INTEGER PROGRAMMING - THEORY AND PRACTICE.

by Donald Brian Cyril Richards.

Thesis submitted for the degree of Doctor of Philosophy in the University of London.

Institute of Computer Science.

November 1967.

ABSTRACT

This thesis is concerned with the study and development of integer programming techniques, that is techniques for the solution of linear programming problems wholly or partly in integers, and the applicability of these techniques in practice. Unfortunately, experimentation on large scale integer programming problems derived from real world situations has not proved feasible and computer experience has been limited to a wide range of test problems designed to exhibit all the characteristics of real integer programmes. All integer programming techniques of any real importance are believed to have been considered.

The order of presentation of the chapters is such that techniques belonging to a particular class, for example that of cutting plane methods, whether existing or original, are discussed together and in relation to each other.

Chapter I is historical and introductory, and sets the scene for what is to follow. Chapter II (CUTTING PLANE METHODS), Chapter IV (BRANCH AND BOUND METHODS) and Chapter VI (OTHER METHODS) are largely expository, the subject matter having been drawn from several sources all of which are fully detailed in the bibliography. Chapter III (ON EXTENSIONS TO CUTTING PLANE METHODS), Chapter V (EXTENSION OF BRANCH AND BOUND METHOD) and Chapter VII (PRACTICE) are claimed to be original, any dependence upon other work being clearly indicated. The techniques described and developed in Chapters II to V have been coded in FORTRAN

(listings are given in Appendix III) and the programs run on the ICT Atlas I computer at the University of London. The test problems are defined in Appendix I. Summarized results are presented and discussed where appropriate throughout the text. Complete results are presented in tabular form at the end of Chapter VII following a discussion of our conclusions.

ACKNOWLEDGEMENTS.

I would like to express my appreciation to the Commonwealth of Australia for awarding me a Postgraduate Research Scholarship and to the Australian Department of Supply for granting me leave in order to pursue a Ph.D. course of advanced study and research at the Institute of Computer Science in the University of London.

I am grateful to Professor R.A. Buckingham, firstly for accepting me as a Ph.D. student at his Institute, which has provided me with excellent computing facilities, and secondly for his encouragement.

I should like to record here my very sincere thanks to Mr. K. Wolfenden, M.A., who first introduced me to this field of study and who has been most helpful with his valuable criticisms and suggestions with regard to the presentation of this work.

The paper supplying the major section of the data employed in this work was made available to me by Professor E.M.L. Beale to whom I am also grateful for a discussion concerning the contents of the chapter describing a modification of the branch and bound method.

Finally, I gratefully thank Mrs. Jeanette Mullins for her work in typing the thesis, and my wife, Sheila, for her patience throughout the past three years.

CONTENTS

	NOTATION	9
CHAPTER I	INTRODUCTION.	10
1.1	General.	10
1.2	Euclidean space, convex sets.	12
1.3	Linear Programming.	15
1.4	Integer Programming.	19
1.5	Integer Programming in Practice.	27
CHAPTER II	CUTTING PLANE METHODS.	30
2.1	The All Integer Integer Method.	31
2.2	Selection of Generator Constraint.	36
2.3	Application to Problem Sample.	38
2.4	Pure Integer Method.	40
2.5	Variants of Generator Selection.	42
2.6	Experience with Method.	43
2.7	Mixed Integer Method.	47
2.8	Remarks on Finiteness Proofs.	52
CHAPTER III	ON EXTENSIONS TO CUTTING PLANE METHODS.	55
3.1	Modified Cutting Plane For The All Integer Integer Method.	56
3.2	Performance.	59
3.3	Optimal Cut For Pure Integer Method.	61
3.4	Diophantine Equation.	65
3.5	Illustration of Method.	67
3.6	Optimal Cut in Practice.	71
3.7	Difficulties and Possible Variations.	74

CHAPTER IV	BRANCH AND BOUND METHODS.	82
4.1	Dual Branching Tree Search Method.	84
4.2	Selection of Branching Variable.	88
4.3	An Example.	89
4.4	Multiple Optima.	95
4.5	Experience.	101
CHAPTER V	EXTENSION OF BRANCH AND BOUND METHOD.	104
5.1	Variable Selection.	105
5.2	Bound Contraction.	114
5.3	Convex Isolation.	117
5.4	Search for Superior Feasible Point.	120
5.5	Difficulties Within the Method.	125
5.6	Variants in Certain Areas of the Method.	129
5.7	The Method in Practice.	132
CHAPTER VI	OTHER METHODS	143
6.1	Primal Methods.	144
6.2	Partial Enumeration Methods.	168
6.3	Approximate Methods.	172
CHAPTER VII	PRACTICE.	181
7.1	Cutting Planes.	182
7.1.1	The All Integer Integer Code.	182
7.1.2	The Pure Integer Code.	185
7.1.3	The Mixed Integer Code.	191
7.1.4	The Accelerated Euclidean Algorithm.	194

7.1.5	The Optimal Cut Code.	198
7.2	Branch and Bound Methods.	203
7.2.1	Dual Branching Tree Search Code.	203
7.2.2	Convex Isolation and Bound Contraction.	209
7.3	Concluding Remarks.	215
7.4	Detailed Results.	218
TABLE 1	Structures of Problems.	220
TABLE 2	Performance of All Integer Integer Code using The Standard Cut and Modified Cut.	221
TABLE 3	Pure Integer Code With Combination of Variants Applied To First Fixed Charge Problem.	222
TABLE 4	Pure Integer Code With Combination of Variants Applied To Second Fixed Charge Problem.	223
TABLE 5	Pure Integer Code Using Variant (0,2).	224
TABLE 6	Pure Integer Code Using Variant (2,2).	225
TABLE 7	Optimal Cut Code Using Variant (0,2).	226
TABLE 8	Optimal Cut Code Using Variant (2,2).	227
TABLE 9	First Variant of the Branch and Bound Code.	228
TABLE 10	Second Variant of the Branch and Bound Code.	229
TABLE 11	Results from Three Variants of Version 0 of the Convex Isolation and Bound Contraction Code.	230

TABLE 12 Results from Three Variants of Version 1 of the Convex Isolation and Bound Contraction Code. 231

Bibliography. 232

Appendix I. Sample of Problems. 237

Appendix II. Flow Chart for the All Integer Integer Algorithm Including the Modified Constraint. 244

Flow Chart for the Optimal Cutting Plane Method.

Flow Chart for the Convex Isolation and Bound Contraction Method.

Appendix III. Program Listings. 263

REMARKS ON NOTATION

Before proceeding to the first chapter we consider it helpful to present a brief description of the notation adopted in the thesis.

$N = \{1,\ldots,n\}$, denotes a set of n column indices.

$P = \{a_o\} + A$, means that the set P consists of the element a_o together with all elements of A.

$N_1 \subset N$ means that N_1 is a subset of N.

$L \frown M$ and $L \smile M$ are, respectively, the intersection and the union of the sets L and M.

$[\alpha]$ denotes the mathematical integral part of α so that $\alpha = [\alpha] + f$ where $0 \leq f < 1$; f is called the fractional part of α and $\bar{f} = 1 - f$ is the compliment of f.

If a, b and c are integers then $a \equiv b$ modulo c means that $(a - b)$ is divisible by c. Furthermore, $a \equiv 0$ modulo 1 implies that a is an integer.

When we refer to a point - lattice $\mathcal{L}$ we mean, in particular, the points of intersection of the lines of the fundamental lattice L, that is the lattice formed by parallels to given rectangular co-ordinate axes at unit distances dividing the plane into unit squares.

A vector is lexicographically positive if its leading non-zero element is positive.

A vector A is lexicographically greater than a vector B (written $A \succcurlyeq B$) if the vector $(A - B)$ is lexicographically positive.

CHAPTER I

INTRODUCTION.

1.1 General

Programming problems of a wide variety arise in quite diverse areas of society, and new fields presenting such problems are being discovered every year. This diversity is perhaps best illustrated by the following three examples of categories in which programming techniques have been applied. The area of military application, the area of pure and applied mathematics and the area of economics, probably the major source for programming problems. Within each area, these problems emerge as a result of examining a structure representing a system of interrelated processes over which a particular objective is to be fulfilled. If the system and objective are representable by a mathematical equivalent then the problem is to select from the possible actions defined by the system that schedule for which the desired objective is attained.

The formulation and solution of problems of this type are embraced by the field of mathematical programming. If the formulation consists of linear inequalities and equations and the objective is measured by means of a linear function then the problem belongs to the field of linear programming. The term linear programming is frequently misunderstood since it is so often regarded solely as a technique for solving inequality systems. It should be emphasized that linear programming is more than this. It is a concept, one which is first concerned with describing the interrelations of the

components of a system whose structure can be mathematically represented by linear inequalities and equations, called constraints. The central mathematical problem of linear programming is one of maximizing or minimizing a linear function subject to these constraints. In the last twenty years linear programming has become one of the most widely used techniques in applied mathematics. This rapid progress has been made possible by the rate of development of high speed computers, which are essential to the practical use of linear programming. Both these fields, linear programming and high speed computing, have been considerably interdependent during their development.

Apart from the linear restrictions mentioned above, some problems have the further requirement that some or all of the variables used in defining the problem be integer valued. Some classical problems in this area are the flight scheduling problem, the fixed charge problem and the travelling salesman problem. Furthermore, there exists a large class of difficult mathematical problems which are reducible to problems of this type (see, for example, Dantzig [9]). The formulation and solution of a linear programming problem in which some or all of the variables must take integral values is called integer programming. Although strictly in the realm of nonlinear programming, it is in fact regarded as an extension of linear programming for reasons which will become apparent later.

The following sections commence with a brief introduction to some concepts and definitions of linear algebra. This is followed

by Sections 1.3 and 1.4 which present the definitions and formulations of the mathematical problems of linear and integer programming respectively. Finally, Section 1.5 briefly discusses the aspect of integer programming in practice.

1.2 Euclidean space, convex sets

Since the algebra of convex sets is fundamental to a proper understanding of linear programming, we shall commence by recalling a number of definitions and results which will be given without proof.

An m - vector P is an ordered set of m real numbers $(a_1, a_2, \ldots, a_m)$, its components.

A set of r vectors $P_1, P_2, \ldots, P_r$ is linearly independent if for all scalars x_i,

$$\sum_{i=1}^{r} x_i P_i = 0$$

implies $x_i = 0$, $i = 1, =, \ldots, r$. Otherwise it is linearly dependent.

