretation of such functions and equations is best accomplished by interpreting algebraic relationships rather than abstracted generalizations. Second, and more importantly, this research centers around systems of linear inequalities as boundary specifications of some feasible domain and is concerned with nonlinear functions as representing surface responses superimposed on such domains.

The purpose of this section is to develop an

understanding of nonlinear functions of higher dimension. More specifically, it is the dimensionality aspect which will receive the attention and not a discussion of nonlinear surfaces in general. It will be assumed the functions are convex. To accomplish an understanding of convex nonlinear functions of higher dimension a simplex function of this class will be employed as a medium for developing the technique. This technique is equally applicable to any such function. The development will proceed in similar fashion to preceding sections in this chapter.

To begin consider the simple nonlinear function,

$$f = x_1^2 \quad (2-10)$$

which represents a parabola centered at the origin.

It is noted that such an equation represents a functional relationship between the values taken by  $f$  or by  $x_1$

when one or the other of these variables is specified.

The relationship, in general, defines a locus of points along a curved path (the parabola) which in this instance has one degree of freedom. However, unlike a linear equation, when the value of  $f$  is specified, the degrees of freedom are not necessarily reduced to zero. To continue, another second degree variable is added to equation (2-10) and the equation,

$$f = x_1^2 + x_2^2 \quad (2-11)$$

is obtained. Here the concept of a surface begins to take some meaning. As before, the three unique variables are

indicative of a space of three dimensions. Observe again that only  $f$  is uniquely specified when values are assigned to the variables  $x_1$  and  $x_2$ . Recall that some limited restriction does exist on the value of  $x_1$  or  $x_2$  when one or the other is specified along with  $f$ , or when  $f$  alone is specified at some value. This restriction is, of course, the number of possible roots which satisfy the resultant equation. The variable  $f$  will now be concentrated on to complete the development of a surface concept.

It is apparent that when the value of  $f$  is fixed, (2-11) becomes the equation of a circle. Further, if  $f$  is allowed to take on several values sequentially, then each has a different and unique circle (defined by (2-11)) associated with it. These circles may be viewed stacked one upon the other in the  $f$  direction to form a paraboloid--a surface (or family of circles).

Continuing in an upward fashion it is observed that the addition of another variable,  $x_3$ , to (2-11) to form,

$$f = x_1^2 + x_2^2 + x_3^2 \quad (2-12)$$

yields a hypersurface of spheroids dependent for their exact size upon the value of  $f$ . The analogy to higher dimensions is straightforward. That is, the functional representation,

$$f = x_1^2 + x_2^2 + x_3^2 + \dots + x_N^2 \quad (2-13)$$

designates a hypersurface of hyperspheroids. Note that by fixing the value of  $f$  and sequentially fixing the

value of each  $x_i$  ( $i=N, \dots, 2$ ), that the defined surface gradually loses its abstraction and becomes discernable as a more familiar geometric configuration.

In general, any nonlinear function,

$$f(x_1, x_2, \dots, x_N)$$

may be interpreted most readily by assuming first that the value of  $f$  is fixed and then sequentially fixing subsets of the constituent  $x_i$  ( $i=1, \dots, N$ ) to determine the exact configuration of the remaining variables in their constituent space. Then by allowing  $f$  to assume other values, the corresponding surface defined by  $f(x_1, x_2, \dots, x_N)$  may be viewed more readily by its traces in the constituent space.

#### Concluding Remarks

This chapter is an attempt to lay the foundation upon which the succeeding research is built. Its purpose is to provide the geometric framework for interpreting and understanding the terminology and concepts which follow. Although there exists an area of mathematics known as "the geometry of numbers" which deals specifically with lattice configurations, integer equations and inequalities and their resultant geometry, the position taken here is that these elegant mathematical theorems and postulates are not necessary to the understanding or development of this research. In fact, their inclusion would rather retard this attempt to enhance

the reader's ability to visualize the techniques which are employed. What is hoped for here is that some old, familiar concepts will be recalled, extended and later utilized as a basis for understanding the research.

This chapter is based principally on the works of Sommerville (28) and Grunbaum (14) with graphical interpretations based upon Woodworth (36).

## CHAPTER III

### FINDING A FEASIBLE LATTICE POINT

Many integer programming techniques are enhanced by the presence of an initial starting point (vector) which is feasible and contains all integer components.

This chapter develops a technique for finding such a feasible integral solution point that satisfies a system of linear diophantine inequality constraints.

#### Lattice Points

A lattice point is defined to be any point in the real vector space,  $R^N$ , whose vector representation,

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix}$$

is comprised of components,  $x_i$  ( $i=1, 2, \dots, N$ ) whose values are all integer. A feasible lattice point is one which satisfies the system of linear diophantine inequalities,

$$\sum_{j=1}^N a_{ij} x_j \geq b_i \quad (i=1, \dots, M) \quad (3-1)$$

and the non-negativity restrictions,

$$x_j \geq 0 \quad (j = 1, \dots, N). \quad (3-2)$$

Such a feasible lattice point, or integral solution vector, represents a point on or within the bounding polytope,  $(Po)_N$ , defined by (3-1) and (3-2).

The purpose of this discussion is to suggest a procedure for finding such a point. The procedure rests upon a fundamental theorem from linear algebra and its consequent geometric interpretation.

#### Foundation of the Procedure

This procedure finds its basis in the following fundamental theorem from linear algebra (30).

**Theorem:** If  $\underline{\alpha}_1, \underline{\alpha}_2, \dots, \underline{\alpha}_M$  are vectors whose elements belong to  $\mathcal{F}$ , the set of all linear combinations,  $C_1 \underline{\alpha}_1 + C_2 \underline{\alpha}_2 + \dots + C_M \underline{\alpha}_M$  for  $C_i$  ( $i=1, \dots, M$ ) in  $\mathcal{F}$ , is a linear vector space. The vectors  $\underline{\alpha}_1, \underline{\alpha}_2, \dots, \underline{\alpha}_M$  are said to span or generate the linear vector space (i.e., any vector in the space can be written as a finite sum of the spanning vectors).

Recalling now the non-negativity restrictions (3-2), it is readily observed that the vector representation of their coefficients,

$$\underline{\alpha}_1^T = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \underline{\alpha}_2^T = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots, \quad \underline{\alpha}_N^T = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

in fact span the feasible orthant of non-negative



lattice points (i.e., the positive  $Z^N$ ). More importantly, since the spanning set is not restricted to include only the spanning vectors, the constraints (3-1) may also be included as,

$$\underline{\alpha}_{N+1}^T = \begin{bmatrix} a_{11} \\ \vdots \\ a_{1N} \end{bmatrix}, \quad \underline{\alpha}_{N+2}^T = \begin{bmatrix} a_{21} \\ \vdots \\ a_{2N} \end{bmatrix}, \quad \dots, \quad \underline{\alpha}_{N+M}^T = \begin{bmatrix} a_{M1} \\ \vdots \\ a_{MN} \end{bmatrix} .$$

Note that by writing the constraints in greater-than or equal-to form, the resultant vectors are normal to their corresponding constraints and point inward to the feasible domain defined by the constraint. It is this geometric interpretation which provides the framework from which the succeeding procedural description arises.

#### Description of the Procedure

Define the system of linear diophantine inequalities which form the bounding polytope as:

$$S: \underline{\alpha}_i \underline{x} \geq b_i \quad (i = 1, \dots, N + M) \quad (3-3)$$

where, again,  $\underline{\alpha}_i$  represents the components of a normal vector to the hyperplane defined by  $\underline{\alpha}_i \underline{x} = b_i$  and pointing inward to the feasible domain defined by the corresponding inequality. When, for any particular integral vector  $\underline{x}_0$ , the values  $b_i - \underline{\alpha}_i \underline{x}_0 \leq 0$  ( $i = 1, \dots, N + M$ ), then  $\underline{x}_0$  represents a feasible lattice point. It may further be observed that the value,

$$V_i = b_i - \underline{\alpha}_i \underline{x}_0 > 0 \quad (3-4)$$

is a measure of the degree of infeasibility of  $\underline{x}_0$  relative

to the  $i^{\text{th}}$  constraint. Consequently, a movement in the direction defined by  $\underline{\alpha}_i$  may be interpreted as necessary to obtain a feasible lattice point. In addition, the  $V_i$  indicating infeasibility may be ranked from greatest to smallest, with the greatest being interpreted as indicating the most desirable movement.

It is apparent that only the  $V_i$  corresponding to constraints  $N+1$  to  $N+M$  need be considered explicitly since a resultant  $\underline{x}$  may be examined at each iteration to determine if it contains a negative element. If a negative element, say  $x_j$ , appears then the non-negativity vector,  $\underline{\alpha}_j$ , is implicitly invoked by setting  $x_j$  equal to zero. In this fashion the non-negativity restrictions are kept inviolate.

Taking now that the vectors  $\underline{\alpha}_i$  ( $i=1, \dots, N+M$ ) determine directions of feasible movement and that they also span the feasible space, it may be concluded that a feasible lattice point is obtainable by summing the  $\underline{\alpha}_i$  indicated by the greatest current  $V_i$  until all  $V_i \leq 0$ . On the surface this may appear quite simple, however there are obvious difficulties to such an approach. The major difficulty with this procedure is that the finite number of summing operations required to obtain a feasible lattice point may be quite large. That is, finiteness is no guarantee of solution convergence in a reasonable number of iterations. Consequently, a parameter limiting the number of iterations must be included. In addition, the

procedure must be able to establish that no feasible lattice point exists on or within the defined polytope. In this regard, it is necessary to define the terms "cycle" and "oscillation" as they apply to this procedure.

### Cycle and Oscillation

Define the term "cycle" to be the occurrence of obtaining an intermediate point,  $\underline{x}_k$ , and after ( $n > 1$ ) succeeding iterations, reobtaining the same point,  $\underline{x}_k = \underline{x}_{k+n}$ . Since for any point,  $\underline{x}$ , there is taken some maximum  $V_i$  indicating the direction,  $\underline{\alpha}_i$ , for movement; then if  $\underline{x}$  is reobtained this cyclic occurrence will continually be repeated and no feasible point obtained. It is observed, however, that with parallel constraints for example, it is possible to "oscillate" across the feasible space (i.e., one intervening iteration between obtaining  $\underline{x}_k$  and reobtaining it,  $\underline{x}_k = \underline{x}_{k+2}$ ) when in fact a feasible point may be present. The number of iterations which occur before a trial point is reobtained is then the differentiating factor between an oscillation and a cycle.

In the case of a cycle, the procedure terminates indicating no feasible lattice point. If an oscillation should be detected, the following steps are taken. Define the weighting factor,

$$w_j = \sum_{i=N+1}^{N+M} \alpha_{ij}, \text{ for } \alpha_{ij} < 0 \text{ (j=1, \dots, N)}. \quad (3-5)$$

Then let the element  $x_k$  of the current trial vector be modified to,

$$x_k = x_k + V_i \quad (3-6)$$

where  $i$  corresponds to the maximum  $V_i$  and  $k$  corresponds to the maximum element of  $\underline{w}$ . In effect a quasi dual-simplexing step is taken to eliminate the oscillation and the procedure then continues as before.

A summary of the procedure is given in the form of an information flow chart in Figure 2. The code employed in testing the procedure is given in Appendix A and is in the PL/I language.

### Example Problem

Consider the problem of finding a feasible lattice point which satisfies the system of linear diophantine inequalities,

$$x_1 - 2x_2 \geq -2 \quad (3-7)$$

$$-2x_1 + x_2 \geq -4 \quad (3-8)$$

$$2x_1 + 3x_2 \geq 12 \quad (3-9)$$

$$x_j \geq 0 \text{ (j = 1, 2)}. \quad (3-10)$$

Figure 3 represents this system graphically and indicates the progression of the above procedure in obtaining the point,

$$\underline{x} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

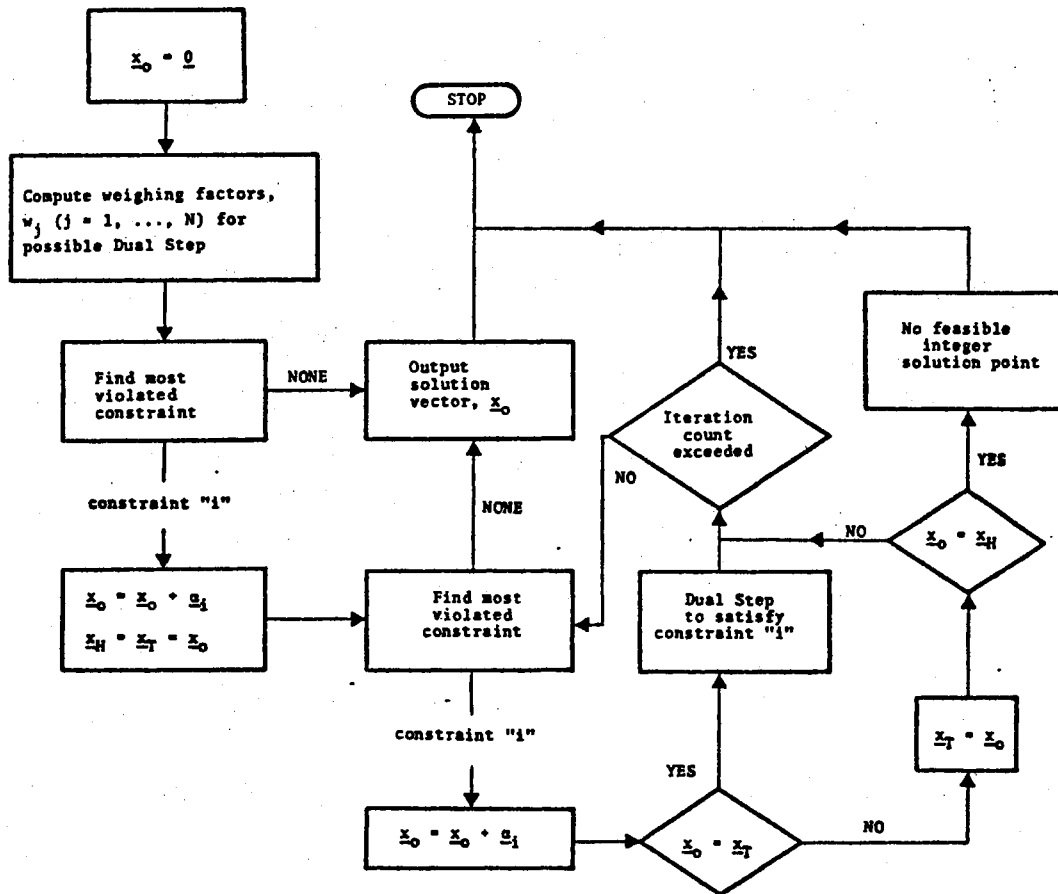


Figure 2. Flow Chart for Procedure for Finding A Feasible Lattice Point

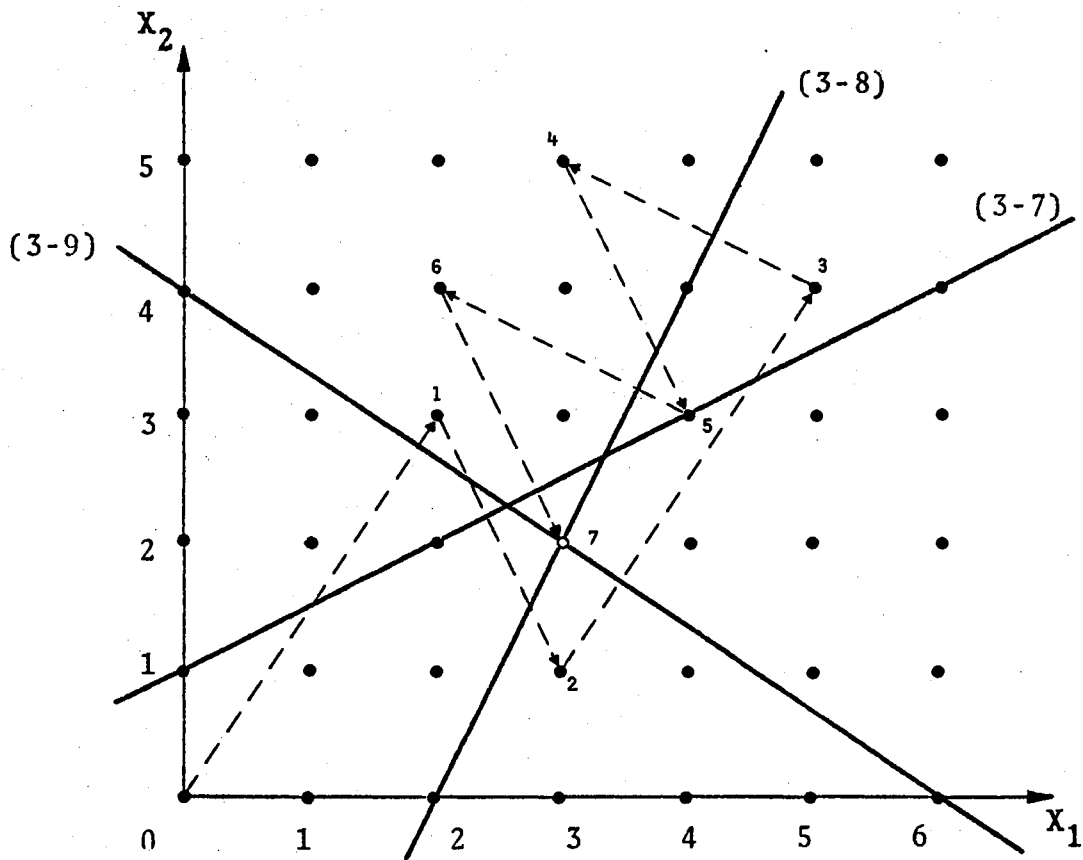


Figure 3. Graphical Description of Example Problem

as a feasible lattice point in seven iterations. It is of interest to note that if the inequality (3-9) is perturbed to be,

$$2x_1 + 3x_2 \geq 13 \quad (3-11)$$

then no feasible lattice point exists. Figure 4 shows the resultant solution space and demonstrates the occurrence of a cycle, detected at the nineteenth iteration, thereby indicating that no feasible lattice point may be found.

#### Comparison with Other Procedures

Garfinkel and Nemhauser (7) present two alternative procedures, developed by F. S. Hillier, for finding a feasible lattice point which satisfies a system of linear inequalities. Both procedures require the solution of a linear programming problem,  $\underline{x}^*$ , over the constraint set as a beginning. From this starting point, the procedures are as follows.

The first procedure is essentially a search of the integral neighborhood of the continuous solution,  $\underline{x}^*$ . This neighborhood is continually increased until a feasible point is detected. The second procedure is, in some respects, similar to that proposed here. With this procedure, a lattice point,  $\underline{x}^1$ , in the integral neighborhood of  $\underline{x}^*$  is used to determine,

$$I(\underline{x}^1) = \sum_{i=1}^M \sum_{j=1}^N \text{MAX}(b_i - \sum_j a_{ij} x_j, 0) \quad (3-12)$$

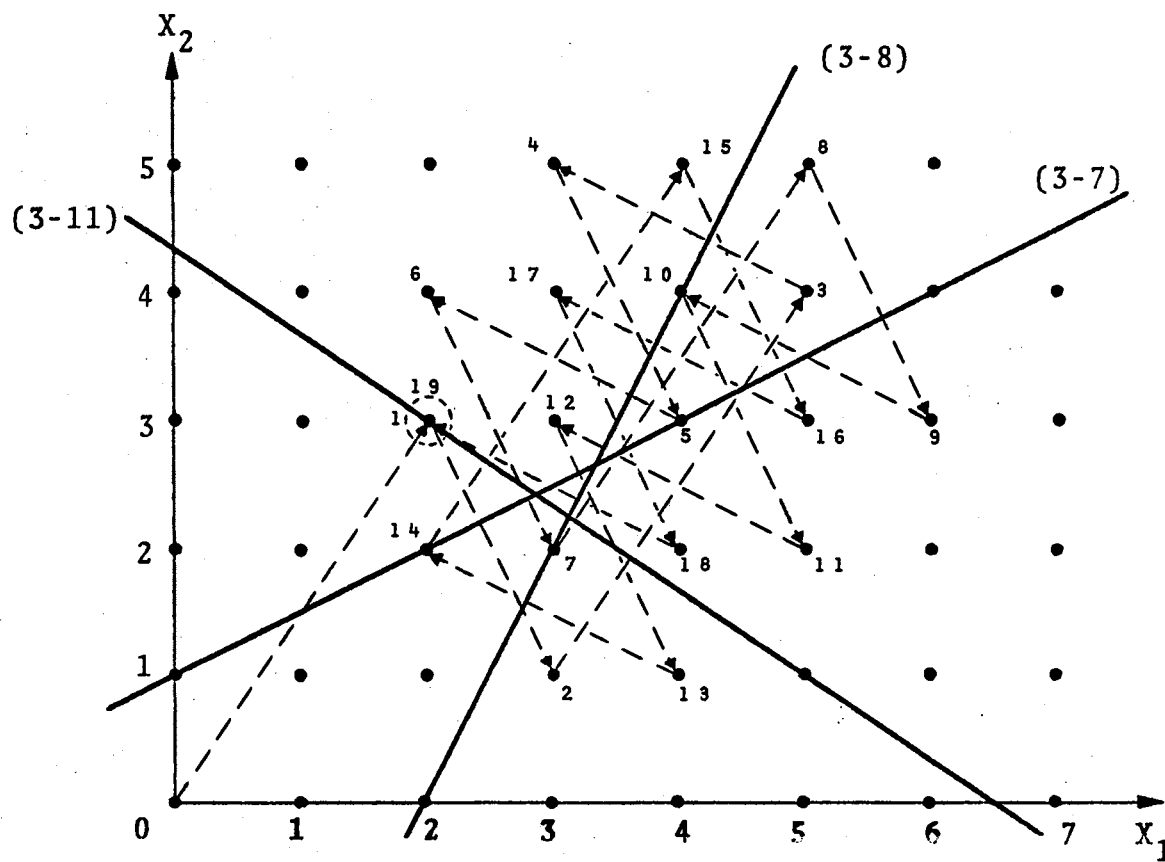


Figure 4. Graphical Description of Modified Example Problem With A Cycle



(i.e., a measure of the infeasibility of the point  $\underline{x}^1$ ). The procedure then seeks to drive  $I(\underline{x}^1) \leq 0$  by moving through successive neighboring lattice points,  $\underline{x}^{1'}$ , of the current point  $\underline{x}^1$ , with a new  $\underline{x}^1$  being established at the point  $\underline{x}^{1'}$  whose value  $I(\underline{x}^{1'})$  is a minimum.

The major advantage of the procedure proposed here over the two stated alternatives is that it does not require the solution of a linear programming problem enroute to obtaining a feasible lattice point. The disadvantage of this procedure relative to the two alternatives is that, if the problem is an integer linear programming problem, the alternative procedures yield a feasible lattice point which has a greater likelihood of being near the optimum integral solution point.

## CHAPTER IV

### A TIME-SHARED ILP ALGORITHM FOR STRICTLY LIMITED RESOURCE ALLOCATION PROBLEMS

This chapter presents an algorithm specifically developed to solve a particular class of optimization problems, referred to here as "strictly limited" resource allocation problems. Such problems take the form,

$$\text{Maximize: } F = \underline{c}^T \underline{x} \quad (4-1)$$

subject to the restrictions,

$$A \underline{x} \leq \underline{b} \quad (4-2)$$

$$x_j \geq 0 \text{ and integer} \quad (4-3)$$

where for all  $i, j$ :

$$a_{ij} \geq 0 \quad (4-4)$$

$$b_i > 0 \quad (4-5)$$

$$c_j > 0 \quad (4-6)$$

Problems of this form are most commonly encountered in a capital budgeting or knapsack context (in which  $A$  becomes  $\underline{a}^T$ ), but also occur frequently where more than one resource is under such limitation.

Although several general integer linear programming (ILP) algorithms are available which adequately solve such problems, they represent characteristically large

codes of the batch processing variety. The algorithm presented here has been implemented with a relatively small code (175 executable CPS PL/I statements), designed specifically for the conversational character of time-shared computation.

#### Foundation of the Procedure

Of importance in the development of this procedure are certain geometric characteristics of the bounding polytope,  $(Po)_N$ , constituted by (4-2) and (4-3), and the objective function surface. Primary among these is the observation that extremity coordinates of the feasible domain lie on the axial planar face constituents of  $(Po)_N$ , and consequently the integral upper bounds of the problem variables cannot increase as the objective function surface, while attempting to maximize, traverses the polytope from its origin vertex. This being understood, it may further be observed that an implicit, sequential examination of these face constituents can be made to determine if a potential improvement in the objective function exists. That is, the feasible domain complementary to  $F(x^0) \cap (Po)_N$  may be examined for a potential improvement in the objective function as follows.

Define:  $\max_j$  = maximum constraint intercept with  
the  $x_j$  axis.

$U_j$  = integral upper bound on  $x_j$ .

Then, for a potential improvement to exist by reducing the value of  $x_k$  and increasing the value of  $x_i$  at any lattice extremity,

$$\left[ \frac{C_k}{C_i} \right] > \left[ \max_i U_k \right] \quad (4-7)$$

Although this does not ensure that such a move will improve the objective function, it does provide a means for ruling out those moves which will not.

To more specifically outline how these observations may be utilized in obtaining optimality, consider the stepwise procedure shown in the following information flow chart (Figure 5).

#### Description of the Procedure

In brief, the procedure may be described as a sequential comparison, using (4-7) above, to establish a potential for improvement. If such a potential exists, the indicated tradeoff of values is made and a new solution point generated. The new point is then accepted or rejected as an improvement direction based solely on the value of the objective function at that point. The solution process by total contribution potential ( $T_j = U_j * C_j$ ) is intended to provide an initial means for creating as large an intersection  $[F(\underline{x}^0) \cap (Po)_N]$  as possible at each iteration. This, in effect, assumes that the trace of  $F(\underline{x}^0)$  on the axial planar face constituents of  $(Po)_N$  is most likely to reflect a maximal lattice point at an extremity of those

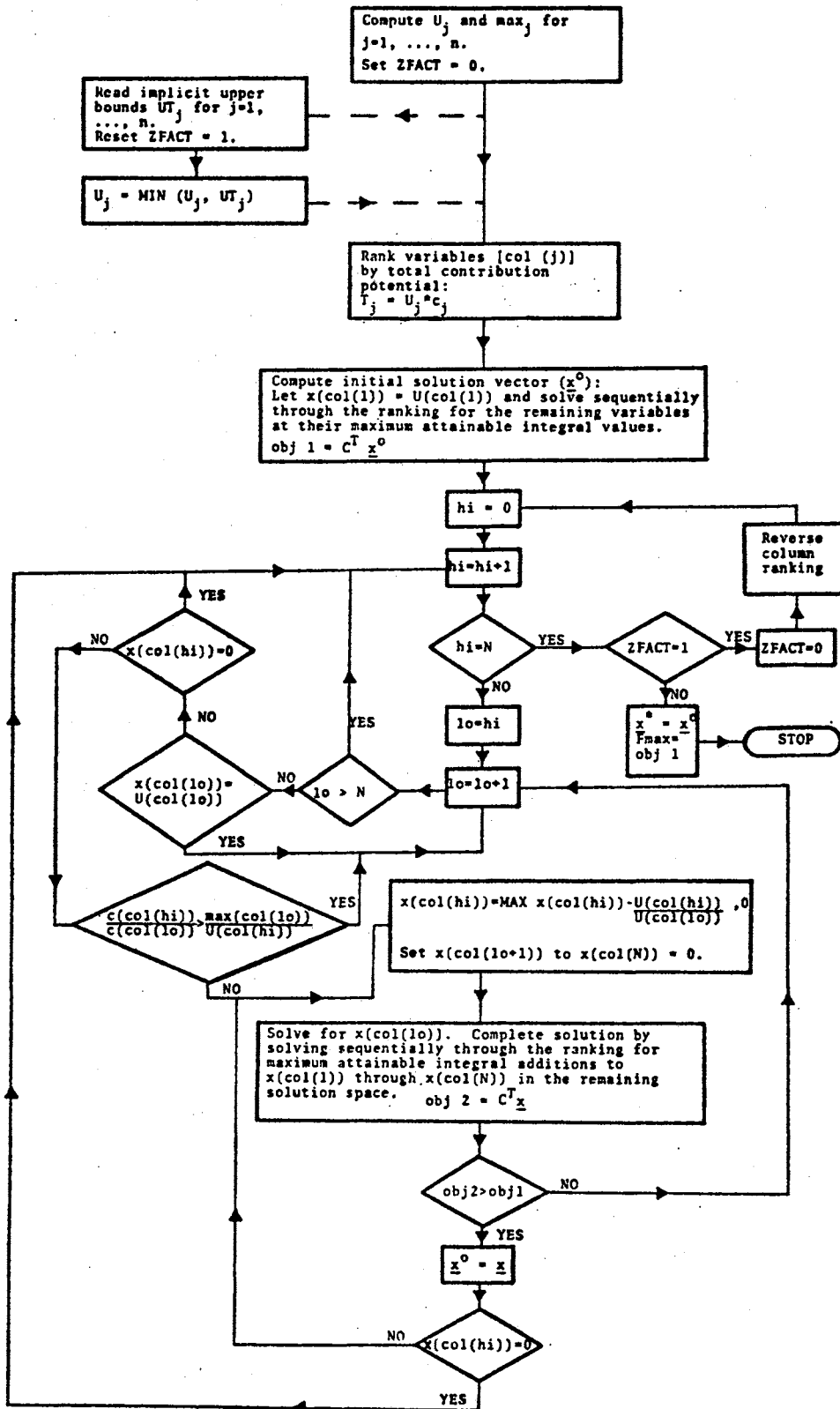


Figure 5. IRESAL Procedure--Information Flow Chart

constituents of greatest contribution potential.

Of particular interest is the necessity for a re-analysis by column-rank reversal (i.e., ZFACT = 1) when implicit upper bounds are specified. In effect, these bounds may prohibit a feasible lattice extremity of the solution space from being reached on a first pass due to the resultant distortion in the geometric comparison of (4-7) and the means employed in generating a complete new solution point. This is especially true of problems which admit only binary solutions. The second pass has the effect of forcing a re-analysis which admits these feasible points by a counter distortion of the comparison and solution process.

#### Computational Experience

This algorithm has been tested with many single and multiple constraint problems. Of special interest is a comparison of the solutions to the nine allocation test problems given by Trauth and Woolsey (33). These problems are described here for clarity. All nine problems have the following form:

$$\text{Maximize: } F = 20x_1 + 18x_2 + 17x_3 + 15x_4 + 15x_5 + 10x_6 \\ + 5x_7 + 3x_8 + x_9 + x_{10}$$

subject to the constraints,

$$30x_1 + 25x_2 + 20x_3 + 18x_4 + 17x_5 + 11x_6 + 5x_7 + 2x_8 \\ + x_9 + x_{10} \leq b_k$$

$$x_i \in (0, 1), i = 1, \dots, 10$$

where the nine values of  $b_k$  are given in Table I.

TABLE I  
CORRESPONDING VALUES OF  $k$  AND  $b_k$

k:	1	2	3	4	5	6	7	8	9
$b_k$ :	55	60	65	70	75	80	85	90	100

These problems were solved by the method proposed here and, for purposes of comparison, by a branch-and-bound mixed-integer programming algorithm (BBMIP). The results are given in Table II.