If

$$\sum_{i=1}^{n} x_i P_i = P_o,$$

then P_o is linearly dependent on the system $\{P_j : j = 1, \ldots, n\}$ and the set of m - vectors P_o generated by the set of all possible n-vectors $X = (x_1, x_2, \ldots, x_n)$ is a vector space closed under the operations of addition and multiplication by a scalar. The rank or dimensionality of a vector space is the largest number of linearly independent vectors in the space.

The set of all possible m - vectors $P = (a_1, a_2, \ldots, a_m)$, structured by the matrix-wise operations of addition and scalar

multiplication, is an m - dimensional euclidean space E^m such that there exist m linearly independent vectors, but every set of $n > m$ vectors is linearly dependent.

A basis for E^m is any set, a minimum spanning set, of exactly m linearly independent vectors and any other vector Q in E^m can be represented uniquely as a linear combination of the vectors of a given basis, $Q = \sum_{i=1}^{m} x_i P_i$; Q is degenerate if one or more $x_i = 0$. If $R \neq 0$ is any vector in the vector space, basis $P_1, P_2, \ldots, P_m$, then

$$R = \sum_{i=1}^{m} y_i P_i$$

and if, say, $y_1 \neq 0$ a new adjacent basis for the space is $R, P_2, \ldots, P_m$.

The rank of a matrix $A = (P_1\ P_2\ \ldots\ P_n)$ is the rank of the vector space generated by its columns $P_1, P_2, \ldots, P_n$; it is the same as the rank of its rows $A_1, A_2, \ldots, A_m$. It is also then the maximum number of independent equations in the consistent system $AX = b$. If the rank of A is m and columns $P_{j1}, P_{j2}, \ldots, P_{jm}$, say, are independent then

$$\det B = \det (P_{j1}\ P_{j2}\ \ldots\ P_{jm}) \neq 0$$

and $x_B = B^{-1}b$ gives the values of the basic variables $x_{j1}, \ldots, x_{jm}$ in a basic solution of $AX = b$, all the $(n - m)$ nonbasic variables taking zero values.

A linear equation in E^n, $A_iX = b_i$ ($A_i \neq 0$, real scalar b_i) is called a hyperplane; it divides E^n into two half-spaces, say one open

$$H^+ (A_i, b_i) = \{ X \mid A_iX > b_i \}$$

and one closed

$$H^{-}(A_i, b_i) = \{ X \mid A_i X \leq b_i \}.$$

A convex combination of points $X_1, X_2, \ldots, X_n$ is a point

$$X = \sum_{i=1}^{n} \alpha_i X_i,$$

where the scalars $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$.

A subset C of E^n is convex if, and only if, for all pairs of points X_1 and X_2 in C,

$$X = \alpha_1 X_1 + \alpha_2 X_2 \qquad (\alpha_1, \alpha_2 \geq 0 \text{ and } \alpha_1 + \alpha_2 = 1)$$

is also in C; any convex combination of any number of points in C is then also in C.

Any point on the line segment joining two points in E^n can be expressed as a convex combination of the two points, and conversely. Geometrically then a convex set is one which contains all the line segments joining any two points of the set. The convex set may be bounded or unbounded in extent and may or may not include its boundary. The set of points common to two or more convex sets is itself convex. Hence a hyperplane, being the intersection of two half-spaces, is convex. A convex polyhedron is the intersection of a finite number of half-spaces. An extreme point or vertex of a convex set C is a point X which cannot be expressed as a convex combination of any other two distinct points in C. An edge of a convex polyhedron is the line segment joining two extreme points such that no point on the segment is a convex combination of two other points of the set not on the segment; the relevant extreme points are then adjacent or neighbours.

A supporting hyperplane for the convex polyhedron C is one through a vertex, an edge or a face such that all of C lies in one closed half-space produced by the hyperplane.

The convex hull C(S) of a set of points S is the set of all convex combinations of sets of points from S; it is the smallest convex set containing S. If S consists of a finite number of points, C(S) is a closed convex polyhedron spanned by these points. It is a closed strictly bounded convex set with a finite number of extreme points.

Given the point X in E^n, a ray is the convex set of points λX, $\lambda \geq 0$. A cone is a set of points S such that if X is in S, so is every point on the ray of X; it is not necessarily convex. However, if C is the set of all points

$$X = \sum_{i=1}^{r} \lambda_i x_i \quad , \lambda_i \geq 0,$$

then C is a convex polyhedral cone.

A real-valued function $f(X) = f(x_1, x_2, \ldots, x_n)$ defined on an n - dimensional vector space is linear if for every vector

$$X = \alpha_1 X_1 + \alpha_2 X_2, \; f(X) = \alpha_1 f(X_1) + \alpha_2 f(X_2)$$

for all n - vectors X_1, X_2 and scalars α_1, α_2.

1.3 Linear Programming

It has already been mentioned that linear programming is concerned, firstly, with describing the interrelations of the components of a system whose structure can be mathematically represented by linear inequalities and equations. It is con-

cerned, secondly, with solving the system in non-negative variables in such a way that a linear function, called the objective function, attains its optimal value for this solution. This thesis deals entirely with the second of these which we refer to as the linear programming problem.

The general linear programming problem is to maximize or minimize the linear function

$$a_{o1} x_1 + a_{o2} x_2 + \cdots\cdots + a_{on} x_n \tag{1.1}$$

subject to the linear constraints

$$\sum_{j \in N} a_{ij} x_j = a_{io}, \qquad i \in M = \{1, \ldots, m\}, \tag{1.2}$$

$$N = \{1, \ldots, n\},$$

and

$$x_j \geq 0, \qquad j \in N, \tag{1.3}$$

where all a_{ij} have given numerical values and $m < n$.

A solution is any set of values of the variables x_j satisfying (1.2) but not necessarily satisfying (1.3). A solution which satisfies both (1.2) and (1.3) is called a feasible solution and a feasible solution which contains not more than m variables at positive level is a basic feasible solution. If the basic feasible solution contains fewer than m variables with positive values then it is called degenerate. An optimal feasible solution is one which satisfies (1.2) and (1.3) and, at the same time, maximizes or minimizes the linear function (1.1).

This formulation can be expressed in a variety of ways, a common one being the following:

Maximize or minimize

$$A_o X$$

subject to

$$x_1 P_1 + x_2 P_2 + \ldots\ldots + x_n P_n = P_o$$

and $$X \geq 0$$

where $A_o = (a_{o1}, \ldots\ldots, a_{on})$ is a row vector, $X = (x_1, \ldots\ldots, x_n)$ and $P_j = (a_{1j}, \ldots\ldots, a_{mj})$ are column vectors.

The relationship between this algebraic problem and the theory of convex sets, as outlined in the previous section, is illustrated by the following results.

The set of all feasible solutions to the linear programming problem is a convex set R. Unless it is void, when the linear programming problem is infeasible, R is a convex polyhedron in E^n bounded by a finite set of hyperplanes and with a finite number of extreme points. If the polyhedron is closed, R is the convex hull of its extreme points, every feasible solution in R can be expressed as a convex combination of the extreme point feasible solutions and the linear programming problem has a solution with a finite maximum (minimum) for the objective function. If the polyhedron is open in some direction, R is not the convex hull of its extreme points, the problem still has a solution but the maximum (minimum) value of the objective function may be unbounded.

The objective function assumes its finite maximum (minimum) at an extreme point of R. If it does so at more than one such point it assumes the same value for every convex combination of those points.

If a set of $r \leq m$ linearly independent m - vectors $P_1, P_2 \ldots\ldots, P_r$

is such that

$$P_1 x_1 + P_2 x_2 + \ldots\ldots + P_r x_r = P_o$$

and $x_j \geq 0$

then $X = (x_1, \ldots., x_r, 0, \ldots., 0)$ is an extreme point of R. X is an n - dimensional vector whose last (n - r) elements are zero.

Conversely, if X is an extreme point of R, the r vectors P_j associated with the positive x_j form a linearly independent set; at most m of the x_j are positive and X is a basic feasible solution.

So $X = (x_1, x_2, \ldots., x_n)$ is an extreme point of R if, and only if, the $x_j > 0$ are the coefficients of the linearly independent vectors P_j in $\sum_{i=1}^{n} x_i P_i = P_o$.

Every basic feasible solution thus corresponds to an extreme point of R and associated with every extreme point of R is at least one set of just m linearly independent vectors from the original set $P_1, P_2, \ldots., P_n$.

The search for the optimal feasible solution to the linear programming problem can now be limited to the extreme points of R, i.e. those feasible solutions generated by sets of linearly independent vectors P_j. There are, at most, ${}^{n}C_{m}$ such sets. A directed search is achieved by the simplex method which seeks a sequence of vertices at which the corresponding values of the objective function are strictly non-decreasing (for a problem of maximization), until the process can be continued no further and the optimum is reached in a finite number of steps. The simplex method also shows whether

the problem has no finite optimal solution or no feasible solution.

These basic ideas are more fully described in any of the texts [9], [20] and [32].

We find it convenient to define the central mathematical problem of linear programming as that of maximizing the linear function

$$x_o = a_{oo} + \sum_{j \in N} a_{oj} (-x_j) \quad , N = \{1, \ldots, n\} \tag{1.4}$$

subject to the linear constraints

$$\sum_{j \in N} a_{ij} x_j \leq a_{io} , \quad i \in M = \{1, \ldots, m\}, \tag{1.5}$$

and

$$x_j \geq 0 \quad , \quad j \in N, \tag{1.6}$$

where all a_{ij} have given numerical value.

As before, an optimal feasible solution is one which satisfies (1.5) and (1.6) and, at the same time, maximizes the linear function (1.4).

We note here that this formulation of the problem can be transformed by elementary operations into the form of (1.1), (1.2) and (1.3).

1.4 Integer Programming

The integer programming problem can be defined as the linear programming problem (1.4), (1.5) and (1.6) with the added constraint that some of the variables x_j must take integer values. Explicitly,

$$x_j \equiv 0 \text{ modulo } 1 \quad , \quad j \in K, \tag{1.7}$$

where K defines an integer subspace of the continuous space. If K contains all j, the problem is said to be a pure integer programming problem while if K contains a subset of j it is a mixed integer programming problem. We should note here that if the inequality con-

straints (1.5) contain rational but non-integral coefficients a_{ij} then the slack variables introduced to form a starting basis $(x_{n+1}, \ldots, x_{n+m})$ would not be restricted to integer values. However, this is simply remedied, and the previous definition of a pure integer programming problem satisfied, by multiplying each inequality by the least common multiple of the denominators of the coefficients before adding the slack variables.

This is an appropriate point to emphasize that, in the following, the term feasible solution refers to a solution which is a true feasible solution i.e. one which satisfies all the given restrictions. If any solution is non-negative but violates some integrality constraints then we refer to such a solution as a continuous feasible solution.