The above problem associated with  $k = 6$  provides an example of the necessity for a re-analysis by column-rank reversal. It may be observed that on the initial pass only three trial vectors were explicitly examined. They are given with their corresponding objective function values below:

TABLE II  
ALLOCATION TEST PROBLEM RESULTS

	b	x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>	x <sub>4</sub>	x <sub>5</sub>	x <sub>6</sub>	x <sub>7</sub>	x <sub>8</sub>	x <sub>9</sub>	x <sub>10</sub>	F	sec
	55	0	0	0	1	1	1	1	1	1	1	50	34.
I	60	0	0	1	1	0	1	1	1	1	1	52*	64.
R	65	0	0	1	1	1	0	1	1	1	1	57	57.
E	70	0	0	1	1	1	1	0	1	1	1	62	61.
S	75	0	0	1	1	1	1	1	1	1	1	67	36.
A	80	0	1	0	1	1	1	1	1	1	1	68*	39.
L	85	0	1	1	1	0	1	1	1	1	1	70*	73.
	90	0	1	1	1	1	0	1	1	1	1	75	53.
	100	0	1	1	1	1	1	1	1	1	1	85	39.
	55	0	0	0	1	1	1	1	1	1	1	50	.82
	60	0	0	1	0	1	1	1	1	1	1	52*	.70
B	65	0	0	1	1	1	0	1	1	1	1	57	.72
B	70	0	0	1	1	1	1	0	1	1	1	62	.75
M	75	0	0	1	1	1	1	1	1	1	1	67	.72
I	80	0	1	1	0	1	1	1	1	0	0	68*	.67
P	85	0	1	1	0	1	1	1	1	1	1	70*	.75
	90	0	1	1	1	1	0	1	1	1	1	75	.75
	100	0	1	1	1	1	1	1	1	1	1	85	.64

\*It is interesting to note the detection of alternate optima by applying solution techniques with different logic structures. A particularly useful result when viewed in a capital budgeting context.



$$\underline{xt}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \underline{xt}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \underline{xt}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$F_1 = 60, \quad F_2 = 65, \quad F_3 = 67.$$

By applying the procedure, it is noted that moving from  $\underline{xt}_2$  to  $\underline{xt}_3$  permits no intermediate solutions containing  $xt(2)$  at any value but zero since it is implicitly restricted to be no greater than one, and any trade-off involving  $xt(2)$  drives the variable to zero. However, taking  $\underline{x}_0 = \underline{xt}_3$  after the first pass, it may be observed that on the eighth second pass iteration,  $xt(3)$  is driven to zero and  $xt(2) = 1$  yielding the optimum solution as indicated in Table II.

The time-shared algorithm developed here is intended to provide a convenient means for solving strictly limited resource allocation problems. Its justification lies in its low storage requirements (i.e., only the current solution vector and one test vector are employed at each iteration) and small object code

which make it amenable to time-shared computation. The wide disparity in execution times between the procedure developed here and the BBMIP algorithm reflect a difference in high speed as opposed to low speed core. This represents only part of the reason for a difference in execution times. It is apparent that this procedure, although of smaller object code, is not as fast in obtaining results as the BBMIP algorithm principally because of the necessity for a reanalysis by column-rank reversal. The CPS PL/I code employed in implementing this procedure is presented in Appendix B.

## CHAPTER V

### INTEGER SEARCH WITH SIMPLEX DESIGNS

This chapter addresses the problem of minimizing a convex, nonlinear function,

$$F(x_1, x_2, \dots, x_N) \quad (5-1)$$

subject to the linear diophantine restrictions,

$$\sum_{j=1}^N a_{ij} x_j \geq b_i \quad (i = 1, \dots, M) \quad (5-2)$$

and  $x_j \geq 0$  and integer ( $j = 1, \dots, N$ ). (5-3)

A solution technique is proposed which employs a sequential search of lattice points based on the repeated construction of simplex vertices,  $S_{(N+1)}$ . The technique is illustrated through an elementary example, then extended and critically analysed by application to a class of non-convex problems referred to as pseudo-Boolean optimization problems.

#### Integer Nonlinear Programming (INLP)

Until the last five years little attention has been given in the literature to the solution of nonlinear programming problems requiring integer solution vectors. This is due, in part, to the fact that such problems are not as commonly encountered in actual situations as

their linear counterparts; but in the main, the reason for an apparent lack of attention to this problem area is that either the problems themselves permitted an alternative linear formulation or there was insufficient expertise available for their solution, resulting in linearization and approximation.

Recent advances in pseudo-Boolean programming (i.e., seeking the optimum solution vector whose components,  $x_j$ , belong to the set  $\{0, 1\}$ ) by Hammer (16, 17, 18) and his associates have made available the basis for solving certain integer nonlinear programming problems by branch-and-bound techniques. The work of Taha (31) has provided additional capacity for minimizing concave, nonlinear functions over a convex polytope. Gisvold and Moe (10) have proposed a modified penalty function approach to the solution of mixed-integer nonlinear programming problems encountered in structural design.

A characteristic most frequently encountered in examining the problems discussed by the above authors is that the functions they seek to optimize are, from a mathematical standpoint, neither convex nor concave. That is, they are either characterized by an indefinite Hessian matrix or are non-unimodal.

A specific problem classification deserves mention since it precipitated the initial inquiry which led to the development of the technique proposed here. The problem is one of obtaining the optimum economic order

quantities (in integers) of several commodities under conditions of constraint. The problem is fundamental and has been traditionally solved as a problem in continuous variables, then rounded to obtain a solution in integers. In effect, an approximation of the most elementary kind was performed. This class of problems, unlike those previously mentioned, is characteristically representative of a strictly convex form.

The preceding discussion is intended to serve as an introduction to the area of integer nonlinear programming. The economic order quantity and pseudo-Boolean problems mentioned will be employed in the discussion and evaluation of the technique here developed.

#### Foundation of the Procedure

The procedure proposed here is a modification and extension of a technique first proposed by Spendley, Hext and Himsworth (29) and later modified by Nelder and Mead (25). As originally proposed, the technique employs the repeated construction of simplex designs,  $S_{(N+1)}$ , to search for the minimum in continuous variables of an unconstrained objective function. To begin, an initial simplex is constructed in  $E^N$  and the objective function evaluated at each vertex  $\underline{x}_i$ . A new vertex is then established by the following procedure.

Define the centroid of the simplex,  $\underline{c}$ , as

$$\underline{c} = \left( \sum_{i=0}^N \underline{x}_i \right) / N+1, \quad (5-4)$$

and the distance from the vertex associated with the worst objective function evaluation,  $\underline{x}_0$ , to the centroid as,

$$\underline{d} = \underline{c} - \underline{x}_0. \quad (5-5)$$

A reflection step is then taken by projecting the vertex,  $\underline{x}_0$ , through the centroid of the simplex some specified distance to a new vertex,  $\underline{x}^1$ , where

$$\underline{x}^1 = \underline{x}_0 + (1 + k) * \underline{d}. \quad (5-6)$$

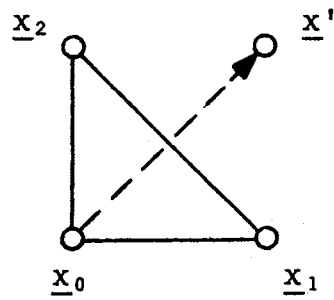
The reflected point,  $\underline{x}_0$ , is then deleted and replaced by the more favorable point,  $\underline{x}^1$ , thereby maintaining a simplex construction. The technique continues in this fashion, varying the value of  $k$  and the size of the simplex until the optimum is attained.

Difficulties encountered with the basic procedure led to refinements providing for acceleration, improved progression in valleys and on ridges, and variations in the basic simplex configuration (4). A major change, instituted by Nelder and Mead, provided for reflections not through the centroid of the entire simplex, but rather through the centroid of the remaining  $N$  points omitting the point to be reflected,  $\underline{x}_0$ . In this instance,

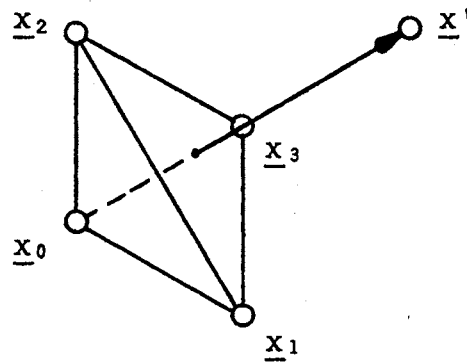
$$\underline{c} = \left( \sum_{i=1}^N \underline{x}_i \right) / N \quad (5-7)$$

with  $\underline{x}^1$  being computed as in equations (5-5) and (5-6).

An example of two and three dimensional reflections using this procedure, with  $k = N-1$ , is given in Figure 6.



A Two Dimensional Reflection



A Three Dimensional Reflection

Figure 6. Example of Simplex Reflections

A more detailed description of these procedures is given in the above cited references and in Himmelblau (20). Of importance here is the basic concept of employing a derivative free search with simplex designs.

### Reflecting to Lattice Vertices

The procedure proposed here employs the basic strategy of the Nelder and Mead simplex search. Certain modifications in the strategy are employed to maintain the integrality of the simplex vertices, and provisions are made for dealing with the linear diophantine constraints should they become active. Before embarking on a description of the procedure, it is necessary to discuss the method employed in maintaining vertex integrality.

Recalling the Nelder-Mead method, consider the following simplex reflection. Given that a simplex is established on lattice points in a space of four dimensions and that  $\underline{x}_0$  represents the vertex of the least desirable objective function value, the reflected point,  $\underline{x}'$ , is computed as follows. Defining the simplex vertices as,

$$\underline{x}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \underline{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \underline{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \underline{x}_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \underline{x}_4 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

the centroid for reflection is



$$\underline{c} = \left( \sum_{i=1}^4 \underline{x}_i \right) / 4 = \begin{bmatrix} 3/4 \\ 3/4 \\ 3/4 \\ 1 \end{bmatrix},$$

and the distance from the reflected point to the centroid becomes

$$\underline{d} = \underline{c} - \underline{x}_0 = \begin{bmatrix} -1/4 \\ -1/4 \\ -1/4 \\ 1 \end{bmatrix}.$$

It is now readily apparent that the first attainable integral vertex along the line of reflection is reached by allowing the parameter  $k$  in equation (5-6) to equal  $N-1$ . The first attainable integral reflection point using centroidial reflections is then generalized to be,

$$\underline{x}^1 = \underline{x}_0 + N * \underline{d}. \quad (5-8)$$

For the above example,

$$\underline{x}^1 = \underline{x}_0 + 4 * \underline{d} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}.$$

Having now established a means for making centroidial reflections to a lattice vertex an obvious difficulty presents itself. Observe that the indicated reflection requires a step which is, in the above example,  $N$  units in one of the component variables (i.e.,  $x_4^1 - x_{04} = 4$ ). In effect, a centroidial reflection can force the search

to cover a minimum neighborhood which omits consideration of lattice points in the immediate (unit) neighborhood of the original simplex.

Recalling the two dimensional reflection illustrated in Figure 6, it may be recognized that a reflection through the centroid of edge vertices on the planar constituents of the simplex provides an alternative reflection procedure which admits consideration of the unit lattice neighborhood of the vertices. As before, an edge reflection may be defined by the following.

Let the centroid of the edge vertices be described by,

$$\underline{c} = (\underline{x}_i + \underline{x}_j)/2 \quad (5-9)$$

where  $(i, j) \in (1, 2, \dots, N)$ ,

$$\text{then } \underline{x}' = 2 * (\underline{c} - \underline{x}_0) + \underline{x}_0 = 2 * \underline{c} - \underline{x}_0 \quad (5-10)$$

For the above example simplex, by allowing  $i = 3, j = 4$  and  $\underline{x}_0$  as before, an edge reflection yields the point,

$$\underline{x}' = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix} .$$

In summary, two basic reflection procedures are employed. The first, centroidal reflections, will be utilized in attaining the general neighborhood of the optimum lattice point by an acceleration mechanism to be described below. Once the general neighborhood is attained, edge reflections are utilized to descend on the optimum lattice point.

### Accelerated Reflections

Considering now a centroidial reflection, let a step length,  $l$ , be defined as the distance between  $\underline{x}_0$  and the first attainable lattice point along the centroidial line of reflection,  $\underline{x}'$ :

$$l = \underline{x}' - \underline{x}_0. \quad (5-11)$$

A method is then desired for generating multiples of this step length to accelerate movement of the simplex toward the optimum. The Fibonacci sequence provides a very compatible basis for such a method by virtue of its integer progression and rapid expansion characteristic.

The sequence is generated by beginning with the numbers  $F(0) = 1$  and  $F(1) = 1$ . Each subsequent number,  $F(i)$ , is equal to the sum of its immediate two predecessors in the sequence. That is,

$$F(i) = F(i-1) + F(i-2). \quad (5-12)$$

The sequence has the added advantage of starting slowly (the numbers  $F(1)$  through  $F(3)$  are only one unit apart) but then increasing rapidly (the number  $F(49) = 12,586,269,025$ ). By employing the Fibonacci sequence as multipliers of the step length, the procedure is accelerated as follows:

Let the centroidial reflection to  $\underline{x}'$  be redefined as,

$$\underline{x}' = F(k) * (N * \underline{d}) + \underline{x}_0 \quad (5-13)$$

where initially  $k = 1$ . Then, if  $\underline{x}'$  has an associated objective function value better than that of  $\underline{x}_0$  and  $\underline{x}'$

is also a feasible lattice point, then  $k$  is incremented by one and the reflection recomputed using (5-13). This method continues in like fashion until either the objective function value at  $\underline{x}'$  is worse than at  $\underline{x}_0$  or until an infeasible lattice point is attained. When this occurs, the simplex is re-established about the last, best, feasible reflection point. The acceleration procedure is then restarted (i.e.,  $k = 1$ ) and terminates when an  $\underline{x}'$  is found to be either infeasible or of worse objective value when  $k = 1$ . At this occurrence, edge reflections are employed to examine a closer neighborhood of the simplex.

#### Handling Constraints

Assume now that edge reflections are being employed in search of the integral optimum. The reflection process proceeds by projecting the worst vertex through the centroid of the two best vertices, with the new vertex replacing the worst. Each vertex is checked for feasibility either upon construction of an initial simplex or when the vertex is otherwise generated. If the vertex is feasible then an associated feasibility parameter for that vertex is set to zero, otherwise the parameter is given a value equivalent to the lattice point feasibility parameter,  $v_i$ , described in Chapter III. If at any step in the reflection process a simplex is obtained which is constituted by vertices whose

feasibility parameters are all non-zero (i.e., a completely infeasible simplex), then the following steps are taken.

First, the last feasible vertex is held and the vertices of the new, infeasible simplex, are ranked according to their feasibility parameters. That is, the most infeasible vertex is ranked last and the remaining vertices are ranked in descending order,  $\underline{x}_{N-1}$  to  $\underline{x}_0$ . Now, instead of reflecting to improve the objective function at a vertex, the criterion for reflection is to reduce the measure of infeasibility at a new vertex. Edge reflections are employed to return a vertex to the feasible domain defined by (5-2) and (5-3). Reflections continue until a new feasible vertex is attained which is not the last feasible vertex being held. An attempt is now made to establish a direction for objective function improvement along the active constraint.