In comparison with the linear programming problem the solution of the integer programming problem is difficult, a major factor being the discrete non-convex nature of the integer space which inhibits complete solution by the simplex type methods of linear programming. If there were some process whereby we could construct the convex hull of the feasible space from the given information (1.5), (1.6) and (1.7) then, clearly, the simplex method of linear programming would produce the feasible optimum in the manner already mentioned. Unfortunately, such a general process has not proved capable of development as yet, although one approach to the solution of the integer programming problem does produce an approximation to the convex hull in the neighbourhood of the optimal vertex.

Frequently, in practice, the integrality requirement is initially

ignored and the solution of the continuous problem then defined is rounded to the nearest integer solution satisfying the constraints. When the solution variables are large enough for the rounding to have negligible effect, this would be valid, but such a situation cannot be taken as general so other solution techniques are necessary. Although strictly nonlinear, the integer programming problems have been handled in such a way that the process of their solution is treated as an extension of the linear programming problem. In fact, apart from the approximate method, the solution techniques to be described in the following chapters consist of solving a series of linear programming problems. The major procedure used for progressing to the feasible optimum is the dual simplex method and it would seem appropriate to present a brief description of it at this stage.

The linear programming problem (1.4), (1.5) and (1.6) can be conveniently displayed in tableau form but since we have elected to give all variables, including slack variables, row representation in the natural order of their indices some minor modifications are necessary. Firstly, by changing the range of the index i in (1.5) from (1, m) to (n + 1, n + m) and by the addition of slack variables $x_i \geq 0$ we transform these inequality constraints into the equations

$$\sum_{j=1}^{n} a_{ij} x_j + x_i = a_{io}, \qquad i = n + 1, \ldots, n + m.$$

This can be written in the form

$$x_i = a_{io} + \sum_{j=1}^{n} a_{ij} (-x_j), \quad i = n + 1, \ldots, n + m.$$

By adding to this and to (1.1) the trivial set

$$x_j = -1 (-x_j), \qquad j = 1, \ldots, n,$$

we can represent the total system by

$$x_i = a_{io} + \sum_{j=1}^{n} a_{ij}(-x_j) \quad , \quad i = 0,1, \ldots, n+m,$$

where $a_{ij} = -\delta_{ij}$ ($\delta_{ij} = 1$ if $i = j$, $\delta_{ij} = 0$ otherwise) if $i \in N = \{1, \ldots, n\}$. This total system can now be displayed in the tableau form

	1	$-x_1$	$-x_2$	$\ldots$	$-x_n$
x_o	a_{oo}	a_{o1}	a_{o2}	$\ldots$	a_{on}
x_1	0	-1	0	$\ldots$	0
x_2	0	0	-1	$\ldots$	0
$\vdots$					
x_n	0	0	0		-1
x_{n+1}	$a_{n+1,o}$	$a_{n+1,1}$	$a_{n+1,2}$	$\ldots$	$a_{n+1,n}$
$\vdots$					
x_{n+m}	$a_{n+m,o}$	$a_{n+m,1}$	$a_{n+m,2}$	$\ldots$	$a_{n+m,n}$

(1.8)

which, due to the inclusion of the trivial rows $x_j = -1(-x_j)$, $j \in N$, is the explicit form of the Tucker diagram. Here $x_{n+1}, \ldots, x_{n+m}$ are basic and $x_1, \ldots, x_n$ nonbasic variables; x_o represents the objective function but can conveniently be regarded as another basic variable.

A simple change of basis through which a current basic variable becomes basic in its place is effected by a pivot operation (Gaussian transformation). This pivot operation, which is dependent upon a pivot element selected by certain rules, transforms tableau (1.8) into another of the same form with the row labels of the variables being maintained while the column label of one nonbasic variable is

changed. This change of tableau is frequently referred to as the updating of the tableau. In the following, the unprimed letters denote the elements of the current tableau while the primed letters denote the elements of the updated tableau. Let a_{ij}, $i = 0, 1, \ldots, n + m$; $j \in N$, be any element of the current tableau and take a_{rk} as the pivot. The pivot operation which interchanges the roles of the variable x_r, assumed basic, and the k-th non-basic variable is defined by the transformation formulae

$$a'_{ij} = a_{ij} - \frac{a_{ik}}{a_{rk}} \cdot a_{rj}, \quad i \neq r, \; j \neq k,$$

$$a'_{rj} = 0, \quad j \neq k,$$

$$= -1, \quad j = k$$

and $$a'_{ik} = -\frac{a_{ik}}{a_{rk}}, \quad i \neq r.$$

As stated earlier, the row labels of the basic variables in (1.8) are unaltered but the column label $-x_k$ becomes $-x_r$.

The dual simplex algorithm, which effects a solution of the primal problem by solving the dual problem in terms of the primal tableau, requires a starting system which is dual feasible i.e. $a_{oj} \geq 0$, $j \in N$. Through a series of pivot operations it generates a sequence of tableaux each with the property $a_{oj} \geq 0$, $j \in N$, until one with the additional property $a_{io} \geq 0$, $i \in I = \{1, \ldots, n + m\}$ is reached. This is the optimal tableau and the optimal solution to the linear programming problem is $(a_{oo}, a_{1o}, \ldots, a_{n+m,o})$, the first column of the tableau.

In any non-optimal tableau there will exist at least one row,

$r > 0$, with $a_{ro} < 0$ and basic variable x_r. Two situations are now possible, either $a_{rj} \geq 0$, $j \in N$, in which case there is no solution due to the noncompatability of the constraints, or some $a_{rk} < 0$ for at least one $k \in N$. In the latter case we choose the k-th nonbasic variable to enter the basis in place of x_r by the criterion

$$\left|\frac{a_{ok}}{a_{rk}}\right| = \min_{\substack{a_{rj} < 0 \\ j \in N}} \left\{ \left|\frac{a_{oj}}{a_{rj}}\right| \right\}, \tag{1.9}$$

which simultaneously defines the pivot a_{rk}. A number of different rules for resolving the situations whenever k is not unique are discussed in later chapters. It suffices to say here that a unique choice is always attainable.

The convergence proof for one major method for the solution of integer programming problems requires that the system (1.8) be dual feasible in the lexicographic sense, i.e. 1 - dual feasible. Such a restriction requires not only that $a_{oj} \geq 0$ but that the column vectors associated with the nonbasic variables be lexicographically non-negative, a vector being lexicographically non-negative if its first non-zero component is positive. If a system is not 1 - dual feasible it can be made so in a number of ways.

Restating the integer programming problem as the maximization of the function

$$x_o = a_{oo} + \sum_{j \in N} a_{oj}\,(-t_j) \tag{1.10}$$

subject to

$$x_i = a_{io} + \sum_{j \in N} a_{ij}\,(-t_j) \quad , \quad i \in I = \{1, \ldots, n + m\}, \tag{1.11}$$

where $a_{ij} = -\delta_{ij}$

$$x_j \geqslant 0 \quad , \quad j \in I \tag{1.12}$$

and $$x_j \equiv 0 \text{ modulo } 1, \quad j \in K \subset I \ , \tag{1.13}$$

the nonbasic variables x_j have been replaced by t_j.

In the above formulation, if K is the entire set I then the problem is a pure integer problem. It is an all integer integer problem if, in addition, the a_{ij} are all integral. If K is a proper subset of I then the problem is a mixed integer problem. Clearly, if K is empty then we have an ordinary linear programming problem.

The methods capable of effectively solving such problems can be classified primarily into four groups. These are cutting plane methods, branch and bound methods, primal methods and partial enumeration methods. Other methods do exist, particularly for problems exhibiting special structures, but we deal with the four groups mentioned, concentrating on the first two of these.

Chapters II and III are confined to presenting and discussing cutting plane methods. Chapter II is devoted to the mathematically sophisticated approach of Gomory [15], [16] and [17]. Early work in this field was done by Dantzig [10] and Markowitz and Manne [28] but Gomory [15] provided, in 1958, the first method of solution of general integer programmes which guaranteed convergence to an optimum in a finite number of steps. With the exception of the All Integer Integer Method, the cutting plane techniques initially ignore the integrality requirement, i.e. K is assumed empty, and determine the continuous optimum of the ordinary linear programming problem. K is then

restored and the iterative process of the integer phase commenced. An integer phase iteration consists of generating a constraint, which is violated by the current continuous solution but not by any feasible solution, and applying the dual simplex algorithm to the current tableau augmented by this constraint until $a_{io} \geq 0$, $i \in I$. The process terminates when a feasible solution is attained. Chapter III is confined to modifications of two of the methods presented in Chapter II.

Branch and bound methods are presented in Chapters IV and V. As with the cutting plane methods, the earlier chapter discusses existing techniques while the latter presents and discusses an extension of one of these. Basically the branch and bound idea is very simple. Given a problem containing bounded discrete variables, a set of subproblems is generated in which upper and lower bounds are imposed on the values of the integer variables. These are solved and their solutions compared to determine the optimal solution. If the search among the subproblems could be directed so as to produce the optimal solution quickly then the number of subproblems generated would be considerably decreased due to the fact that some subproblem generator will have an objective function value which is exceeded by the currently held best value. The results of early work in this direction were presented by Land and Doig [26]. Although successful in hand computations its implementation on a computer would be difficult. Due to this, other methods, basically of similar idea, have been developed by Dakin [8] and Beale and Small [5].

Methods which have not been investigated in detail are those under the headings of primal methods and partial enumeration methods. These methods, with which we have had no computational experience but which, we feel, should be presented in order to complete the previously defined four theoretical groups, are confined to discussion in Chapter VI.

Finally, Chapter VII is concerned with practical experience and presents and discusses the codes used, their variants, the sample of problems and their solutions by particular codes. Chapter VII concludes with a detailed set of tables. The first defines the structures of the problems in the sample and the remainder give details of results produced by the application of the codes to this sample.

1.5 Integer Programming in Practice

Before proceeding to the next chapter it might be well at this point to give a brief discussion of integer programming in practice.

As was mentioned earlier, it is often a valid exercise in practice to round the solution of the problem in which integrality restrictions have been ignored to the nearest integer solution satisfying the constraints. The validity of this exercise of course being dependent upon the integer variables having sufficiently large values for the rounding to have negligible effect. However, many integer programming problems do not satisfy this situation and these are probably the most interesting. Dantzig [11] presents a comprehensive survey of many practical problems formulated as integer programming problems. Of course, it sometimes happens that the solution to the linear programming

problem (1.4), (1.5) and (1.6) also satisfies (1.7) and, in fact, for some classes of problems this is always true. For example, the transportation problem with integer valued demand and supply elements belongs to one such class. In problems of this class the constraint matrices possess a special structure in which all subdeterminants have the value 0, + 1 or -1. This leads to the pivot element of a simplex algorithm pivot operation having the value ± 1 so that the elements of the constraints matrix remain integral throughout the pivot operations. Geometrically, the convex region associated with such a constraint matrix has its vertices at lattice points. Some problems when formulated as integer programming problems, although mathematically correct, present computational difficulties due to the large numbers of constraints and variables required to define the problem. This appears to be the case for some scheduling problems. Wiest [38], for example, has found that a project with 55 jobs in 4 shops with a time span of 30 days has some 5275 equations and 1650 variables (excluding slack variables and the additional equations and variables necessary to assure an integral solution). Experience of this sort has discouraged many from applying integer programming techniques.

The methods which we describe in the succeeding chapters by no means exhaust the possible techniques for solving practical problems. Commercial airlines have reported success with various techniques applied to aircraft and crew scheduling problems. For example, the rounding process, mentioned above, has been successfully used by United Airlines in solving problems of more than 100

constraints and 2,000 variables. The Hungarian method and an algorithm based on Monte Carlo methods have been successfully employed by Air France and Swissair respectively. However, the methods considered in this project are probably the most widely used and important ones that presently exist.

Our practice has been concerned primarily with the application of two classes of methods to a sample of thirty test problems drawn from various sources. The data sample does not contain any problem which could be considered large but does contain many which have proved difficult for at least one member of one class of methods. Although a number of the codes produced for executing the methods to be described have been written to cater for mixed integer problems they have, due to circumstances regarding data availability, been applied to pure problems. For test purposes during their development they were, of course, applied to mixed problems but the results are not included in the discussion on the experiments in Chapter VII for reasons which will be apparent there.

CHAPTER II

CUTTING PLANE METHODS.

Early work in integer programming was done by Dantzig [10] and Markowitz and Manne [28]. However, the first method which was capable of producing solutions to general linear programming problems in integers, and further which was proved to be finite, was due to Gomory [15] in 1958.

Given that $\max(x_0)$ is the optimal value of a linear functional x_0 defined over a convex region C, enclosing a feasible point lattice K, and that this optimum occurs at a vertex V of C, the approach common to all the methods to be described in this chapter consists of adding linear inequality constraints to the system so that non-integral extreme points of C in the neighbourhood of V are excluded but the set of extreme points of the convex hull C(K), of admissible solutions is always in the new convex. The bounding hyperplane of an added inequality constraint is called a cutting plane. This is, in fact, the idea that was applied by Markowitz and Manne, for example, to specific problems but their generation of the cutting planes was unsystematic. The Gomory approach generates the added constraints systematically and is consequently most satisfactory from the point of view of computer implementation.

The original algorithm of Gomory (The Method of Integer Forms) is applicable to pure integer problems but the same basic ideas have been used by him to develop an algorithm [17] for mixed integer problems. A further algorithm [16] for the solution of pure integer

problems has been developed. Called the All Integer Integer Algorithm, it requires all the elements of the original matrix to be integers and this property is maintained throughout the calculation. This clearly avoids any problems associated with rounding. These methods are described in the following sections. Associated with each method are three sections the first of which describes the method, the second presents possible variations within the method and the third discusses our practical experience with the method concerned.

Gomory's proofs of finiteness for the algorithms, which are dependent upon both a particular rule for selecting a variable from which to produce a cutting plane and lexicography, are not given but in Section 2.8 we discuss the conditions necessary for the proofs to be valid.

2.1 The All Integer Integer Method

The All Integer Integer Method requires the initial tableau to satisfy certain conditions: it must be all integer and lexicographically dual feasible. The problem as defined by (1.10), (1.11), (1.12) and (1.13) thus has the added restrictions

(i) $a_{ij} \equiv 0$ modulo 1, $i \in \{0\} + I$, $j \in \{0\} + N$,

(ii) the subset K is the entire set I,

and (iii) the columns P_j, $j \in N$, are lexicographically positive or zero,

i.e. $P_j \succcurlyeq 0$, $j \in N$,

where P_j is the column vector of elements a_{ij}, $i \in \{0\} + I$.

This third restriction signifies that the leading non-zero element of P_j must be positive.

If $a_{io} \geq 0$, $i \in I$, then the problem is already solved with the solution vector given by $(a_{oo}, a_{1o}, \ldots, a_{n+m,o})$. Otherwise, at least one $a_{io} < 0$, $i \in I$.

The technique is to introduce a succession of new variables into the solution in such a way that we move towards the optimum while preserving the integral characteristics of the tableau. This is done by determining a series of cutting planes, each defined in terms of the current nonbasic variables and a new slack vector. Since the dual simplex method is used to change solutions, the introduction of the new slack vector into the basis with a pivot element of -1 preserves the integer tableau and moves the solution point closer to the optimum. A finite number of applications of new constraints leads to the optimum.

Consider an equation from the subset defining the basic variables, say,

$$x_i = a_{io} + a_{i1}(-t_1) + \ldots\ldots + a_{in}(-t_n). \tag{2.1}$$

Dividing through by $\lambda > 0$ and rearranging, we obtain

$$\frac{1}{\lambda} x_i + \frac{a_{i1}}{\lambda} t_1 + \ldots\ldots + \frac{a_{in}}{\lambda} t_n = \frac{a_{io}}{\lambda}. \tag{2.2}$$

Let
$$a_{ij} = \left[\frac{a_{ij}}{\lambda}\right]\lambda + f_{ij}, \quad j \in \{0\} + N,$$
$$1 = \left[\frac{1}{\lambda}\right]\lambda + f,$$

where
$$0 \leq f_{ij} < \lambda, \quad 0 \leq f < \lambda,$$

then (2.2) may be written as

$$\left(\left[\frac{1}{\lambda}\right] + \frac{f}{\lambda}\right) x_i + \sum_{j \in N}\left(\left[\frac{a_{ij}}{\lambda}\right] + \frac{f_{ij}}{\lambda}\right)t_j = \left[\frac{a_{io}}{\lambda}\right] + \frac{f_{io}}{\lambda}. \tag{2.3}$$

Any non-negative integer values for x_i and t_j which satisfy (2.1) will also satisfy (2.3) so we have that

$$\frac{f}{\lambda} x_i + \sum_{j \in N} \frac{f_{ij}}{\lambda} t_j \equiv \frac{f_{io}}{\lambda} \text{ modulo } 1, \tag{2.4}$$

in which the left hand side is clearly non-negative. Then it follows that

$$\frac{f}{\lambda} x_i + \sum_{j \in N} \frac{f_{ij}}{\lambda} t_j \geqslant \frac{f_{io}}{\lambda}, \tag{2.5}$$

and so from (2.3)

$$\left[\frac{1}{\lambda}\right] x_i + \sum_{j \in N} \left[\frac{a_{ij}}{\lambda}\right] t_j \leqslant \left[\frac{a_{io}}{\lambda}\right] \tag{2.6}$$

into which we can introduce a non-negative slack integer variable s to give

$$\left[\frac{1}{\lambda}\right] x_i + \sum_{j \in N} \left[\frac{a_{ij}}{\lambda}\right] t_j + s = \left[\frac{a_{io}}{\lambda}\right]$$

or

$$s = \left[\frac{a_{io}}{\lambda}\right] + \sum_{j \in N} \left[\frac{a_{ij}}{\lambda}\right](-t_j) + \left[\frac{1}{\lambda}\right](-x_i), \tag{2.7}$$

which is the new variable introduced by Gomory.

When $\lambda = 1$, equation (2.7) becomes

$$s = \left[a_{io}\right] + \sum_{j \in N} \left[a_{ij}\right] (-t_j) - x_i$$

or, using (2.1),

$$s = -f_{io} - \sum_{j \in N} f_{ij} (-t_j) \tag{2.7a}$$

which is an equation to be discussed in succeeding sections.

Consider the case $\lambda > 1$, which removes the variable x_i from (2.7) to give

$$s = \left[\frac{a_{io}}{\lambda}\right] + \sum_{j \in N} \left[\frac{a_{ij}}{\lambda}\right] (-t_j). \tag{2.8}$$

For any $\lambda > 1$ equation (2.8) represents a constraint which must be satisfied by any integer solution to the original linear programming problem, so that, after suitable selection of λ, (2.8) can be adjoined to the system as a possible pivot row.

By definition, $a_{io} < 0$ so we must seek a pivot element from among $\left[\frac{a_{ij}}{\lambda}\right]$ with $a_{ij} < 0$. The restriction that must be satisfied by the pivot row is that it must contain a pivot element of -1. Clearly, for sufficiently large λ, all $\left[\frac{a_{ij}}{\lambda}\right]$, $a_{ij} < 0$, in (2.8) become -1 and the pivot can only be -1. Thus it would be sufficient to take any λ satisfying

$$\lambda \geqslant \max_{j \in N} (- a_{ij}) .$$

Such a value of λ is not, however, adopted. Suppose a pivot column k is selected so that $\left[\frac{a_{ik}}{\lambda}\right] = -1$ in (2.8). Then the new value of the objective function is

$$x_o = a_{oo} + a_{ok} \left[\frac{a_{io}}{\lambda}\right]$$

so that its value decreases. We also observe that the smaller we take λ the greater the decrease in the value of the objective function. In the dual simplex method one moves outside the convex region and x_o decreases in order to attain its maximum. The rate of convergence is therefore related to the rate of decrease of the value a_{oo}. This would imply that an appropriate requirement for choosing λ is that it should be as small as possible subject to the condition that it still produces a pivot element of -1. Gomory, in fact, shows that, to produce as large a change as possible in the constants column, λ should be chosen as small as possible but he also points out that such a choice does not necessarily produce a cut superior to its generating constraint.

We now turn to the determination of λ. Suppose that (2.8) represents the pivot row and suppose that R is the set of indices

j such that $a_{ij} < 0$. Since the lexicographical dual simplex method is used for transforming tableaux the pivot column k is defined by

$$\frac{1}{-\left[\frac{a_{ik}}{\lambda}\right]} P_k = \underset{j \in R}{1\text{-}\min} \left\{ \frac{1}{-\left[\frac{a_{ij}}{\lambda}\right]} P_j \right\}, \tag{2.9}$$

where 1 -min denotes lexicographical minimum.