Let  $\underline{x}'$  represent the newly generated feasible vertex and  $\underline{x}_0$  the last feasible vertex being held. Define the exploratory direction,  $\underline{\delta}$ , as follows.

$$\underline{\delta} = \begin{cases} \underline{x}' - \underline{x}_0 ; & \text{if } \underline{x}' \text{ better than or} \\ & \text{equal in objective function} \\ & \text{value to } \underline{x}_0 . \\ \underline{x}_0 - \underline{x}' ; & \text{otherwise.} \end{cases} \quad (5-14)$$

An exploratory point,  $\underline{x}_e$ , is then established at,

$$\underline{x}_e = \underline{x}_0 + \underline{\delta}. \quad (5-15)$$

If the objective function is better at  $\underline{x}_e$  and the point is feasible, then  $\underline{x}_0$  is set at  $\underline{x}_e$  and the above step reapplied until a worse or infeasible  $\underline{x}_e$  is attained. When this occurs the simplex is reconstructed about the new  $\underline{x}_0$  and the reflection procedure begins anew. If, on the other hand,  $\underline{x}'$  is a better and feasible point relative to  $\underline{x}_0$  and  $\underline{x}_e$  is not, then the simplex is constructed about  $\underline{x}'$ . The vertices of the new simplex are then evaluated for objective function value and feasibility. If a better and feasible vertex exists on this simplex, then a new simplex is constructed about that vertex (in effect, the simplex slides in that direction). This process continues until no better vertex in the newly constituted simplex is found. At this point the procedure of edge reflections is restarted.

#### Stopping Criteria

When the attempt to reobtain feasibility above yields points  $\underline{x}'$  and  $\underline{x}_e$  neither of which is an improvement over the last, best feasible vertex,  $\underline{x}_0$ , a unit neighborhood search is employed first around  $\underline{x}_0$  and then the best vertex point of the simplex containing  $\underline{x}_0$ . If neither search yields an improved feasible point, then the procedure terminates with the optimum integral solution vector indicated at  $\underline{x}_0$ . If a better point is found, the simplex is reconstructed about this point and

the procedure of edge reflections is restarted.

Until now it has been assumed that unit simplex reflections in the basic procedure produced points of better objective function evaluation. If such is not the case, then the following occurs. Rather than reflecting through the centroid of the two best vertices, the second best vertex is replaced by the next best and a new edge reflection is made. If this vertex is an improvement, then it replaces the reflected vertex and the procedure continues as before. Otherwise, the best vertex is maintained and the next vertex in the ranking employed with it to make an edge reflection. This process continues until either a vertex better than the reflected vertex is found or until the best vertex has been paired with the remaining vertices yielding no improvement. When the latter occurs, the same unit neighborhood search is employed as with the case of  $\underline{x}'$  and  $\underline{x}_e$  being worse than  $\underline{x}_0$  above. Again, if no better point is found then the procedure terminates with the best feasible vertex of this final simplex indicated as the solution point.

The preceding description illustrates the major aspects of the procedure proposed here. A more unified description is given in the following information flow chart (Figure 7). The PL/I code employed in implementing the procedure is given in Appendix C. To clarify this description, consider the following elementary example

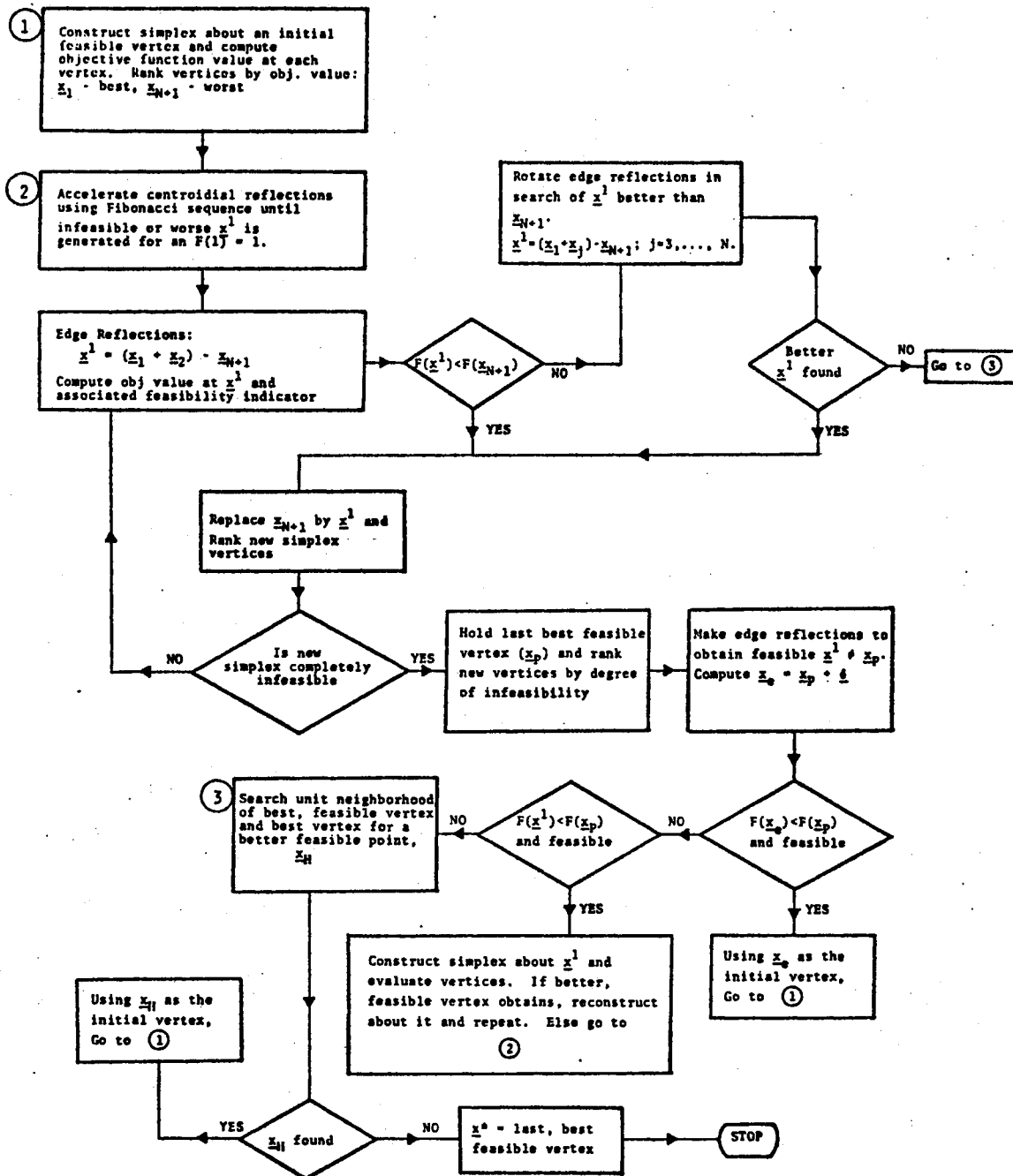


Figure 7. Information Flow Chart for Integer Search With Simplex Designs



developed to illustrate in more specific detail the particulars of the proposed technique.

### Example Problem

Consider the problem of minimizing the function,

$$F = x_1^2 - 8x_1 + 2x_2^2 - 16x_2 \quad (5-16)$$

subject to the linear diophantine restrictions,

$$2x_1 - 10x_2 \geq -20 \quad (5-17)$$

$$-3x_1 + 2x_2 \geq -12 \quad (5-18)$$

and  $x_1, x_2 \geq 0$  and integer. (5-19)

The problem and the progression of this procedure in obtaining the optimum are shown graphically in Figure 8.

The starting simplex is constructed about a feasible lattice point, point 1, with points 2 and 3 being generated by equation (5-20),

$$x_{kj} = x_{ij} + \Delta_j \quad \text{where } \Delta_j = \begin{cases} 1; & j = k \\ 0; & \text{otherwise} \end{cases} \quad (5-20)$$

$$i = 1; k = 2, 3; j = 1, 2.$$

The simplex vertices are then ranked by objective function value ( $F(\underline{x}_1) = 0$ ,  $F(\underline{x}_2) = -7$ ,  $F(\underline{x}_3) = -14$ ) and the worst point ( $\underline{x}_1$ ) is reflected through the centroid of the remaining two yielding  $\underline{x}_4$  as,

$$\underline{x}_4 = \underline{x}_2 + \underline{x}_3 - \underline{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad F(\underline{x}_4) = -21.$$

Since  $\underline{x}_4$  is better than  $\underline{x}_1$  and is feasible, accelerated reflections begin and points 5 and 8 are generated by,

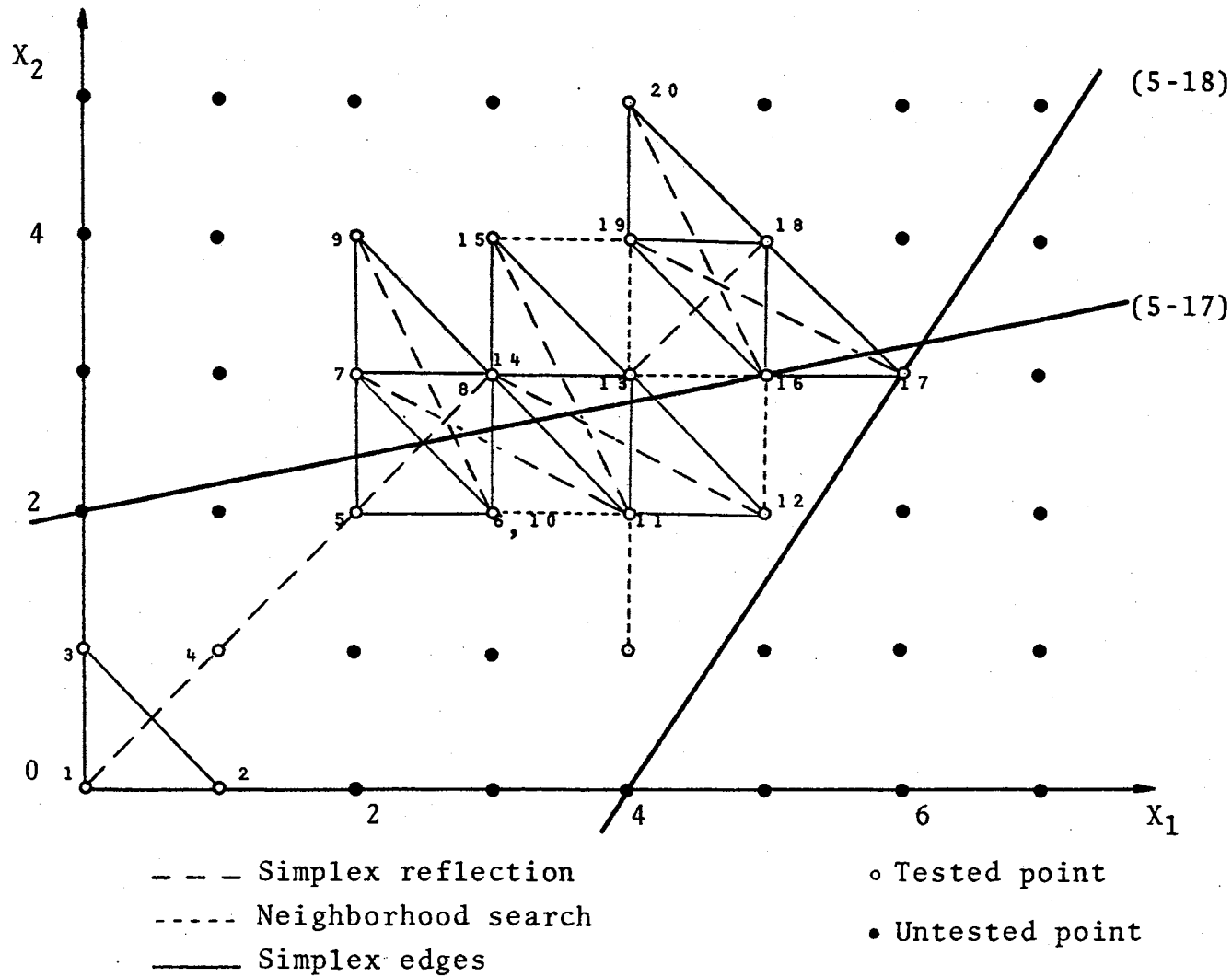


Figure 8. Integer Search Example Problem

$$\underline{x}_5 = 2*(\underline{x}_4 - \underline{x}_1) + \underline{x}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, F(\underline{x}_5) = -36;$$

$$\underline{x}_8 = 3*(\underline{x}_4 - \underline{x}_1) + \underline{x}_1, F(\underline{x}_8) = -45.$$

Since point 8 is infeasible the acceleration is temporarily halted and a new simplex is constructed about the last best point along the line of reflection ( $\underline{x}_5$ ). Points 6 and 7 are generated using equation (5-20) yielding the new simplex vertices,

$$\underline{x}_5 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \underline{x}_6 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \underline{x}_7 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

with objective function values,

$$F(\underline{x}_5) = -36, F(\underline{x}_6) = -39, F(\underline{x}_7) = -42.$$

The acceleration process is restarted by reflecting point 5 through the centroid on points 6 and 7 but stops immediately when point 8, an infeasible point, is reobtained. Now edge reflections (which are the only reflections in a two dimensional space) are begun. Point 8 is held as a better vertex replacing point 5, since there still remains a feasible vertex (point 6);

$$\underline{x}_8 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, F(\underline{x}_8) = -45.$$

Point 6, now the worst vertex, is reflected through the centroid of points 7 and 8 to point 9. Point 6 is replaced by point 9, but since the resulting simplex is completely infeasible point 6 is retained as the last best feasible vertex. The vertices of the infeasible simplex (points 7, 8, 9) are now ranked according to their

degree of infeasibility,

$$v(\underline{x}_9) = 16, v(\underline{x}_7) = 6, v(\underline{x}_8) = 4,$$

and edge reflections are made in an attempt to reobtain a feasible vertex other than the last best feasible vertex (point 6). Point 9 is reflected through the centroid of points 7 and 8 thereby reobtaining point 10. Point 10 replaces point 9 and point 7 is reflected through the centroid of points 10 and 8 yielding point 11. Point 11 replaces point 7 and, since it is a feasible vertex whose objective function value is better than that of point 6 ( $F(\underline{x}_{11}) = -40$ ), a  $\delta$  step is taken as follows,

$$\underline{x}_{12} = (2*\underline{x}_{11} - \underline{x}_6) = \begin{bmatrix} 5 \\ 2 \end{bmatrix}, F(\underline{x}_{12}) = -39.$$

Since point 12 is worse than point 11, but point 11 is better than point 6, a new simplex is constructed about point 11 using equation (5-20),

$$\underline{x}_{11} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}, \underline{x}_{12} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}, \underline{x}_{13} = \begin{bmatrix} 4 \\ 3 \end{bmatrix};$$

$$F(\underline{x}_{11}) = -40, F(\underline{x}_{12}) = -39, F(\underline{x}_{13}) = -46.$$

Edge reflections are begun again, point 12 is reflected to point 14, point 11 is reflected to point 15 and once more the simplex is driven infeasible. Holding point 11 as the last best feasible vertex and ranking the new infeasible simplex vertices by degree of infeasibility,

$$v(\underline{x}_{15}) = 14, v(\underline{x}_{14}) = 4, v(\underline{x}_{13}) = 2.$$

On reobtaining feasibility as before, point 12 is obtained

again. Now a  $\delta$  step is taken to point 10, a worse point than point 11 is obtained. Since no better vertex than the last best feasible vertex (point 11) is found a unit neighborhood search about point 11 and point 13, the best vertex on the simplex containing point 11 (i.e., vertices 11, 13, 14) is made. The neighborhood points which are investigated ( $\underline{x}$ ) are obtained using equation (5-21),

$$\begin{aligned} \underline{x}_j &= \underline{x}_{ij} \pm 1 & j &= 1, \dots, N & (5-21) \\ & & i &= 11, 13. \end{aligned}$$

The search about point 11 yields no better point than point 13; however, on searching the unit neighborhood of point 13, a better point is found (point 16). A new simplex is constructed about this point using equation (5-20),

$$\underline{x}_{16} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}, \quad \underline{x}_{17} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}, \quad \underline{x}_{18} = \begin{bmatrix} 5 \\ 4 \end{bmatrix};$$

$$F(\underline{x}_{16}) = -45, \quad F(\underline{x}_{17}) = -42, \quad F(\underline{x}_{18}) = -47.$$

Once again edge reflections are begun and the simplex is driven infeasible (points 18, 19, 20). This time, after reobtaining feasibility at point 17, making a  $\delta$  step to point 13 and searching the unit neighborhood of point 16 (the last best feasible vertex) and point 19 (the best vertex on the simplex containing point 16), no better feasible vertex is found and the procedure terminates with the optimum integral solution:

$$\underline{x}^* = \begin{bmatrix} 5 \\ 3 \end{bmatrix}, F(\underline{x}^*) = -45.$$

### Computational Experience

As mentioned previously, the development of this proposed technique was precipitated by a desire to obtain the optimum order quantity, in integers, in an inventory control problem subject to constraints. In this regard, consider the following problem from Taha (32).