A necessary condition for the pivot to be $\left[\frac{a_{ik}}{\lambda}\right]$, and equal to -1, is then

$$-\frac{1}{\left[\frac{a_{ij}}{\lambda}\right]} P_j \succcurlyeq P_k . \tag{2.10}$$

Let h_j be the largest integer for which $(1/h_j)\ P_j \succcurlyeq P_k$. Then for P_k to be the pivot column we must have

$$-\left[\frac{a_{ij}}{\lambda}\right] \leq h_j$$

which implies

$$\lambda \geqslant -\frac{a_{ij}}{h_j} . \tag{2.11}$$

Now this must be satisfied for all P_j, $j \in R$, so we have

$$\lambda_{min} = \max_{j \in R} \left(-\frac{a_{ij}}{h_j} \right) . \tag{2.12}$$

That condition (2.11) is sufficient is easily seen because such a choice of λ leads to the selection of P_k as pivot column and for P_k we have $h_k = 1$ so

$$\lambda_{min} = -a_{ik}$$

and this implies that

$$\left[\frac{a_{ik}}{\lambda_{min}}\right] = -1.$$

To summarize, we describe in detail the steps necessary to progress from the stage of being given a row with $a_{io} < 0$ and $a_{ij} < 0$ for at least one $j \in N$.

(i) Select a column k such that $a_{ik} < 0$ and $a_{ok} = \min_{j \in R} a_{oj}$.

If k is not unique, suppose it defines a set $R_1 \subset R$ then choose $a_{1k} = \min_{j \in R_1} a_{1j}$. If again k is not unique continue this process until $a_{sk} = \min_{j \in R_s} a_{sj}$ does resolve the choice of k.

This step is equivalent to (2.9).

(ii) The next step is to choose the largest integer h_j satisfying

$$\frac{1}{h_j} P_j \succcurlyeq P_k. \tag{2.13}$$

If $a_{ok} \neq 0$ then

$$h_j = \left[\frac{a_{oj}}{a_{ok}} \right]$$

unless $a_{oj} \equiv 0$ modulo a_{ok} and $P_j \prec h_j P_k$ in which case

$$h_j = \left[\frac{a_{oj}}{a_{ok}} \right] - 1.$$

If P_j and P_k begin with an unequal number of zeros then h_j may be arbitrarily large.

If P_j and P_k each have p zeros then h_j is given by the above relations with a_{oj} and a_{ok} replaced by a_{pj} and a_{pk}.

(iii) Apply (2.12) to determine λ_{min}.

This method eliminates rounding problems and is straightforward in application.

2.2 Selection of Generator Constraint

The method as described in the previous section is susceptible to a number of variations. As already stated, the λ as defined does not necessarily produce the strongest cutting plane and it may therefore be possible to improve the choice of λ. This investigation is discussed

in the next chapter. When seeking a constraint from which to construct a new constraint there are, in general, a number of candidates satisfying the conditions outlined in the previous section. To satisfy Gomory's finiteness proof the choice is strictly defined but, as he himself has suggested, the non-existence of finiteness proofs for other algorithms need not preclude them from practical consideration and it appears reasonable to consider other selection criteria.

We have, in fact, applied four different rules for selecting the generating row to a sample set of problems and these are defined below.

Firstly, however, we should mention that for this particular algorithm, both in its standard form as presented above and in its extended form to be described in the next chapter, in the initial tableau we interchanged the unit rows and the constraint rows. This was done to simplify the situation regarding lexicographical positivity in our particular maximization problems.

With this in mind our variations on generator selection were as follows.

(i) Select the first row from the top of the tableau satisfying $a_{io} < 0, \quad i \neq 0$.

(ii) Select cyclically within the group of candidates. That is, on the first iteration select the first of the group satisfying $a_{io} < 0$; on the second iteration select the second of the group satisfying $a_{io} < 0$; and so on.

For example, if at the k-th iteration there are p rows with $a_{io} < 0$, $i \neq 0$, then select the k-th modulo p, such that $0 < k \leqslant p$, as generator.

(iii) Select a row which produces the lexicographically largest pivot column.

(iv) Select, from the group satisfying $a_{io} < 0$, $i \neq 0$, the row containing the least number of negative elements.

Before discussing our experience with these variants we make some brief remarks concerning them.

Only the first variant satisfies Gomory's condition for finiteness. The second is quite different from the cyclic selection defined by Gomory. Since an integral multiple of the pivot column is added to the constants column in a pivot operation then clearly the third variant is an attempt at changing the constants column by as much as possible. Such a selection is suggested by Gomory. The last choice above attempts to avoid, or at least limit, degeneracy, a condition which occurs frequently in integer programming problems.

2.3 Application to Problem Sample

We have applied our codes for this algorithm to 13 problems. A complete table of results is presented later but we give in Table 2.1 summary performance figures for the four variants defined above.

PROBLEM	VARI-ABLES	CONST-RAINTS	NUMBER OF ITERATIONS. VARIANT (i)	VARIANT (ii)	VARIANT (iii)	VARIANT (iv)
11	7	7	9	38	9	10
12	7	7	15	24	15	13
13	4	3	8	5	9	5
14	15	15	28	64	28	23
15	2	2	18	18	10	10
16	3	3	23	22	22	21
17	2	2	13	13	12	13
18	3	3	>2000	27	>2000	19
19	3	3	1560	1769	1560	1560
20	2	2	100	100	100	100
21	2	2	234	234	235	234
22	2	2	288	288	288	288
23	8	8	>2000	>2000	>2000	119

TABLE 2.1 Performance of four variants of the All Integer Integer Code.

One noteworthy feature is the difficulty certain variants have had in dealing with very small problems, in particular problems 18 & 19. Overall, variant (iv) appeared the most successful, being the only one to solve all problems. Moreover, on no problem did it display the worst performance and it had the best performance on 84% of the problems. There appears little to choose between variants (i) and (iii), the only significant difference in performance being concerned with problem 15. Variant (ii) behaved less well than

the others in a few problems but did, in fact, prove to be remarkably successful on problem 18, which was unsolved by variants (i) and (iii).

2.4 Pure Integer Method

This was the first cutting plane method developed by Gomory and has probably stimulated much of the research on integer programming undertaken since 1958. It is applicable to problems in which all the variables are required to be integral, but unlike the algorithm just described it is not necessary for the coefficients a_{ij} to be integral.

To start the description of the method we need only assume that the system is dual and primal continuous feasible, that is $a_{oj} \geqslant 0$, $j \in N$ and $a_{io} \geqslant 0$, $i \in I$. If the system is also primal feasible, that is, by our definition, $a_{io} \equiv 0$ modulo 1 as well as $a_{io} \geqslant 0$, $i \in I$, then the tableau is optimal and there is nothing further to consider. Suppose some infeasible row is defined by

$$x_i = a_{io} + \sum_{j \in N} a_{ij}(-t_j).$$

As in the previous section, any feasible solution to the linear programming problem (1.10), (1.11), (1.12) and (1.13) must satisfy the constraint represented by (2.7a), namely

$$s = -f_{io} - \sum_{j \in N} f_{ij}(-t_j) \qquad (2.14)$$

where $a_{ij} = \left[a_{ij}\right]^{+} f_{ij}$, $0 \leqslant f_{ij} < 1$ and s is a non-negative integer slack variable. Clearly, the continuous optimal solution of the original problem excluding the integer restriction (1.13)

does not satisfy the constraint represented by (2.14) since $t_j = 0$, $j \in N$ for this solution. Thus if we add such a constraint to the problem constraints, the new set of feasible solutions will be smaller than the set of solutions to the continuous problem but will still contain all feasible solutions to the integer problem. The constraint represented by (2.14) is the original cutting plane introduced by Gomory.

The objective function value x_0 is unrestricted in sign and so it does not necessarily satisfy (2.5) and (2.6). Consequently, the objective function is not used as a possible generator in the All Integer Integer Method. However, in the method now considered (when $\lambda = 1$, $f = 0$ and (2.5) contains no term in x_i), if we assume that the a_{oj} in the original problem are integer valued, so that the optimal x_0 is also required to be integral, then the objective function may also be considered as a cut generator.

The computational details of the algorithm are described in Chapter VII but briefly the procedure computes, by the simplex method, the optimal solution of the linear programming problem when the integrality restrictions are ignored. Then the problem tableau is augmented by a new equation (2.14) and the resulting infeasible system solved by the dual simplex method. This process of adding a constraint and solving a new linear programming problem is continued until a feasible solution is reached or until an infeasible problem is produced. Whenever a slack variable s re-enters the basis and assumes a non-negative value, the corresponding basic

solution is associated with a vertex which is inside the half-space defined by the appropriate s constraint and so this constraint may then be suppressed.

2.5 Variants of Generator Selection

As was the case with the previous method this method too is capable of variation, particularly in the process of selecting one row from which to derive a new cutting plane. We tried the following four variants.

From rows i, $i \in \{0\} + N$, select

(i) the row r such that

$$f_{rk} = \min_i f_{ik} \text{ (provided } a_{rj},\ j \in \{0\} + N, \text{ not all integral)}$$

where the column k is given by

$$a_{ok} = \min_{j \in N} a_{oj} \quad ,$$

(ii) the first row with $a_{io} \not\equiv 0$ modulo 1,

(iii) the row with the largest f_{io},

(iv) the row with the smallest $-\bar{f}_{io}$.

Table 2.2 illustrates two sets of performance figures which are typical.

		NUMBER OF ITERATIONS.		
PROBLEM	VARIANT (i)	VARIANT (ii)	VARIANT (iii)	VARIANT (iv)
1	> 104	> 104	26	84
2	> 104	> 104	16	95

TABLE 2.2 Performance of Pure Integer Code.

Results of this sort would appear to favour variant (iii). However, such a decision would not be satisfactory because the figures associated with the first two variants are not meaningful.

During the application of the dual simplex method the effect of dual degeneracy, a frequently encountered state in integer programming, is noticed when seeking a pivot column, for then a number of columns will satisfy the pivot column criterion. The figures in the above table were produced by a code which always selected the first of the eligible columns. This led to cycling, a situation in which the same sequence of bases is repeated indefinitely. Variant (i) applied to problem 1, for example, caused the solution process to commence cycling, with cycle length 8, at the first iteration in the integer phase. In an attempt to devise a process which would avoid cycling we defined three other rules for selecting a pivot column from among tying columns. These and the first rule used are now defined.

(i) Select the first of the tying columns.

(ii) If the number of tying columns is b at the k-th event of a tie select the k-th modulo b such that $0 < k \leqslant b$. That is, on the first event of a tie select the first, on the second event of a tie select the second and so on.

(iii) Select randomly from among the tying columns.

(iv) If one of the tying columns is k and t_k is a variable of the original system, select it; otherwise select at random.

2.6 Experience With Method

The performance figures for the combined row and column variants applied to the previous two problems are displayed in Table 2.3. The figures are representative of the complete solution set from which we chose our combination of generator and column selection rules. The

results indicated that the random column selection rule would probably be the most satisfactory and further, that row variants (i) and (iii) were probably the best with no preference for either. To our sample of 30 test problems we applied the

VARIANT		NUMBER OF ITERATIONS	
ROW	COLUMN	PROBLEM 1.	PROBLEM 2.
(i)	(i)	> 104	> 104
(i)	(ii)	74	25
(i)	(iii)	25	18
(i)	(iv)	69	64
(ii)	(i)	> 104	> 104
(ii)	(ii)	> 104	49
(ii)	(iii)	57	26
(ii)	(iv)	> 104	93
(iii)	(i)	26	16
(iii)	(ii)	39	14
(iii)	(iii)	18	14
(iii)	(iv)	36	17
(iv)	(i)	84	95
(iv)	(ii)	60	> 104
(iv)	(iii)	50	> 104
(iv)	(iv)	> 104	99

TABLE 2.3 Performance using combined variants.

row and column selection combinations ((i), (iii)) and ((iii), (iii)).

As we remarked above, dual degeneracy is a common phenomenon in integer programming and the majority of problems in our sample are no

exception. For this reason, it was considered worthwhile to experiment with yet another variation. In some cases when the objective function is chosen as the constraint from which to derive a cutting plane it may generate a plane parallel to itself, so that the succeeding tableau will clearly be dual degenerate. In an attempt to avoid dual degeneracy we excluded the objective function from the search for a generator. The entire set of results obtained from the experiments is presented and discussed in detail in Chapter VII but a summary of performance is given in Table 2.4.

PROBLEM	NUMBER OF ITERATIONS			
	OBJECTIVE INCLUDED		OBJECTIVE EXCLUDED	
	(i) + (iii)	(iii) + (iii)	(i) + (iii)	(iii) + (iii)
1	25	18	21	23
2	18	19	25	19
3	43	21	43	31
4	13	8	14	47
5	1431	855	73	455
6	80	559	116	753
7	> 2004	1862	643	1566
8	89	> 2004	514	877
9	558	1922	819	> 2006
10	8	7	14	10
11	9	9	14	10
12	25	21	30	35

TABLE 2.4 Performance of pure integer code using selected variants.

PROBLEM	NUMBER OF ITERATIONS			
	OBJECTIVE INCLUDED		OBJECTIVE EXCLUDED	
	(i) + (iii)	(iii) + (iii)	(i) + (iii)	(iii) + (iii)
13	17	30	21	93
14	41	251	50	100
15	37	> 2002	48	> 2002
16	30	59	36	266
17	13	7	25	17
18	15	9	27	19
19	55	> 2003	454	> 2003
20	200	200	1595	821
21	305	324	1597	762
22	298	308	1673	762
23	46	73	34	73
24	10	7	9	10
25	5	5	7	7
26	8	8	8	8
27	4	4	4	4
28	4	4	5	5
29	12	10	12	10
30	9	8	10	8

TABLE 2.4 (continued)

The only combination of variants which succeeded in solving all thirty problems was that which

(a) did not allow the objective function to be a generator,

(b) chose as generator the row r such that $f_{rk} = \min_i f_{ik}$, $i \in N$,

where the column k is given by $a_{ok} = \min_j a_{oj}$, $j \in N$,

and (c) chose randomly from among tying columns in the event of two or more columns qualifying as pivot column.

However, apart from the fact that it did not fail on any problem, this combination gave, in general, a performance inferior to that obtained by using the corresponding generator and column rules but including the functional row as a possible generator. Its behaviour was no better in 25 of the 30 problems. In 10 of these the behaviour was much worse, the iteration count for two of them being increased by as much as a factor of 8. A fairly similar behaviour is displayed by the pairs of columns representing the variant which selects the generator by the max f_{io} rule. This appears to indicate that by not allowing the objective function to generate a cutting constraint our aim of improving the performance by avoiding obvious cases of degeneracy was not attained. Finally, based on the evidence of Table 2.4 there appears little to choose between either of the generator selection variants used.

2.7 Mixed Integer Programming

When only a subset of the variables in the integer programming problem is required to be integral the problem is one in mixed integer programming. It is sometimes possible to transform a mixed integer problem into a pure integer problem. For example, if the constraints were initially given in the form of inequalities with rational coefficients and integer structural variables and if the a_{io} were non-integral then the slack variables introduced would also be non-integral.

However, if each inequality constraint is multiplied by the least common multiple of the denominators of the rational coefficients and the constant term before the addition of the slack variables, a system of constraints in integer variables and with integer coefficients is obtained. If necessary the objective function may be similarly transformed and the problem becomes one in pure integers.

Mixed integer programming is not a special case of pure integer programming but does use the same principle as that used in the pure integer method described in Section 2.4. As before, if the solution to a linear programming problem is infeasible over the integer subspace then a cut which excludes the infeasible solution but no feasible solution is added to the system. The difference between the solution processes for the pure and the mixed problems arises from the way in which the cuts are determined.

Again let the problem be defined by (1.10), (1.11), (1.12) and (1.13) but with K a proper subset of I. Suppose that the tableau is primal dual continuous feasible but integer infeasible, that is, $a_{oj} \geq 0$, $j \in N$, $a_{io} \geq 0$, $i \in I$ but $a_{io} \not\equiv 0$ modulo 1 for some $i \in K$. Consider

$$x_i = a_{io} + \sum_{j \in N} a_{ij} (-t_j) ,$$

which represents an appropriate tableau row

Representing the coefficient a_{io} by its integral and fractional parts separately we can write this as

$$x_i - \left[a_{io}\right] = f_{io} + \sum_{j \in N} a_{ij} (-t_j), \qquad (2.15)$$

and for any feasible solution to the original problem

$$f_{io} + \sum_{j \in N} a_{ij} (-t_j) \equiv 0 \text{ modulo } 1. \tag{2.16}$$

Let N^+ be the subset of N for which $a_{ij} \geq 0$ and N^- the subset of N for which $a_{ij} < 0$. Then

$$\sum_{j \in N^-} a_{ij} t_j + \sum_{j \in N^+} a_{ij} t_j - f_{io} \equiv 0 \text{ modulo } 1. \tag{2.17}$$

We must now consider the two cases in which $\sum_{j \in N} a_{ij} t_j$ is either non-negative or negative.

Considering non-negativity first, since $0 \leq f_{io} < 1$ then it follows from (2.16) that

$$\sum_{j \in N} a_{ij} t_j \geq f_{io}$$

or, by (2.17) and the definition of N and N^+, that

$$\sum_{j \in N^+} a_{ij} t_j \geq f_{io}. \tag{2.18}$$

Now consider the case when $\sum_{j \in N} a_{ij} t_j < 0$. From (2.16)

$$\sum_{j \in N} a_{ij} t_j \leq f_{io} - 1 ,$$

or, for any feasible solution

$$\sum_{j \in N^+} a_{ij} t_j + \sum_{j \in N^-} a_{ij} t_j \leq f_{io} - 1.$$

Since $\sum_{j \in N^+} a_{ij} t_j \geq 0$, it follows that

$$\sum_{j \in N^-} a_{ij} t_j \leq f_{io} - 1 , \tag{2.19}$$

and so multiplying throughout by $f_{io} / (f_{io} - 1)$,

$$\sum_{j \in N^-} \frac{f_{io}}{1 - f_{io}} \left| a_{ij} \right| t_j \geq f_{io}. \tag{2.20}$$

Combining (2.18) and (2.20), for each case, any feasible solution must satisfy

$$\sum_{j \in N^+} a_{ij} t_j + \sum_{j \in N^-} \frac{f_{io}}{1 - f_{io}} \left| a_{ij} \right| t_j \geq f_{io} \tag{2.21}$$

since the left hand sides of both (2.18) and (2.20) are positive

and at least one must be greater than or equal to f_{io}.

The relationship (2.21) could be used as a cut. Introducing a non-negative slack variable s we obtain

$$s = -f_{io} - \sum_{j \in N^+} a_{ij}(-t_j) - \sum_{j \in N^-} \frac{f_{io}}{1-f_{io}} |a_{ij}|(-t_j).$$

This equation is clearly not satisfied by the current solution since

$$t_j = 0,\ j \in N, \implies s = -f_{io} < 0.$$

However, such a cut should only be used when all the t_j, $j \in N$, are continuous variables, since if this is not the case, and so some t_k, $k \in N$, are integer variables, a stronger cut can be constructed.

So far we have only used information about the signs of the coefficients a_{ij}, $j \in N$, and the fact that the t_j are non-negative. Suppose now that one or more of the variables t_j must be integer variables and let t_j, $j \in R \subset N$, be such a set. Denote by N_c the subset of N not in R, and let R^+ be the subset of R for which $a_{ij} \geq 0$ and R^- the subset of R for which $a_{ij} < 0$. Similarly, let N_c^+ and N_c^- be equivalently defined subsets of N. Then the congruence (2.17) may be expressed as

$$\sum_{j \in R^+} a_{ij}\, t_j + \sum_{j \in N_c^+} a_{ij}\, t_j + \sum_{j \in R^-} a_{ij} t_j + \sum_{j \in N_c^-} a_{ij}\, t_j - f_{io} \equiv 0 \quad \text{modulo } 1$$

which is equivalent to

$$\sum_{j \in R_1^+} a_{ij}'\, t_j + \sum_{j \in N_c^+} a_{ij}\, t_j + \sum_{j \in R_1^-} a_{ij}'\, t_j + \sum_{j \in N_c^-} a_{ij}\, t_j - f_{io} \equiv 0 \quad \text{modulo } 1$$

where $a_{ij}' \equiv a_{ij}$ modulo 1 and the sign of a_{ij}' is arbitrary when a_{ij} is positive or negative.