As stated the problem is to obtain the optimum order quantities of three items whose annual demands are known and constant. The items are assumed replenishable instantaneously with no shortages allowed. In addition, there exists a storage limitation on the space available for inventory. Described mathematically, the problem is to minimize the variable inventory cost,

$$F = \frac{20}{x_1} + 0.15x_1 + \frac{20}{x_2} + 0.05x_2 + \frac{45}{x_3} + 0.10x_3$$

subject to the storage limitation,

$$x_1 + x_2 + x_3 \leq 25$$

with  $x_1, x_2, x_3 \geq 0$  and integer. Recall that the traditional procedure now requires a reformulation in the Lagrangian context as,

$$\begin{aligned} \text{minimize } L = & \frac{20}{x_1} + 0.015x_1 + \frac{20}{x_2} + 0.05x_2 + \frac{45}{x_3} + 0.10x_3 \\ & + \lambda(x_1 + x_2 + x_3 - 25) \end{aligned}$$

where it is assumed (and may be verified) that the constraint is active. The procedure is then to search

for an optimum value of  $\lambda$ , determining an associated  $\underline{x}$  by the usual unconstrained procedure at each test value of  $\lambda$ . This accomplished, the optimum continuous solution is given as:

$$\lambda \doteq 0.3, \underline{x} \doteq \begin{bmatrix} 6.7 \\ 7.6 \\ 10.7 \end{bmatrix} .$$

Assuming that the optimum integer solution lies in the unit neighborhood of the continuous optimum, there are now 26 possible feasible integer combinations to be checked before the optimum is assured.

By applying the procedure suggested here, the optimum integer solution,

$$\underline{x}^* = \begin{bmatrix} 7 \\ 8 \\ 10 \end{bmatrix}, F(\underline{x}^*) = 12.31$$

is directly obtained (program execution time = 1.41 seconds). It should be noted that, unlike the above example, the integer optimum to such problems does not always lie in such a convenient neighborhood of the continuous optimum. In such a case the procedure developed here becomes even more desirable. As a slightly more complex example, consider a similar problem, given in Hadley and Whitin (15), of minimizing

$$F = \frac{50000}{x_1} + 2x_1 + \frac{37500}{x_2} + 10x_2 + \frac{200000}{x_3} + 5x_3$$

subject to the limitation on investment in inventory,

$$20x_1 + 100x_2 + 50x_3 \leq 14000.$$

The optimum integer solution given in the above reference with  $\lambda = 0.091$  is

$$\underline{x} = \begin{bmatrix} 114 \\ 44 \\ 145 \end{bmatrix}, F(\underline{x}) = 4064.$$

Again by the procedure suggested here, the optimum integer solution is found to be:

$$\underline{x}^* = \begin{bmatrix} 112 \\ 46 \\ 143 \end{bmatrix}, F(\underline{x}^*) = 4059.25.$$

Having achieved a measure of success in solving the strictly convex economic order quantity problems, an attempt to extend the procedure to problems of a less desirable mathematical structure was made.

In particular, the procedure was tested on several pseudo-Boolean optimization problems. Hammer and Rudeanu (18) demonstrate that numerous problems in operations research, graph theory and combinatorial mathematics can be brought to the form of optimizing an unconstrained pseudo-Boolean function (in effect, a nonlinear objective function subject to the restriction  $x_j \in \{0,1\}; j = 1, \dots, N$ ). Such being the case, the ability of the proposed technique to solve such problems would add greatly to its generality. Consequently, the technique was applied and the following results obtained.

Several problems similar in form to that given



here were attempted with an equivalent measure of success. Consider the problem, from Saaty (26), of minimizing

$$F = 2x_1 + 3x_2 - 7x_3 - 5x_1 x_2 x_3 + 3x_2 x_4 + 9x_4 x_5 - 2x_1 x_5$$

subject to the restriction  $x_j \in \{0,1\}; j = 1, \dots, N$ .

The optimum solution,

$$\underline{x}^* = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, F(\underline{x}^*) = -9$$

being attained by the program in Appendix C with an execution time of 2.09 seconds, beginning the search with the origin as an initial vertex. The form of this problem becomes important when considering the following problem for which the procedure failed to initially produce the optimum, again starting with the origin as an initial vertex.

The problem, from Hammer and Peled (17), is to minimize,

$$\begin{aligned} F = & -8x_1 x_4 - 7x_2 x_4 x_6 - 5x_3 x_6 - 3x_1 x_2 x_5 \\ & -2x_5 x_6 + 2x_4 x_5 + 3x_2 x_5 x_6 + 4x_1 x_5 \\ & +4x_1 x_5 x_7 x_8 + 6x_3 x_4 x_6 + 6x_4 x_7 x_9 x_{10}, \end{aligned}$$

and subject to the binary restriction

$$x_j \in \{0,1\}; j = 1, \dots, N.$$

Here the non-convexity of the objective function and limited feasible domain exact their toll on the procedure.

It is readily observed that, on constructing the simplex about the origin, no distinguishable vertices exist. That is, unlike the preceding example which contained linear terms that provided improvement vertices, the above problem contains no such terms and consequently provides no such vertices. In fact, centroidial and edge reflections also fail to produce a feasible improvement direction, and the procedure terminates with the origin indicated as the optimum. It is interesting to note, however, that if the simplex is constructed about the point  $\underline{x} = \underline{1}$  with constituent vertices lying in the feasible domain, the optimum is attained at,

$$\underline{x}^* = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, F(\underline{x}^*) = -15.$$

Unfortunately, even with this modification, the execution time required to obtain this solution was far inferior to that given in the above reference for a branch-and-bound technique designed specifically for such problems (i.e., 11.46 seconds as opposed to 0.90

seconds).

Although the above modification did enable the optimum to be attained, similar adjustments fail to produce optima with other such functions where the feasible domain is more rigidly restricted (additional constraints above the  $\{0,1\}$  restrictions are present) or where the objective function is not only non-convex but exhibits finite discontinuities.

In summary, it may be stated that the procedure developed here works well on functions which admit a feasible domain of distinguishable vertices attainable with edge or centroidial reflections from the initial simplex. However, the procedure bogs down immediately without the presence of an initial improvement direction existing in the immediate feasible domain about the initial simplex.

## CHAPTER VI

### CONCLUSIONS AND RECOMMENDATIONS

#### Conclusions

This research demonstrates that certain fundamental concepts in N-dimensional geometry can be employed as a basis for the development of heuristic search techniques applicable to integer programming problems subject to linear diophantine inequality constraints. Specifically, the following are the basic conclusions of this research.

In Chapter III it was stated that the coefficients of a constraint, in greater-than or equal-to form, may be interpreted as components of a vector normal to that constraint and pointing inward to the feasible domain defined by the constraint. This geometric interpretation provides the basis for the development of a procedure for finding a lattice point which satisfies a system of linear diophantine inequality constraints. In essence, the procedure moves toward a feasible point by strategically summing the normal vectors given by the constraints (i.e., moving in a direction indicated by a normal) until a feasible lattice point is found or until the procedure

begins to cycle around the feasible domain. It should be noted that the occurrence of a cycle does not guarantee that no feasible lattice point exists. In this regard, one should recognize that more than one constraint may be equivalently the most violated at a particular intermediate point. The procedure developed here will select only one of these constraint normals (and the same one at every such occurrence) to employ in search of a feasible lattice point. This fact can lead to the occurrence of a cycle if the wrong normal happens to be selected, even though a feasible lattice point exists. As a final remark, it should be observed that the ability of this technique to find a feasible lattice point is much more dependent on the configuration of the bounding polytope than on the number of variables or constraints. For example, if the constraint  $\underline{\alpha} \underline{x} \geq \underline{1}$  is the only constraint which bounds the feasible domain away from the origin and  $\underline{\alpha}^T = \underline{1}$ , then if  $\underline{x}_0 = \underline{1}$  is a feasible lattice point the procedure suggested here will obviously produce the point,  $\underline{x}_0$ , in one iteration regardless of the number of variables or other constraints.

In Chapter IV the integer solution to strictly limited resource allocation problems with linear diophantine objective functions and constraints was examined. It was demonstrated that a geometric interpretation of the constraint inequalities and objective function hypersurface provides a useful basis for the

development of a solution technique for such problems. As a major consequence of this interpretation, the resultant solution technique is shown to be of small object code requiring limited storage and therefore easily implemented in a time-shared computational environment.

In Chapter V it was shown that the basic technique of a search with simplex patterns can be modified to obtain the optimum integer solution to nonlinear objective functions subject to linear diophantine inequality constraints. However, owing to the fact that such a technique is a rudimentary steep descent procedure, it fails to obtain optima without the presence of a feasible improvement direction in the unit neighborhood of the minimum simplex configuration (i.e., the unit simplex). It may further be concluded that an inherent partitioning characteristic of this procedure can be utilized to obtain the optimum solution to certain pseudo-Boolean problems (e.g., the problem from Hammer and Peled). By constructing the initial simplex about the point  $\underline{x}_0 = \underline{1}$  and employing the process of edge reflections and neighborhood searches described previously, the procedure partitions out the constituent variables (by setting them equal to zero) with undesirable effect on the objective function.

### Recommendations

It is recommended that the approach taken in this research, of interpreting mathematical programming problems in a geometric context, be investigated as a means for developing other solution techniques which are amenable to time-shared computation. It is believed that this computational environment is advantageous both for its ease of accessibility to many users and inherent desirability as an educational medium.

A recommendation for modification of the simplex search proposed here can be made in the context of a specific problem. This modification incorporates the idea of edge reflections along with that of partitioning the feasible space. The specific problem is one of allocating police patrol cars to areas within a precinct. Blank (3) discusses the development of this problem and employs an algebraic solution by pseudo-Boolean methods.

Mathematically such problems are, in form, similar to the following problem with finite discontinuities.

$$\text{Minimize } F = \sum_{i=1}^7 r_i + (r_i - \bar{r})^2 * (r_i \neq 0) \quad (6-1)$$

subject to the restrictions,

$$x_1 + x_2 \geq 1 \quad (6-2)$$

$$x_1 + x_3 + x_4 \geq 1 \quad (6-3)$$

$$x_2 + x_3 + x_4 + x_5 \geq 1 \quad (6-4)$$

$$x_4 + x_5 + x_7 \geq 1 \quad (6-5)$$

$$x_3 + x_6 + x_7 \geq 1 \quad (6-6)$$

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \leq 3 \quad (6-7)$$

$$x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 + x_6 + x_7 \leq 5 \quad (6-8)$$

$$\text{and } x_j \in \{0,1\}; j = 1, \dots, 7. \quad (6-9)$$

Here the variables,  $x_j$ , represent the assignment of a patrol car to an area and the variables,  $r_i$ , represent distance parameters defined as:

$$r_1 = 1.3x_1 + 0.3x_1 x_2 x_3 - 0.5x_1 x_3 - 0.6x_1 x_2$$

$$r_2 = 1.5x_2 + 0.3x_1 x_2 x_4 - 0.5x_1 x_2 - 0.8x_2 x_4$$

$$r_3 = 2x_3 + 0.4x_3 x_4 x_6 - 0.7x_3 x_6 - 0.9x_3 x_4$$

$$r_4 = 2.2x_4 + 0.6x_3 x_4 x_5 + 0.4x_3 x_4 x_7 + 0.7x_4 x_5 x_7 \\ - 0.9x_3 x_4 - 1.1x_4 x_5 - x_4 x_7 - 0.4x_3 x_4 x_5 x_7$$

$$r_5 = x_5 - 0.3x_5 x_7 - 0.2x_4 x_5$$

$$r_6 = 0.8x_6 - 0.2x_6 x_7$$

$$r_7 = 1.2x_7 + 0.1x_5 x_6 x_7 - 0.3x_6 x_7 - 0.3x_5 x_7$$

$$\text{and } \bar{r} = (r_1 + r_2 + r_3 + r_4 + r_5 + r_6 + r_7)/3.$$

The method proposed for further investigation may best be described in the following stepwise fashion.

Step 1: Initialize a simplex about the point  $\underline{x}_0 = \underline{1}$ , with

$$\underline{x}_k = \underline{x}_0 + \underline{\delta}_k$$

where

$$\delta_{kj} = \begin{cases} 0; & k \neq j \\ -1; & k = j \end{cases}$$

for

$$k = 1, \dots, N$$



- Step 2: Compute the objective function value,  $F(k)$ , and feasibility parameter,  $V(k)$ , as in Chapter V for each vertex. If a vertex satisfies the lower bound restrictions, (6-2) through (6-6), then let the parameter  $lv(k) = 1$  otherwise  $lv(k) = 0$ .
- Step 3: Rank vertices by objective function value.
- Step 4: Select first as the vertex,  $\underline{x}_i$ , the one having the best objective function value and having the parameter  $lv(i) = 1$ . Select as a second vertex,  $\underline{x}_j$ , that vertex with the best objective function value among those whose feasibility parameter,  $V(j)$ , is a minimum and having the parameter  $lv(j) = 1$ ; where  $i \neq j$ .
- Step 5: If  $\underline{x}_i = \underline{x}_j$  and  $V(i) = 0$ , or if  $V(i) = 0$  and no  $lv(j) = 1$  is found then stop with  $\underline{x}_i$  as the solution.
- Step 6: Reflect  $\underline{x}_0$  through the edge constituted by  $\underline{x}_i$  and  $\underline{x}_j$  to  $\underline{x}'$ .
- Step 7: Partition the space by elimination of the constituents of  $\underline{x}'$  having zero elements. That is, reduce the feasible space by eliminating from further consideration the constituent dimensions,  $k$ , having  $x_k' = 0$ .
- Step 8: Utilizing  $\underline{x}_0 = \underline{x}'$  partitioned, return to Step 1.

A time-shared code of this procedure was tested on the above problem yielding the optimum solution as

given by Blank,

$$\underline{x}^T = (0, 1, 0, 1, 0, 0, 1), F(\underline{x}) = 3.2667.$$

It is further suggested that the above procedure be validated and compared with the branch-and-bound algorithm of Hammer and Peled (17) for problems as above.

## SELECTED BIBLIOGRAPHY

- (1) Agnew, Jeanne. Explorations in Number Theory. Monterey, California: Brooks/Cole Publishing Co., 1972.
- (2) Balas, Egon. "An Additive Algorithm for Solving Linear Problems with Zero-One Variables." Operations Research, Vol. 13 (1965), 517-546.
- (3) Blank, L. T. "A Quantitative Investigation of Police Patrol Force Distribution." (Unpub. Ph.D. dissertation, Oklahoma State University, July, 1970.)
- (4) Box, M. J. "A New Method of Constrained Optimization and A Comparison with Other Methods." Computer Journal, Vol. 8 (1965), 42-52.
- (5) Cassels, J. W. S. An Introduction to the Geometry of Numbers. Berlin: Springer-Verlag, 1959.
- (6) Cooper, Leon and David Steinberg. Introduction to Methods of Optimization. Philadelphia: W. B. Saunders Co., 1970.
- (7) Garfinkel, R. S. and G. L. Nemhauser. Integer Programming. New York: John Wiley and Sons, 1972.
- (8) Geoffrion, A. M. "Integer Programming by Implicit Enumeration and Balas' Method." SIAM Review, Vol. 9 (1967), 178-190.
- (9) Geoffrion, A. M. and R. E. Marsten. "Integer Programming Algorithms: A Framework and State-of-the-Art Survey." Management Science, Vol. 18 (1972), 465-491.
- (10) Gisvold, K. M. and J. Moe. "A Method for Nonlinear Mixed-Integer Programming and its Application to Design Problems." Journal of Engineering for Industry (May, 1972), 353-364.

- (11) Glover, Fred. "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem." Operations Research, Vol. 13 (1965), 879-919.
- (12) Gomory, R. E. "Faces of an Integer Polyhedron." Proceedings of the National Academy of Science, Vol. 57 (1967), 16-18.
- (13) Greenberg, Harold. Integer Programming. New York: Academic Press, 1971.
- (14) Grunbaum, Branko. Convex Polytopes. Interscience Publishers. New York: John Wiley and Sons, Ltd., 1967.
- (15) Hadley, G. L. and T. M. Whitin. Analysis of Inventory Systems. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1963.
- (16) Hammer, P. L. "B-B-B Methods. I. Maximization of a Pseudo-Boolean Function." Technion, Mimeograph Series on Operations Research, Statistics and Economics, No. 32 (1969).
- (17) Hammer, P. L. and U. N. Peled. "On the Maximization of a Pseudo-Boolean Function." Journal of the Association for Computing Machinery, Vol. 19 (1972), 265-282.
- (18) Hammer, P. L. and S. Rudeanu. Boolean Methods in Operations Research and Related Areas. Berlin: Springer-Verlag, 1968.
- (19) Hammer, P. L. and E. Shlifer. "Applications of Pseudo-Boolean Methods to Economic Problems." Technion, Mimeograph Series on Operations Research, Statistic and Economics, No. 31 (1969).
- (20) Himmelblau, D. M. Applied Nonlinear Programming. New York: McGraw-Hill Book Co., 1972.
- (21) Lasdon, L. S. Optimization Theory for Large Systems. New York: The Macmillan Company, 1972.
- (22) Lawler, E. L. and M. D. Bell. "A Method for Solving Discrete Optimization Problems." Operations Research, Vol. 14 (1966), 1098-1112.
- (23) Little, J. D. C., K. G. Murty, D. W. Sweeney and C. Karel. "An Algorithm for the Traveling Salesman Problem." Operations Research, Vol. 11 (1963), 979-989.

- (24) Mordell, L. J. Diophantine Equations. New York: Academic Press, 1969.
- (25) Nelder, J. A. and R. Mead. "A Simplex Method for Function Minimization." Computer Journal, Vol. 7 (1964), 308-313.
- (26) Saaty, T. L. Optimization in Integers and Related Extremal Problems. New York: McGraw-Hill Book Co., 1970.
- (27) Shanno, D. F. and R. L. Weil. "Management Science: A View from Nonlinear Programming." Communications of the ACM, Vol. 15 (1972), 542-549.
- (28) Sommerville, D. M. Y. An Introduction to the Geometry of N-dimensions. New York: Dover Publications, Inc., 1958.
- (29) Spendley, W., G. R. Hext and F. R. Himsworth. "Sequential Application of Simplex Designs in Optimization and Evolutionary Operation." Technometrics, Vol. 4 (1962), 441-461.
- (30) Stein, F. M. Introduction to Matrices and Determinants. Belmont, California: Wadsworth Publishing Co., Inc., 1969.
- (31) Taha, H. A. "A Class of Convexity Cuts for 0-1 Linear Programs with Application to the Minimization of Certain Concave Problems." (Report No. 71-3, Department of Industrial Engineering, University of Arkansas, 1971.)
- (32) Taha, H. A. Operations Research An Introduction. New York: The Macmillan Company, 1971.
- (33) Trauth, C. A. and R. E. Woolsey. "Integer Linear Programming: A Study in Computational Efficiency." Management Science, Vol. 15 (1969), 481-493.
- (34) Wagner, H. M. Principles of Operations Research. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1969.
- (35) Wilde, D. J. and C. S. Beightler. Foundations of Optimization. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1967.
- (36) Woodworth, Forrest. Graphical Simulation. Scranton, Pennsylvania: International Textbook Company, 1967.

APPENDIX A

CODE FOR FINDING A FEASIBLE LATTICE POINT

STMT LEVEL NEST  
1

PROGRAM\_2: PROCEDURE OPTIONS (MAIN) ;

```
/* ***** */  
/* FINDING A FEASIBLE LATTICE POINT SATISFYING */  
/* A SYSTEM OF LINEAR DIOPHANTINE INEQUALITIES */  
/* REMARKS: */  
/* A. CONSTRAINTS MUST BE IN ">=" FORM */  
/* B. INPUT: */  
/* 1. NUMBER OF CONSTRAINTS (M) */  
/* NUMBER OF VARIABLES (N) */  
/* NUMBER OF ITERATIONS BEFORE */  
/* TERMINATION (ITER) */  
/* 2. CONSTRAINT COEFFICIENTS (IA(I,J)) */  
/* FOR EACH CONSTRAINT, SEPARATED BY */  
/* COMMAS */  
/* 3. RIGHT HAND SIDE (IB(I)) OF EACH */  
/* CONSTRAINT IMMEDIATELY FOLLOWING */  
/* COEFFICIENTS AND ON SEPARATE CARD */  
/* ***** */
```

```
2 1 DECLARE M FIXED, N FIXED ;  
3 1 DECLARE L FIXED ;  
4 1 DECLARE ITER FIXED ;  
5 1 DECLARE IA(10,20) FIXED, IB(10) FIXED, IBT(10) FIXED ;  
6 1 DECLARE X(20) FLOAT(16) ;  
7 1 DECLARE XT(20) FLOAT(16) ;  
8 1 DECLARE LH FIXED, XH(20) FLJAT(16) ;  
9 1 DECLARE CSUM(20) FLOAT(16) ;  
  
10 1 GET LIST (M,N,ITER) ;  
11 1 PUT PAGE ;  
12 1 PUT EDIT('CONSTRAINING RELATIONSHIPS')(SKIP,X(20),A,SKIP) ;  
13 1 DO I=1 TO M ;  
14 1 1 GET LIST((IA(I,J) DO J=1 TO N)) ;  
15 1 1 GET LIST (IB(I)) ;  
16 1 1 PUT EDIT((IA(I,J) DO J=1 TO N), ' >=',IB(I))(SKIP,(N) F(6,0),A,F(8,0)) ;  
17 1 1 END ;  
  
/* **** COMPUTING WEIGHTING FACTORS FOR DUAL STEP **** */  
  
18 1 DO J=1 TO N ;  
19 1 1 CSUM(J)=0. ;  
20 1 1 DO I=1 TO M ;  
21 1 2 IF IA(I,J) >= 0 THEN GO TO T2 ;  
23 1 2 CSUM(J)=CSUM(J)+IA(I,J) ;  
24 1 2 T2: END ;  
25 1 1 END ;  
  
/* ***** */  
/* ***** INITIAL STEP ***** */  
  
26 1 XH(*)=0. ;  
27 1 IBT(*)=IB(*) ;
```

STMT LEVEL NEST

```

28 1          KSTEP=1 ;
29 1          CALL FIND (M,IBT,L) ;
30 1          IF L=0 THEN GO TO ANSWER ;
32 1          CALL DIRECT (N,L,IA,IB,X) ;
33 1          PUT SKIP(2) ;
34 1          PUT LIST('VARIABLE ASSIGNMENTS AT ITERATION ',KSTEP) ;
35 1          PUT EDIT((X(J) DO J=1 TO N) (SKIP,(N)(F(4,0),X(1))) ;
36 1          XT(*)=X(*) ;

          /* ***** */

37 1          T1: KSTEP=KSTEP+1 ;

          /* ***** ITERATION COUNT CHECK ***** */
38 1          IF KSTEP>ITER THEN GO TO ITERCOUNT ;
          /* ***** */

40 1          DO I=1 TO M ;
41 1      1      IBT(I)=IB(I)-SUM(IA(I,*)X(*)) ;
42 1      1      END ;
43 1          CALL FIND (M,IBT,L) ;
44 1          IF L=0 THEN GO TO ANSWER ;
46 1          IF MOD(KSTEP,2)≠0 THEN XH(*)=X(*) & LH=L ;
48 1          CALL DIRECT (N,L,IA,IB,X) ;
49 1          PUT SKIP(2) ;
50 1          PUT LIST('VARIABLE ASSIGNMENTS AT ITERATION ',KSTEP) ;
51 1          PUT EDIT((X(J) DO J=1 TO N) (SKIP,(N)(F(4,0),X(1))) ;

          /* ** CHECKING FOR A CYCLE - NO FEASIBLE LATTICE PT ** */
52 1          IF X(*)=XT(*) THEN GO TO INFEAS ;
          /* ***** */

          /* ***** CHECKING FOR AN OSCILLATION ***** */
54 1          IF X(*)=XH(*) THEN CALL DJAL(LH,M,N,IA,IB,CSUM,X) ;
56 1          ELSE GO TO T1 ;
          /* ***** */

57 1          XT(*)=X(*) ; GO TO T1 ;

59 1          ITERCOUNT: PUT SKIP(2) ;
60 1          PUT EDIT('ITERATION COUNT EXCEEDED')(SKIP,A) ;
61 1          GO TO OUT ;
62 1          INFEAS: PUT EDIT('*** NO FEASIBLE INTEGER SOLUTION ***')(SKIP(2),A) ;
63 1          PUT EDIT ('NUMBER OF ITERATIONS = ',KSTEP) (SKIP,A,F(4,0)) ;
64 1          GO TO OUT ;
65 1          ANSWER: PUT SKIP(2) ;
66 1          PUT EDIT ('---- SOLUTION VECTOR ----') (SKIP(2),X(20),A) ;
67 1          DO IV=1 TO N ;
68 1      1      PUT EDIT('X(',IV,')=',X(IV)) (SKIP,X(25),A,F(3,0),A,F(8,0)) ;
69 1      1      END ;
70 1          PUT EDIT ('NUMBER OF ITERATIONS = ',KSTEP) (SKIP,A,F(4,0)) ;

71 1          FIND: PROCEDURE (M,IBT,L) ;
72 2          /* ** FINDING MOST VIOLATED CONSTRAINT ** */
          DECLARE M FIXED, IBT(10) FIXED, L FIXED ;

```



STMT LEVEL NEST

```

73  2          ITEST,L=0 ;
74  2          DO I=1 TO M ;
75  2  1          IF IBT(I) <= ITEST THEN GO TO F1 ;
77  2  1          L=I ; ITEST=IBT(I) ;
79  2  1  F1: END ;
80  2          RETURN ;
81  2          END FIND ;

82  1  DIRECT: PROCEDURE (N,L,IA,IB,X) ;
      /* ** MOVING NORMAL TO MOST VIOLATED CONSTRAINT ** */
83  2          DECLARE N FIXED, L FIXED ;
84  2          DECLARE IA(10,20) FIXED, IB(10) FIXED ;
85  2          DECLARE X(20) FLOAT(16) ;
86  2          D1: SCALE=0 ;
87  2          X(*)=X(*)+IA(L,*) ;
88  2          DO J=1 TO N ;
89  2  1          IF X(J)>=SCALE THEN GO TO D2 ;
91  2  1          X(J)=SCALE ;
92  2  1  D2: END ;
93  2          IF IB(L)-SUM(IA(L,*)X(*))>0 THEN GO TO D1 ;
95  2          RETURN ;
96  2          END DIRECT ;

97  1  DUAL: PROCEDURE(LH,M,N,IA,IB,CSUM,X) ;
      /* ** DUAL STEP TO ELIMINATE OSCILLATION ** */
98  2          DECLARE LH FIXED, M FIXED, N FIXED ;
99  2          DECLARE IA(10,20) FIXED, IB(10) FIXED, X(20) FLOAT(16) ;
100 2          DECLARE CSUM(20) FLOAT(16) ;
101 2          K=0 ;
102 2          DO J=1 TO N ;
103 2  1          IF IA(LH,J) <= 0 THEN GO TO DU1 ;
105 2  1          IF K=0 THEN GO TO DU2 ;
107 2  1          IF CSUM(J) <= CSUM(K) THEN GO TO DU1 ;
109 2  1  DU2: K=J ;
110 2  1  DU1: END ;
111 2          X(K)=X(K)+CEIL( (IB(LH)-SUM(IA(LH,*)X(*)))/IA(LK,K) ) ;
112 2          IF X(K) < 0. THEN X(K) = 0. ;
114 2          RETURN ;
115 2          END DUAL ;

116 1          OUT: END PROGRAM_2 ;

```

APPENDIX B

CODE FOR IRESAL PROCEDURE

```

1.      /* ***** */;
2.      /*          INTEGER RESOURCE ALLOCATION ALGORITHM          */;
3.      /*  Maximization context for linear objective function with "<=" constraints.  */;
4.      /*          Program restrictions:          */;
5.      /*          1. All a(i,j) greater than or equal to zero.          */;
6.      /*          2. All b(i), c(j) greater than zero.          */;
7.      /* ***** */;
8.      DECLARE M DEC(3), N DEC(3);
9.      PUT EDIT('Enter the number of constraints (M) and the number of variables (N).')(SKIP,A);
10.     GET LIST(M);
11.     GET LIST(N);
12.     DECLARE A(20,20) DEC(8),B(20) DEC(8),BT(20) DEC(8),x(20) DEC(8),xt(20) DEC(8);
13.     DECLARE c(20) DEC(8),col(20) DEC(3),U(20) DEC(8),T(20) DEC(8);
14.     DECLARE max(20) DEC(9);
15.     DECLARE UT(20) DEC(8);
16.     T,U,c=0;
17.     col=0;
18.     B=0;
19.     A=0;
20.     PUT EDIT('Enter the coefficients of each constraint followed by the right hand side')(SKIP,A);
21.     PUT EDIT('of that constraint, one constraint at a time as requested.')(A);
22.     DO j=1 TO M;
23.     DO i=1 TO N;
24.     GET LIST(A(j,i));
25.     END ;
26.     GET LIST(B(j));
27.     END ;
28.     PUT EDIT('Enter the coefficients of the objective function as requested.')(SKIP,A);
29.     DO i=1 TO N;
30.     GET LIST(c(i));
31.     END ;
32.     /* UPPER BOUND COMPUTATIONS */;
33.     DO j=1 TO N;
34.     max1,max2=0;
35.     DO i=1 TO M;
36.     IF A(i,j)=0 THEN GO TO skip1;
37.     tst1=B(i)/A(i,j);
38.     max1=tst1;
39.     GO TO skip2;
40.     skip1:  tst1=100000;
41.     skip2:  IF i=1 THEN GO TO skip3;
42.     IF tst1>tst2 THEN GO TO last;
43.     skip3:  tst2=tst1;
44.     last:  IF max1<max2 THEN GO TO next;
45.     max2=max1;
46.     next:  END ;
47.     U(j)=TRUNC(tst2);
48.     max(j)=CEIL(max2);
49.     END ;
50.     PUT EDIT('If any implicit upper bounds exist enter a "1" for ZFACT; otherwise enter a "0.')(SKIP,A);