As before any feasible solution to the original problem must satisfy

$$\sum_{j \in R_1^+} a_{ij}' \, t_j + \sum_{j \in N_c^+} a_{ij} \, t_j + \sum_{j \in R_1^-} \frac{f_{io}}{1-f_{io}} \left| a_{ij}' \right| t_j + \sum_{j \in N_c^-} \frac{f_{io}}{1-f_{io}} \left| a_{ij} \right| t_j \geq f_{io} \tag{2.22}$$

This cut can be made as strong as possible by making the coefficients of t_j, $j \in R_1^+$ and $j \in R_1^-$, as small as possible. Starting from any a_{ij}, positive or negative, we can produce an $a_{ij}' \geq 0$ ($a_{ij}' \geq f_{ij}$) or an $a_{ij}' < 0$ ($a_{ij}' \leq f_{ij} - 1$) which, in turn, produce coefficients a_{ij}' or $\frac{f_{io}}{1-f_{io}} \left| a_{ij}' \right|$ respectively. The smallest values of these are therefore

$$f_{ij} \quad \text{and} \quad \frac{f_{io}}{1-f_{io}} (1-f_{ij})$$

and the smaller of these is

$$\begin{aligned} & f_{ij} \quad \text{if} \quad f_{ij} \leq f_{io} \\ \text{and} \quad & \frac{f_{io}}{1-f_{io}} (1-f_{ij}) \quad \text{if} \quad f_{ij} \geq f_{io} \end{aligned}$$

We can now write the best cut in the general form

$$\sum_{j \in N} d_{ij} \, (-t_j) \leq -f_{io} \tag{2.23}$$

where

$$d_{ij} = \begin{cases} a_{ij}, & j \in N_c^+, \\ \frac{f_{io}}{1-f_{io}} \left| a_{ij} \right|, & j \in N_c^-, \\ f_{ij}, & j \in R \text{ and } f_{ij} \leq f_{io}, \\ \frac{f_{io}}{1-f_{io}} (1-f_{ij}), & j \in R \text{ and } f_{ij} > f_{io}. \end{cases}$$

Once the constraint (2.23) is added to the tableau the dual simplex method is applied as before.

In passing, we note a particular feature of the cuts introduced

in the last two methods. All the coefficients a_{sj}, $j \in N$, of the adjoined row

$$s = a_{so} + \sum_{j \in N} a_{sj} (-t_j)$$

of the tableau satisfy $a_{sj} \leqslant 0$. For example, from (2.14) we have

$$s = -f_{io} - \sum_{j \in N} f_{ij} (-t_j) \geqslant 0$$

and from (2.23) we obtain

$$s = -f_{io} - \sum_{j \in N} d_{ij} (-t_j) \geqslant 0$$

after the addition of a non-negative slack variable s. Suppose the dual simplex method selected column k as the pivot column, then the value of x_i ($x_i = a_{io} + \sum_{j \in N} a_{ij} (-t_j)$) would be changed by

$$- \frac{a_{ik}}{a_{sk}} (-f_{io}) ,$$

where $a_{sk} = -f_{ik}$ in the pure integer case,

and $a_{sk} = -d_{ik}$ in the mixed integer case.

From this we see that if $a_{ik} > 0$, the current value of x_i is decreased by an amount $\frac{a_{ik}}{a_{sk}} f_{io} \geqslant f_{io}$, so that the new value of the basic variable must satisfy $x_i \leqslant [a_{io}]$.

Although our code for this method caters for a number of variants of generator selection, experience has been limited to small test problems. In addition, it was used during the development of two pure integer programs, as discussed in Chapter VII.

2.8 Remarks on Finiteness Proofs

Rather than present Gomory's original proofs of finiteness for the methods of this chapter we shall discuss, briefly, some of their major features.

As mentioned earlier, Gomory's finiteness proofs are dependent upon both a particular rule for generator selection and lexicography. For each of the algorithms he proves that the solution will be reached in a finite number of steps if, from among the eligible generator rows, we select the first encountered. Although the lexicographic pivot selection rule is necessary to his proofs, he does state in [19] that it is not essential for finiteness.

For the Method of Integer Forms, described in Section 2.4, Gomory gives an alternative proof applicable to a different process in which dual pivot operations are interspersed with the addition of constraints in any way that ensures that the cumulative product of all pivot elements does not exceed a given upper bound at any stage.

The finiteness proof for the All Integer Integer Algorithm is dependent upon the elimination of the possibility that, from a certain iteration onwards, the p first components of the constants column vector remain fixed and the (p+1)st component decreases indefinitely. This is achieved by any rule which ensures that any row in which a_{io} becomes negative and remains negative will eventually be chosen as a generator row. One such rule is that already mentioned, another is that which selects cyclically.

As experience has indicated, there appears to be no way of assessing in advance the rate of progress we may expect from the application of any given variant of any of the algorithms. In some cases, a number of variants which have not been proved to give finite algorithms, have in fact produced solutions more rapidly than proven variants. To be able to make some sort of prior assessment of

behaviour of particular variants applied to given problems would clearly be of immense value. However, because of the erratic performance of any given variant on different problems no such technique yet exists.

Gomory's mixed integer algorithm can be shown to be finite provided that a particular generator selection rule is used and that the objective function to be maximized is a function of the integer variables only. The latter condition cannot always be met in practice, yet it has been found that the algorithm has frequently been finite when applied to problems violating this condition.

It has been seen that, although finiteness proofs do not exist for them, many algorithms have, in practice, been found to be finite. As is often the case though, no single algorithm has displayed consistent behaviour so finiteness proofs for such algorithms would serve no worthwhile purpose. However, if a consistent algorithm appeared to be available proof of its finiteness in general application would be of both theoretical and practical interest.

CHAPTER III

ON EXTENSIONS TO CUTTING PLANE METHODS.

This chapter is wholly concerned with extensions to two of the cutting plane methods described in the previous chapter. In particular, stronger cutting planes are derived for both methods of solution of pure integer programming problems. The basic principles are exactly as before but the cutting plane is modified in such a way that it now excludes an amount of infeasible space not less than that excluded by the cutting planes as defined by Gomory. In Sections 3.1 and 3.2 we deal with the All Integer Integer Algorithm, presenting first the derivation of the new cut and second its performance when applied to a small set of problems. Improved cutting planes for the pure integer method are derived in Section 3.3. When applied in practice these require an intermediate computation involving the solution of a diophantine equation; this is discussed in Section 3.4. In Section 3.6 we discuss our experience with this particular modification when applied to the complete sample of 30 test problems. Finally, we discuss a further modification of the same technique and illustrate this by an example.

3.1 Modified Cutting Plane for the All Integer Integer Algorithm

The cutting plane for the All Integer Integer Algorithm is obtained from a tableau row representing

$$x_i = a_{io} + \sum_{j \in N} a_{ij} (-t_j) ,$$

in which $a_{io} < 0$, and is defined by

$$s = \left[\frac{a_{io}}{\lambda} \right] + \sum_{j \in N} \left[\frac{a_{ij}}{\lambda} \right] (-t_j) , \qquad (3.1)$$

where $\lambda = \max_{j \in R} \left(- \frac{a_{ij}}{h_j} \right)$, $R = \left\{ j \,\middle|\, a_{ij} < 0 \right\}$,

h_j being the largest integer satisfying $(1/h_j) P_j \geqslant P_k$, in which k is the index of the lexicographically smallest column belonging to the set of columns R.

In the second of the improved cut methods, to be described later in this chapter, we improve the Gomory cut by attempting to "push it" parallel to itself and further into the convex region currently defined. The cutting plane (3.1) cannot be improved by a similar method. However, the possibility of improving the cut does exist. As was noted in Section 2.1, it has been pointed out by Gomory that the selection of λ as defined above does not necessarily produce the optimal cut at each stage; choosing a larger $\lambda (> 1)$ while still satisfying the condition that a pivot element of -1 is obtained, may produce better cuts. Of course the pivot column is unaffected by the choice of λ, so suppose λ_1 and λ_2 both provide cuts with a pivot element of -1. Pivoting on the indicated column $P_k = (a_{ok}, a_{ik}, \ldots\ldots)$ will add to the current constants column either $\left[\frac{a_{io}}{\lambda_1} \right] P_k$ or $\left[\frac{a_{io}}{\lambda_2} \right] P_k$ and the smaller λ produces the greatest change in the constants column P_o. As mentioned, this

does not necessarily introduce a stronger inequality than the original. For example, suppose we had $x = -4 - 3(-x_1) - 5(-x_2)$ then $\lambda = 2$ will produce the equation $s = -2 - 2(-x_1) - 3(-x_2)$, $\lambda = 3$ will produce $s = -2 - 1(-x_1) - 2(-x_2)$ and $\lambda = 4$ will produce $s = -1 - 1(-x_1) - 2(-x_2)$. Rewriting these as inequalities we have, since $s \geq 0$,

$$\begin{aligned} 3x_1 + 5x_2 &\geq 4, \\ 2x_1 + 3x_2 &\geq 2, \\ x_1 + 2x_2 &\geq 2, \\ x_1 + 2x_2 &\geq 1 \end{aligned}$$

where the second is weaker than the first, the third stronger and the last weaker again.

A re-evaluation of λ, as described below, enables us to produce a further constraint, always at least as strong as Gomory's constraint

$$s = \left[\frac{a_{io}}{\lambda_{min}}\right] + \sum_{j \in N} \left[\frac{a_{ij}}{\lambda_{min}}\right](-t_j) ,$$

where λ_{min} is the value of λ as determined by (3.1). In any strengthening of this by increasing the value of λ above λ_{min}, preservation of the pivot -1 is the first essential and is achieved simply by choosing any $\lambda \geq \lambda_{min}$. In order to improve the Gomory constraint we can seek a $\lambda > \lambda_{min}$ such that

$$\left[\frac{a_{io}}{\lambda}\right] = \left[\frac{a_{io}}{\lambda_{min}}\right], \text{ and } \left[\frac{a_{ij}}{\lambda}\right] = \left[\frac{a_{ij}}{\lambda_{min}}\right] , \; a_{ij} > 0, \text{ so that } \left[\frac{a_{ij}}{\lambda}\right] = \left[\frac{a_{ij}}{\lambda_{min}}\right] ,$$

$a_{ij} < 0$, and the cut is obviously at least as deep as before, if not deeper.

The first of these conditions is satisfied by any new λ in the range

$$-\left[\frac{a_{io}}{\lambda_{min}}\right] \geqslant -\frac{a_{io}}{\lambda} > -\left[\frac{a_{io}}{\lambda_{min}}\right] - 1.$$

That is

$$\frac{a_{io}}{\left[\frac{a_{io}}{\lambda_{min}}\right]+1} > \lambda \geqslant \frac{a_{io}}{\left[\frac{a_{io}}{\lambda_{min}}\right]} = \lambda_{min} - \alpha, \alpha \geq 0, \qquad (3.2)$$

unless $-a_{io} \leqslant \lambda_{min}$ in which case the only restriction on λ is $\lambda \geqslant -a_{io}$ which is automatically satisfied because $\lambda \geq \lambda_{min} \geqslant -a_{io}$.