```

```

51. GET LIST(ZFACT);
52. IF ZFACT=0 THEN GO TO NOW;
53. PUT EDIT('Enter the implicit upper bounds as requested. If a variable has no such upper')(A);
54. PUT EDIT('bound enter a value of 10000.')(A);
55. DO zd=1 TO N;
56. GET LIST(UT(zd));
57. END ;
58. DO ze=1 TO N;
59. U(ze)=MIN(U(ze),UT(ze));
60. END ;
61. /* ***** */;
62. NOW: T=U*c;
63. /* COLUMN REARRANGEMENT BY TOTAL CONTRIBUTION */;
64. DO j=1 TO N;
65. DO jj=1 TO N;
66. IF jj=1 THEN GO TO skip4;
67. IF T(jj)<test THEN GO TO last1;
68. IF T(jj)~=test THEN GO TO skip4;
69. IF c(jj)<c(col(j)) THEN GO TO last1;
70. skip4: test=T(jj);
71. col(j)=jj;
72. last1: END ;
73. T(col(j))=0;
74. END ;
75. /* ***** */;
76. BT=B;
77. obj1=0;
78. xt,x=0;
79. xt(col(1))=U(col(1));
80. hi=1;
81. lo=1;
82. DO k=1 TO M;
83. BT(k)=BT(k)-A(k,col(1))*xt(col(1));
84. END ;
85. /* ***** */;
86. CALL SOLSET;
87. /* ***** */;
88. x=xt;
89. BT=B;
90. obj1=obj2;
91. try0: hi=0;
92. try1: hi=hi+1;
93. IF hi=N THEN GO TO REV;
94. KFACT=0;
95. IF x(col(hi))=0 THEN GO TO try1;
96. lo=hi;
97. try2: lo=lo+1;
98. IF lo>N THEN GO TO try1;
99. IF x(col(lo))=U(col(lo)) THEN GO TO try2;
100. rto=U(col(hi))/U(col(lo));

```

```

101.          IF c(col(hi))/c(col(lo))>CEIL(max(col(lo))/U(col(hi))) THEN GO TO try2;
102.  try3:    IF KFACT=1 THEN GO TO try1;
103.          xt(col(hi))=x(col(hi))-CEIL(rto);
104.          IF xt(col(hi))<=0 THEN GO TO null;
105.  adj1:    DO i=1 TO M;
106.          DO j=1 TO lo-1;
107.          BT(i)=BT(i)-A(i,col(j))*xt(col(j));
108.          END adj1;
109.          GO TO jump1;
110.  null:    xt(col(hi))=0;
111.          KFACT=1;
112.          GO TO adj1;
113.  jump1:   DO i=1 TO M;
114.          IF A(i,col(lo))=0 THEN GO TO adj2;
115.          tst3=BT(i)/A(i,col(lo));
116.          GO TO jump2;
117.  adj2:    tst3=100000;
118.  jump2:   IF i=1 THEN GO TO jump3;
119.          IF tst3>=tst4 THEN GO TO adj3;
120.  jump3:   tst4=tst3;
121.  adj3:    END jump1;
122.          xt(col(lo))=MIN(TRUNC(tst4),U(col(lo)));
123.          IF xt(col(lo))=0 THEN GO TO jump4;
124.          DO ii=1 TO M;
125.          BT(ii)=BT(ii)-A(ii,col(lo))*xt(col(lo));
126.          END ;
127.          /* ***** */;
128.  jump4:   CALL SOLSET;
129.          /* ***** */;
130.          IF obj2>=obj1 THEN GO TO jump5;
131.          xt=x;
132.          BT=B;
133.          KFACT=0;
134.          GO TO try2;
135.  jump5:   x=xt;
136.          BT=B;
137.          obj1=obj2;
138.          IF x(col(hi))=0 THEN GO TO try1;
139.          GO TO try3;
140.          ;
141.          /* COLUMN REVERSAL SECTION */;
142.  REV:    IF ZFACT=0 THEN GO TO OPT;
143.          PUT LIST('COLUMN REVERSAL HAS OCCURRED');
144.          PUT LIST(' ');
145.          ZFACT=0;
146.          jhs=TRUNC(N/2);
147.          DO jh=1 TO jhs;
148.          hold=col(jh);
149.          col(jh)=col(N+1-jh);
150.          col(N+1-jh)=hold;

```

```

151.         END ;
152.         GO TO try0;
153.         /* ***** */;
154. SOLSET: PROCEDURE ;
155. major: DO jk=1 TO N;
156.         ;
157. minor: DO ik=1 TO M;
158.         IF A(ik,col(jk))=0 THEN GO TO stp3;
159.         tst5=BT(ik)/A(ik,col(jk));
160.         GO TO stp2;
161. stp3:   tst5=100000;
162. stp2:   IF ik=1 THEN GO TO stp4;
163.         IF tst5>tst6 THEN GO TO stp5;
164. stp4:   tst6=tst5;
165. stp5:   END minor;
166.         IF jk<=10 THEN GO TO stp6;
167.         xt(col(jk))=MIN(TRUNC(tst6),U(col(jk)));
168.         GO TO stp7;
169. stp6:   IF TRUNC(tst6)=0 THEN GO TO stp1;
170.         d=U(col(jk))-xt(col(jk));
171.         inc=MIN(TRUNC(tst6),d);
172.         xt(col(jk))=xt(col(jk))+inc;
173.         DO k1=1 TO M;
174.         BT(k1)=BT(k1)-A(k1,col(jk))*inc;
175.         END ;
176.         GO TO stp1;
177. stp7:   DO kk=1 TO M;
178.         BT(kk)=BT(kk)-A(kk,col(jk))*xt(col(jk));
179.         ;
180.         END ;
181. stp1:   END major;
182.         obj2=0;
183.         DO lv=1 TO N;
184.         obj2=obj2+xt(col(lv))*c(col(lv));
185.         END ;
186.         RETURN ;
187.         END SOLSET;
188.         ;
189.         /* ***** OUTPUT ***** */;
190. OPT:   PUT EDIT('SOLUTION VECTOR      UPPER BOUNDS      COLUMN REARRANGEMENT      MAXIMUM INTERSECTION')(SKIP,X(5),A);
191.         DO j=1 TO N;
192.         PUT EDIT('X(' ,j, ') = ' ,x(j),U(j),col(j),max(j))(A,F(2,0),A,X(3),F(9,0),X(7),F(9,0),X(13),F(3,0),X(16),F(9,0));
193.         END ;
194.         PUT EDIT('OBJECTIVE FUNCTION VALUE = ' ,obj1)(SKIP,A,F(12,2));
195.         STOP ;
196.         END ;

```

APPENDIX C  
CODE FOR INTEGER SEARCH WITH  
SIMPLEX DESIGNS

STMT LEVEL NEST

1

BOUND: PROCEDURE OPTIONS (MAIN) :

```
/* ***** */
/* INTEGER ROUTINE FOR NONLINEAR OBJECTIVE FUNCTIONS */
/* SUBJECT TO LINEAR DIOPHANTINE INEQUALITY RESTRAINTS */
/* ( MINIMIZATION CONTEXT ) */
/* ***** */
/* REMARKS: */
/* 1. CONSTRAINTS MUST BE LINEAR AND IN ">=" FORM */
/* 2. OBJECTIVE FUNCTION: */
/* A. MUST BE CONVEX */
/* B. ENTERED AS INDICATED IN PROCEDURE "OBJ" */
/* 3. INPUT DATA: */
/* A. NUMBER OF CONSTRAINTS(M), NUMBER OF */
/* PROBLEM VARIABLES(N) - SEPARATED BY */
/* COMMAS */
/* B. CONSTRAINT COEFFICIENTS: */
/* 1. INTEGER VALUES */
/* 2. SEPARATED BY COMMAS */
/* C. RIGHT HAND SIDE VALUE FOLLOWS EACH SET */
/* OF CONSTRAINT COEFFICIENTS ON A */
/* SEPARATE CARD (AGAIN AN INTEGER) */
/* D. INITIAL FEASIBLE SOLUTION VECTOR: */
/* 1. MUST BE INTEGER */
/* 2. X(I), I=1 TO N, SEPARATED BY COMMAS */
/* E. PROBLEM NAME(NAME) UP TO 50 CHARACTERS */
/* INCLUDED IN 'TIC' MARKS */
/* 4. PROGRAM VARIABLES: */
/* A. X(*,*) - SIMPLEX VERTICES */
/* B. F(*) - OBJ VALUES AT VERTICES */
/* C. MARK(*) - VERTEX FEASIBILITY PARAMETERS */
/* D. XT(*) - TEST POINT */
/* E. FT - TEST POINT OBJ VALUE */
/* F. MK - TEST POINT FEASIBILITY PARAMETER */
/* G. XH(*),XP(*),XB(*) - HOLDING VECTORS */
/* H. FH,TEST,MH,MP - HOLDING PARAMETERS */
/* ***** */

2 1 DECLARE X(11,10) FIXED , XT(10) FIXED , XH(10) FIXED ;
3 1 DECLARE M FIXED , N FIXED , NN FIXED ;
4 1 DECLARE A(14,10) FIXED , B(14) FIXED ;
5 1 DECLARE MARK(11) FIXED , MK FIXED , MH FIXED ;
6 1 DECLARE F(11) FLOAT (16) , FT FLOAT(16) , FH FLOAT(16) ;
7 1 DECLARE ISUM(10) FIXED ;
8 1 DECLARE XP(10) FIXED , MP FIXED , FP FLOAT(16) ;
9 1 DECLARE NAME CHAR(50) VARYING ;
10 1 DECLARE XB(10) FIXED ;
11 1 DECLARE TEST FLOAT(16) ;

12 1 GET LIST(M,N) ;
13 1 NN=N+1 ;
14 1 DO I=1 TO M ;
15 1 1 GET LIST((A(I,J) DO J=1 TO N)) ;
16 1 1 GET LIST(B(I)) ;
17 1 1 END ;
18 1 GET LIST((X(I,J) DO J=1 TO N)) ;
```



STMT LEVEL NEST

```

19 1 GET LIST(NAME) ;
20 1 MARK(*)=0 ;
21 1 BREAK=0. ;
22 1 CHANGE=0. ;

/* ** INITIAL SIMPLEX CONSTRUCTION ** */

23 1 MAG: DO J=2 TO NN ;
24 1 1 X(J,*)=X(1,*) ;
25 1 1 X(J,J-1)=X(J,J-1)+1 ;
26 1 1 END MAG ;
27 1 DO J=1 TO NN ;
28 1 1 XT(*)=X(J,*) ;
29 1 1 CALL OBJ(XT,FT) ;
30 1 1 F(J)=FT ;
31 1 1 END ;
32 1 CALL RANK(NN,MARK,F,X) ; XH(*)=X(1,*) ;

/* ***** ACCELERATED REFLECTIONS USING FIBONACCI SEQUENCE ***** */

34 1 FO=1 ; F1=1 ;
36 1 ISUM(*)=0 ;
37 1 DO I=1 TO NN-1 ;
38 1 1 ISUM(*)=ISUM(*)+X(I,*) ;
39 1 1 END ;
40 1 XPI(*)=(2.*ISUM(*)-2.*N*X(NN,*)/2. ;
41 1 LONG: XT(*)=X(NN,*)+F1*XP(*) ;
42 1 CALL OBJ(XT,FT) ;
43 1 IF FT < F(NN) & FEAS(M,A,B,XT)=0 THEN GO TO EXPAND ;
45 1 IF F1=1 THEN GO TO START ; ELSE DO ;
48 1 1 X(1,*)=XH(*) ;
49 1 1 GO TO MAG ;
50 1 1 END ;
51 1 EXPAND: F2=F1 ; F1=F1+FO ; FO=F2 ;
54 1 XH(*)=XT(*) ;
55 1 GO TO LONG ;

/* ***** SEARCHING WITH UNIT SIMPLEX REFLECTIONS ***** */

56 1 START: ISUM(*)=0 ;
57 1 XP(*)=0 ; FP=1000000. ;
59 1 DO KL=1 TO NN ;
60 1 1 XH(*)=X(KL,*) ;
61 1 1 MARK(KL)=FEAS1(M,A,B,XH) ;
62 1 1 IF F(KL) < FP & MARK(KL)=0 THEN DO ;
64 1 2 XP(*)=XH(*) ; FP=F(KL) ; MP=MARK(KL) ;
67 1 2 END ;
68 1 1 END ;
69 1 DO I=1 TO 2 ;
70 1 1 ISUM(*)=ISUM(*)+X(I,*) ;
71 1 1 END ;
72 1 XT(*)=ISUM(*)-X(NN,*) ;
73 1 CALL OBJ(XT,FT) ;

```

STMT LEVEL NEST

```

74 1      IF FT>=F(NN) THEN GO TO CHECK ;
76 1
77 1      RST: MK=FEAS1(M,A,B,XT) ;
79 1 1    IF FT < F(1) & MK=0 THEN DO ;
83 1 1    XP(*)=XT(*)-X(NN,*) ; XH(*)=XT(*) ; F1=2 ; GO TO LONG ;
84 1      END ;
85 1 1    DO J=1 TO N ;
87 1 1    IF MARK(J) = 0 THEN GO TO POINT ;
88 1      END ;
90 1      IF MK=0 THEN GO TO POINT ;
      GO TO BOUND ;

91 1      CHECK: KV=NN ; K=1 ;
93 1      CK1: IF NN-K=0 THEN DO ;
95 1 1    XT(*)=XP(*) ; FT=FP ; MK=MP ;
98 1 1    GO TO RUN ;
99 1 1    END ;
100 1     IF NN-K<3 THEN ISUM(*)=ISUM(*)-X(NN-K,*)+X(NN-K+1,*) ;
102 1     XT(*)=ISUM(*)-X(NN-K,*) ;
103 1     CALL OBJ(XT,FT) ;
104 1     IF FT>=F(NN-K) THEN DO ;
106 1 1   K=K+1 ;
107 1 1   GO TO CK1 ;
108 1 1   END ;
109 1     GO TO RST ;

110 1     POINT: DO I=1 TO NN ;
111 1 1   IF FT > F(I) THEN GO TO NEXT ;
113 1 1   DO K=I TO NN ;
114 1 2   XH(*)=X(K,*) ; FH=F(K) ;
116 1 2   MH=MARK(K) ;
117 1 2   X(K,*)=XT(*) ; F(K)=FT ;
119 1 2   MARK(K)=MK ;
120 1 2   XT(*)=XH(*) ; FT=FH ;
122 1 2   MK=MH ;
123 1 2   END ;
124 1 1   NEXT: END POINT ;
125 1     GO TO START ;

/* ***** */
/* ***** SLIDING THE SIMPLEX IN AN OPTIMAL DIRECTION ***** */

126 1     BOUND: IF CHANGE=2. & X(NN,*)=XP(*) THEN DO ;
128 1 1   X(NN,*)=XT(*) ; MARK(NN)=MK ; F(NN)=FT ;
131 1 1   GO TO BRUSH ;
132 1 1   END ;
133 1     X(1,*)=X(NN,*) ; F(1)=F(NN) ; XT(*)=X(1,*) ;
136 1     MARK(1)=FEAS1(M,A,B,XT) ;
137 1     SLIDE: CHANGE=0. ;
138 1     DO J=2 TO NN ;
139 1 1   X1J,*)=X(1,*) ;
140 1 1   X(J,J-1)=X(J,J-1)+1 ;
141 1     END ;
142 1     DO J=2 TO NN ;
143 1 1   XT(*)=X(J,*) ;

```

STMT LEVEL NEST

```

144      1      1      CALL OBJ(XT,FT) ; F(J)=FT ;
146      1      1      MARK(J)=FEAS1(M,A,B,XT) ;
147      1      1      IF F(J) <= F(I) & MARK(J)=0 THEN DO ;
149      1      2          X(I,*)=X(J,*) ; F(I)=F(J) ; MARK(I)=MARK(J) ;
152      1      2          CHANGE=1. ;
153      1      2      END ;
154      1      1      END ;
155      1      1      IF CHANGE=1. THEN GO TO SLIDE ;
157      1      1      XP(*)=X(I,*) ; FP=F(I) ; MP=MARK(I) ;
160      1      1      CHANGE=2. ;
161      1      1      DO J=2 TO NN ;
162      1      1      IF MARK(J)=0 THEN DO ;
164      1      2          CALL RANK(NN,MARK,F,X) ;
165      1      2          GO TO START ;
166      1      2      END ;
167      1      1      END ;
168      1      1      STEP: CALL RANK(NN,MARK,F,X) ;
169      1      1      ISUM(*)=0 ;
170      1      1      DO J=1 TO 2 ;
171      1      1      ISUM(*)=ISUM(*)+X(J,*) ;
172      1      1      END ;
173      1      1      XT(*)=ISUM(*)-X(NN,*) ;
174      1      1      CALL OBJ(XT,FT) ;
175      1      1      MK=FEAS1(M,A,B,XT) ;
176      1      1      IF FT > F(NN) THEN GO TO RUN ;
178      1      1      IF MK<=0 & X(NN,*)=XP(*) THEN GO TO FINAL ;
180      1      1      DO J=1 TO NN ;
181      1      1      IF FT > F(J) THEN GO TO LAB ;
183      1      1      DO K=J TO NN ;
184      1      2          XH(*)=X(K,*) ; FH=F(K) ; MH=MARK(K) ;
187      1      2          X(K,*)=XT(*) ; F(K)=FT ; MARK(K)=MK ;
190      1      2          XT(*)=XH(*) ; FT=FH ; MK=MH ;
193      1      2      END ;
194      1      1      LAB: END ;
195      1      1      GO TO STEP ;

/* ***** */

/* ***** REFLECTING TO REOBTAIN FEASIBILITY ***** */

196      1      1      FINAL: DO I=1 TO NN ;
197      1      1      IF MARK(I)<=0 THEN GO TO MORE ;
199      1      1      IF F(I) > FP THEN GO TO MORE ;
201      1      1      XP(*)=X(I,*) ; FP=F(I) ; MP=MARK(I) ;
204      1      1      X(I,*)=XT(*) ; F(I)=FT ; MARK(I)=MK ;
207      1      1      GO TO BRUSH ;
208      1      1      MORE: END FINAL ;
209      1      1      BRUSH: CALL FRANK(NN,MARK,F,X) ;
210      1      1      BI: ISUM(*)=0 ;
211      1      1      DO I=1 TO 2 ;
212      1      1      ISUM(*)=ISUM(*)+X(I,*) ;
213      1      1      END ;
214      1      1      XT(*)=ISUM(*)-X(NN,*) ;
215      1      1      CALL OBJ(XT,FT) ;
216      1      1      MK=FEAS1(M,A,B,XT) ;

```

STMT LEVEL NEST

```

217 1      IF MK>=MARK(NN) THEN GO TO TURN ;
219 1      IF MK=0 THEN GO TO PLACE ;
221 1      B2: IF FT < FP THEN DO ;
223 1 1      X(1,*)=XT(1) ; F(1)=FT ; MARK(1)=FEAS1(M,A,B,XT) ;
226 1 1      GO TO SLIDE ;
227 1 1      END ;
228 1      IF FT>=FP THEN GO TO LOOK ;
230 1      B3: CALL RANK(NN,MARK,F,X) ; GO TO SOLVE ;
232 1      LOOK: X(NN,*)=XT(*) ;
233 1      F(NN)=FT ; MARK(NN)=MK ;
235 1      XH(*)=XT(*) ; FH=FT ; MH=MK ;
238 1      CALL FRANK(NN,MARK,F,X) ;
239 1      ISUM(*)=0 ;
240 1      DO LK=1 TO 2 ;
241 1 1      [SJM(*)=(SUM(*)+X(LK,*) ;
242 1 1      END ;
243 1      DO LKK=NN TO 2 BY -1 ;
244 1 1      XT(*)=[SUM(*)-X(LKK,*) ;
245 1 1      MK=FEAS1(M,A,B,XT) ; CALL OBJ(XT,FT) ;
247 1 1      IF MK=0 & FT<FP THEN DO ;
249 1 2          BREAK=BREAK+1. ;
250 1 2          IF BREAK>=2. THEN GO TO RUN ;
252 1 2          X(1,*)=XT(1) ; F(1)=FT ; MARK(1)=MK ;
255 1 2          GO TO SLIDE ;
256 1 2          END ;
257 1 1      IF LKK=3 THEN [SUM(*)=[SUM(*)+X(LKK,*)-X(LKK-1,*) ;
259 1 1      END ;
260 1      XT(*)=XH(*) ; FT=FH ; MK=MH ;

/* ***** SLIDING THE SIMPLEX ALONG A CONSTRAINT ***** */
263 1      RUN: XH(*)=XT(*) ; FH=FT ; MH=MK ;
266 1      XB(*)=XH(*)-XP(*) ;
267 1      IF FH > FP THEN XT(*)=XP(*)-XB(*) ; ELSE XT(*)=XH(*)+XB(*) ;
270 1      CALL OBJ(XT,FT) ; MK=FEAS1(M,A,B,XT) ;
272 1      IF FT < FH & MK=0 THEN GO TO RUN ;
274 1      TEST=FH ;

/* ***** NEIGHBORHOOD SEARCH OF LAST, BEST FEASIBLE VERTEX ***** */
/* ***** NEIGHBORHOOD SEARCH OF LAST, BEST FEASIBLE VERTEX ***** */
275 1      LK=1 ;
276 1      DO KJ=1 TO 2 ;
277 1 1      DO JJ=1 TO N ;
278 1 2      XT(JJ)=XT(JJ)+LK ;
279 1 2      CALL OBJ(XT,FT) ; MK=FEAS1(M,A,B,XT) ;
281 1 2      IF FT < TEST & MK=0 THEN DO ;
283 1 3          TEST=FT ; XB(*)=XT(*) ;
285 1 3          END ;
286 1 2      XT(JJ)=XT(JJ)-LK ;
287 1 2      END ;
288 1 1      LK=-1 ;
289 1 1      END ;
/* ***** */

290 1      IF TEST < FH THEN DO ;

```

STMT LEVEL NEST

```

292 1 1      XT(*)=XB(*) ; FT=TEST ; MK=0 ;
295 1 1      GO TO RUN ;
296 1 1      END ;
297 1        IF FH < FP & MH=0 THEN DO ;
299 1 1      X(1,*)=XH(*) ; F(1)=FH ; MARK(1)=MH ;
302 1 1      GO TO SLIDE ;
303 1 1      END ;
304 1 1      GO TO B3 ;
/* ***** */

305 1        PLACE: DO I=1 TO NN ;
306 1 1      IF MARK(I) < MK THEN GO TO P1 ;
308 1 1      DO K=1 TO NN ;
309 1 2      XH(*)=X(K,*) ; FH=F(K) ; MH=MARK(K) ;
312 1 2      X(K,*)=XT(*) ; F(K)=FT ; MARK(K)=MK ;
315 1 2      XT(*)=XH(*) ; FT=FH ; MK=MH ;
318 1 2      END ;
319 1 1      P1: END PLACE ;
320 1 1      GO TO B1 ;
321 1 1      TURN: X(NN,*)=XT(*) ; MARK(NN)=MK ;
323 1 1      K=1 ;
324 1 1      T1: IF NN-K <= 2 THEN ISUM(*)=ISUM(*)-X(NN-K,*)+X(NN-K+1,*) ;
326 1 1      XT(*)=ISUM(*)-X(NN-K,*) ;
327 1 1      MK=FEAS(M,A,B,XT) ;
328 1 1      CALL OBJ(XT,FT) ;
329 1 1      IF MK >= MARK(NN-K) & K < NN-1 THEN DO ;
331 1 1      X(NN-K,*)=XT(*) ; MARK(NN-K)=MK ;
333 1 1      K=K+1 ;
334 1 1      F(NN-K)=FT ;
335 1 1      GO TO T1 ;
336 1 1      END ;
337 1 1      IF MK >= MARK(NN-K) THEN DO ;
339 1 1      GO TO SOLVE ;
340 1 1      END ;
341 1 1      IF MK=0 THEN GO TO B2 ; ELSE GO TO PLACE ;

/* ***** */
/* ***** NEIGHBORHOOD SEARCH OF BEST VERTEX ***** */

344 1        SOLVE: KL=1 ;
345 1 1      DO K=1 TO 2 ;
346 1 1      DO JJ=1 TO N ;
347 1 2      XT(*)=X(1,*) ; XT(JJ)=XT(JJ)+KL ;
349 1 2      CALL OBJ(XT,FT) ; MK=FEAS(M,A,B,XT) ;
351 1 2      IF FT < FP & MK=0 THEN DO ;
353 1 3      X(1,*)=XT(*) ; GO TO MAG ;
355 1 3      END ;
356 1 2      XT(JJ)=XT(JJ)-KL ;
357 1 2      END ;
358 1 1      KL=-1 ;
359 1 1      END ;
/* ***** */

/* ***** */
/* ***** OUTPUT OF THE SOLUTION VERTEX ***** */

```

STMT LEVEL NEST

```

360      1          X(1,*)=XP(*) ; F(1)=FP ; MARK(1)=MP ;
363      1          KL=0 ;
364      1          DO J=1 TO NN ;
365      1      1      IF MARK(J) = 0 THEN GO TO S1 ;
367      1      1      KL=KL+1 ;
368      1      1      IF KL=1 THEN GO TO S2 ;
370      1      1      IF F(J) >= F(1) THEN GO TO S1 ;
372      1      1      S2: X(1,*)=X(J,*) ; F(1)=F(J) ;
374      1      1      S1: END ;
375      1          PUT PAGE ;
376      1          PUT SKIP(2) ;
377      1          PUT SKIP(2) ;
378      1          PUT LIST(NAME) ;
379      1          PUT SKIP(2) ;
380      1          PUT LIST('INTEGRAL SOLUTION VECTOR :') ;
381      1          DO J=1 TO N ;
382      1      1      PUT EDIT('X(' , J, ') = ' , X(1, J)) (SKIP, A, F(3, 0), A, F(8, 0)) ;
383      1      1      END ;
384      1          PUT EDIT('OBJECTIVE FUNCTION VALUE = ' , F(1)) (SKIP(2), A, F(10, 2)) ;

          /* ***** */

385      1          FEAS: PROCEDURE(P, A, B, XV) RETURNS(FIXED) ;
          /* ** BINARY VERTEX FEASIBILITY INDICATOR ** */
386      2          DECLARE M FIXED , A(14, 10) FIXED , B(14) FIXED , XV(10) FIXED ;
387      2          DECLARE VALUE FIXED ;
388      2          VALUE=0 ;
389      2          DO J=1 TO N ;
390      2      1      IF XV(J) >= 0 THEN GO TO FSTEP ; ELSE VALUE =-1 ;
393      2      1      GO TO FSTEP2 ;
394      2      1      FSTEP: END ;
395      2          DO I=1 TO M ;
396      2      1      IF B(I)-SUM(A(I, *)*XV(*)) <= 0 THEN GO TO FSTEP1 ;
398      2      1      VALUE =-1 ;
399      2      1      GO TO FSTEP2 ;
400      2      1      FSTEP1: END ;
401      2          FSTEP2: RETURN(VALUE) ;
402      2          END FEAS ;

403      1          RANK: PROCEDURE(NN, MARK, F, X) ;
          /* RANKING THE SIMPLEX VERTICES BY OBJECTIVE FUNCTION VALUE */
404      2          DECLARE NN FIXED , F(11) FLOAT(16) , X(11, 10) FIXED ;
405      2          DECLARE XH(10) FIXED , FH FLOAT(16) ;
406      2          DECLARE MARK(11) FIXED , MH FIXED ;
407      2          ADJ: DO J=1 TO NN-1 ;
408      2      1      DO K=J+1 TO NN ;
409      2      2      IF F(K) > F(J) THEN GO TO RSTEP1 ;
411      2      2      XH(*)=X(J, *) ; FH=F(J) ; MH=MARK(J) ;
414      2      2      X(J, *)=X(K, *) ; F(J)=F(K) ; MARK(J)=MARK(K) ;
417      2      2      X(K, *)=XH(*) ; F(K)=FH ; MARK(K)=MH ;
420      2      2      RSTEP1: END ;
421      2      1      END ADJ ;
422      2          RETURN ;
423      2          END RANK ;

```

STMT LEVEL NEST

```

424     1     OBJ: PROCEDURE(XT,FT) ;
425     2     /* COMPUTING THE VALUE OF THE OBJECTIVE FUNCTION AT A VERTEX */
426     2     DECLARE XT(10) FIXED ;
427     2     DECLARE FT FLOAT (16) ;
428     2     FT=XT(1)**2-8.*XT(1)+2.*XT(2)**2-16.*XT(2) ;
429     2     PUT SKIP ;
430     2     PUT EDIT((XT(KM) DO KM=1 TO N))((N))(F(6,0),X(2)) ;
431     2     PUT SKIP ;
432     2     PUT LIST(FT) ;
433     2     RETURN ;
433     2     END OBJ ;

434     1     FEAS1: PROCEDURE (M,A,B,XV) RETURNS(FIXED) ;
435     2     /* VERTEX FEASIBILITY INDICATOR BY VALUE OF INFEASIBLE SLACKS */
436     2     DECLARE M FIXED , A(14,10) FIXED , B(14) FIXED , XV(10) FIXED ;
437     2     DECLARE VALUE FIXED ;
438     2     VALUE=0 ;
439     2     DO JZ=1 TO N ;
440     2     1     IF XV(JZ) < 0 THEN VALUE=VALUE-XV(JZ) ;
441     2     1     END ;
442     2     DO I=1 TO M ;
443     2     1     VALUEI=B(I)-SUM(A(I,*)*XV(*)) ;
444     2     1     IF VALUEI <= 0 THEN GO TO F1 ;
445     2     1     VALUE=VALUE+VALUEI ;
446     2     1     F1: END ;
447     2     1     RETURN(VALUE) ;
448     2     END FEAS1 ;

450     1     FRANK: PROCEDURE (NN,MARK,F,X) ;
451     2     /* ** VERTEX RANKING BY DEGREE OF INFEASIBILITY ** */
452     2     DECLARE NN FIXED, MARK(11) FIXED, X(11,10) FIXED ;
453     2     DECLARE XB(10) FIXED, MB FIXED, FB FLOAT(16) ;
454     2     DECLARE F(11) FLOAT(16) ;
455     2     DO I=1 TO NN-1 ;
456     2     1     DO K=I+1 TO NN ;
457     2     2     IF MARK(K) > MARK(I) THEN GO TO FR1 ;
458     2     2     XB(*)=X(I,*) ; MB=MARK(I) ; FB=F(I) ;
459     2     2     X(I,*)=X(K,*) ; MARK(I)=MARK(K) ; F(I)=F(K) ;
460     2     2     X(K,*)=XB(*) ; MARK(K)=MB ; F(K)=FB ;
461     2     2     FR1: END ;
462     2     1     END ;
463     2     1     RETURN ;
464     2     END FRANK ;

471     1     END HOUND ;

```

VITA

Robert Pratt Davis

Candidate for the Degree of

Doctor of Philosophy

Thesis: INTEGER SEARCH UNDER LINEAR DIOPHANTINE  
CONSTRAINTS

Major Field: Industrial Engineering

Biographical:

Personal Data: Born in Hampton, Virginia,  
November 25, 1946, the son of Mr. and Mrs.  
D. G. Davis, Sr.

Education: Graduated from Newport News High School,  
Newport News, Virginia, in June, 1965;  
received Bachelor of Science degree in  
Industrial Engineering from the University of  
Tennessee in 1970; received Master of Science  
degree in Industrial Engineering from the  
University of Tennessee in 1971; completed  
requirements for the Doctor of Philosophy  
degree at Oklahoma State University in  
December, 1973.

Professional Experience: Engineering-aide, NASA-  
Langley, 1966; Engineering-aide, USAAVLABS-  
Fort Eustis, 1967; Industrial Engineer, ORTEC,  
1970; graduate research and teaching assistant,  
University of Tennessee, 1971; NDEA Fellow,  
Oklahoma State University, 1972; part-time  
instructor, Oklahoma State University, 1973.