The second of the conditions, namely $\left[\frac{a_{ij}}{\lambda}\right] = \left[\frac{a_{ij}}{\lambda_{min}}\right]$, $a_{ij} > 0$, is satisfied by any new λ in the range

$$\min_{a_{ij}>0}\left(\frac{a_{ij}}{\left[\frac{a_{ij}}{\lambda_{min}}\right]}\right) \geqslant \lambda > \max_{a_{ij}>0}\left(\frac{a_{ij}}{\left[\frac{a_{ij}}{\lambda_{min}}\right]+1}\right) = \lambda_{min} - \beta, \beta \geq 0. \qquad (3.3)$$

Clearly then if λ_{min} is to be improved we must use the upper bounds provided by (3.2) and (3.3) and choose

$$\bar{\lambda} = \min\left\{\frac{a_{io}}{\left[\frac{a_{io}}{\lambda_{min}}\right]+1} - \delta, \min_{a_{ij}>0}\left(\frac{a_{ij}}{\left[\frac{a_{ij}}{\lambda_{min}}\right]}\right)\right\}, \delta > 0.$$

Hence a possibly stronger cut will be produced if $\bar{\lambda} > \lambda_{min}$.

We note here that in deriving (3.2) it was pointed out that when $|a_{io}| \leqslant \lambda_{min}$ no upper bound is defined for λ. However, for any situation with this and the further property that $a_{ij} < 0$ for all $j \in N$ (2.9) and (2.10) are satisfied by any $\lambda \geqslant \lambda_{min}$. The best

cut is provided in our computations by choosing

$$\lambda = \max\ (\lambda_{\min}, -a_{ij})$$

As an illustration of an improved cut consider a tableau in which the a_{oj}, $j \in N$, are 1, 5, 8 and 2 and an eligible generator row is

$$x_i = -26 - 9(-t_1) - 19(-t_2) - 29(-t_3) + 12(-t_4).$$

Then by the All Integer Integer Algorithm we have

$$\lambda_{\min} = \max\ \left(\frac{9}{[1]}, \frac{19}{[5/1]}, \frac{29}{[8/1]}\right) = 9$$

and the added constraint is

$$s = \left[\frac{-26}{9}\right] + \left[\frac{-9}{9}\right](-t_1) + \left[\frac{-19}{9}\right](-t_2) + \left[\frac{-29}{9}\right](-t_3) + \left[\frac{12}{9}\right](-t_4)$$

$$= -3 - 1(-t_1) - 3(-t_2) - 4(-t_3) + 1\ (-t_4).$$

This produces the inequality

$$t_1 + 3t_2 + 4t_3 - t_4 \geq 3.$$

Now, by the above

$$\bar{\lambda} = \min\left\{\frac{-26}{[-26/9]+1} - \delta\ ,\ \frac{12}{[12/9]}\right\} = \min\ (13 - \delta\ ,\ 12) = 12$$

which produces the inequality

$$\left[\frac{-9}{12}\right](-t_1) + \left[\frac{-19}{12}\right](-t_2) + \left[\frac{-29}{12}\right](-t_3) + \left[\frac{12}{12}\right](-t_4) \geq -\left[\frac{-26}{12}\right],$$

i.e. $\quad t_1 + 2t_2 + 3t_3 - t_4 \geq 3,$

clearly a stronger inequality than the previous one.

3.2 Performance

Once one stronger cut has been obtained the pattern of convergence will be quite different from that exhibited by the standard

PROBLEM	NUMBER OF ITERATIONS							
	STANDARD CUT				MODIFIED CUT			
	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)
11	9	38	9	10	9	38	9	10
12	15	24	15	13	15	24	15	13
13	8	5	9	5	7	7	8	7
14	28	64	28	23	28	64	28	23
15	18	18	10	10	18	18	10	10
16	23	22	22	21	23	22	22	21
17	13	13	12	13	13	13	12	13
18	>2000	27	>2000	19	>2000	27	>2000	19
19	1560	1769	1560	1560	1560	1761	1560	1560
20	100	100	100	100	100	100	100	100
21	234	234	235	234	234	234	235	234
22	288	288	288	288	288	288	288	288
23	>2000	>2000	>2000	119	>2000	>2000	>2000	120
SECONDS	18.54	24.27	32.70	7.10	19.70	28.21	44.30	

The last row gives the appropriate execution times in seconds. Columns (a), (b), (c) and (d) represent variant 0 (select first row with $a_{io} < 0$); variant 1 (select cyclically); variant 2 (select the row producing the lexicographically largest pivot column) and variant 3 (select the row containing the least number of negative elements).

TABLE 3.1 Performance of standard and modified cuts using four variants of the All Integer Integer code.

cut solution. Hence, although stronger cuts may be produced they do not necessarily accelerate the convergence of the solution process.

We applied the extended method to the same set of 13 problems to which the standard method had already been applied. The performance figures of both codes, using four generator variants, are given in Table 3.1. The figures are perhaps disappointing in that only two problems were solved in fewer iterations by the extended method. Problem 13 displays improvement for variants 0 and 2 but behaved worse for variants 1 and 3. Apart from this, on the evidence of this small data sample, the effort involved in seeking an improved λ does not seem worthwhile since any advantage gained by decreasing the iteration count is outweighed by the extra computation. This leads to an increase in execution time. For example, variant 0 applied to the standard and modified cutting methods required overall execution times of 18.54 seconds and 19.70 seconds respectively, yet apart from an improvement by just one iteration in problem 13 the performances were identical in terms of the number of iterations. Similarly, variant 1 required times of 24.27 seconds and 28.21 seconds respectively.

3.3 Optimal Cut for the Pure Integer Method

Again, the basic principles are exactly as described in Section 2.4 but instead of using the cutting planes as defined by Gomory we attempt to define improved cuts by displacing the Gomory cutting planes in such a way that more of the current convex region is

excluded and thus a quicker approximation to the convex hull of feasible space in the neighbourhood of the optimum is obtained.

The possibility of the existence of an extension similar to that described in this section is really exposed during the derivation of a cutting plane in Gomory's paper on the Method of Integer Forms [15]. To show this we extract from Section 2.1 the congruence (2.4) and set $\lambda = 1$ to give

$$\sum_{j \in \bar{N}} f_{ij}\, t_j \equiv f_{io} \quad \text{modulo } 1. \tag{3.4}$$

The argument in [15] proceeds as follows. The left hand side of (3.4) is non-negative and differs from the non-negative right hand side by an integer, so it must be either f_{io}, $1 + f_{io}$, $2 + f_{io}$ etc. Consequently,

$$\sum_{j \in \bar{N}} f_{ij}\, t_j \geqslant f_{io}, \tag{3.5}$$

which is the cut derived in Section 2.4

Clearly then, the cut (3.5) is not necessarily the strongest since it may have been possible to make the constant term equal to $n + f_{io}$ with $n \geqslant 1$.

Our code for the method to be described in this section is based on a technique which carries out the computation entirely in fixed point arithmetic, in which the numerators and denominators of all fractions are stored separately. This method does have a disadvantage due to these denominators possibly becoming excessively large during the application of the simplex or dual simplex methods. This situation is discussed in Chapter VII. We now use this fractional representation to derive the improved cutting plane.

Assume that an element a_{ij}, $i = 0,\ldots,n+m$, $j = 0,\ldots,n$ is written as a'_{ij}/D, where D is the least common multiple of the denominators of all the elements. The term D does, in fact, have a far greater significance, as is shown in [15], but this need not concern us here. Suppose an integer programming problem is at some intermediate stage of its solution. In the tableau, whose elements are assumed to be in the a'_{ij}/D form, consider any equation defining an integer variable x_i in terms of nonbasic variables t_j, say

$$x_i = \frac{1}{D}\left\{a'_{io} + \sum_{j \in N} a'_{ij}(-t_j)\right\} \tag{3.6}$$

where $a_{io} = \frac{a'_{io}}{D}$ is not integral. Since $a'_{io} \not\equiv 0$ modulo D the equation (3.6) may be written as

$$x_i = \left[\frac{a'_{io}}{D}\right] + \frac{r_{io}}{D} + \sum_{j \in N}\left[\frac{a'_{ij}}{D}\right](-t_j) + \sum_{j \in N}\frac{r_{ij}}{D}(-t_j) \tag{3.7}$$

where r_{ij}, $j \in \{0\} + N$, is the least residue of the congruence $a_{ij} \equiv r_{ij}$ modulo D, so that $f_{ij} = r_{ij}/D$ by a previous definition of f_{ij}. Now multiply (3.7) by D to obtain

$$Dx_i = D\left[\frac{a'_{io}}{D}\right] + D\sum_{j \in N}\left[\frac{a'_{ij}}{D}\right](-t_j) + r_{io} + \sum_{j \in N} r_{ij}(-t_j). \tag{3.8}$$

Using similar arguments to those in Chapter II we see from (3.8) that $r_{io} + \sum_{j \in N} r_{ij}(-t_j)$ must be integral and furthermore, must be divisible by D. Hence

$$\sum_{j \in N} r_{ij} t_j \equiv r_{io} \text{ modulo } D. \tag{3.9}$$

Now $1 > f_{io}$, so $D > r_{io}$ or $r_{io} - D < 0$. Thus, since $r_{ij} \geq 0$ and $t_j \geq 0$ the congruence (3.9) implies that

$$\sum_{j \in N} r_{ij}\, t_j = r_{io} + kD, \qquad k = 0, 1, \ldots\ldots \tag{3.10}$$

It may therefore be possible to find a particular k, = k^{*} say, such that

$$\sum_{j \in N} r_{ij}\, t_j \geqslant r_{io} + kD \tag{3.11}$$

is violated by the continuous solution to the currently solved linear programming problem but is satisfied by every feasible solution the original problem when $k = k^{*}$ but not when $k \geqslant k^{*} + 1$. When k = 0 the equation (3.10) is, of course, equivalent to the standard cutting plane (2.14) of Gomory. The hyperplanes (3.10) must be satisfied by integer valued t_j but by choosing k = 0 we are in no way restricting these values, and the plane so defined may contain only lattice points lying entirely outside the region $t_j \geqslant 0$, $j \in N$. As shown above, such a plane can be pushed further into the convex region by increasing the value of the constant term.

The problem now becomes one of determining the maximum displacement which may be applied to the Gomory cutting plane, subject to the condition that no feasible lattice point is contained strictly within the space bounded by the original and displaced positions of the plane. Explicitly, the problem is one of finding the smallest non-negative integral k, such that

$$\sum_{j \in N} r_{ij}\, t_j = r_{io} + kD.$$

is satisfied by non-negative integer values of t_j.

Putting $k = t_{n+1}$ this can be expressed as an integer programming problem

$$\text{minimize } t_{n+1}$$

$$\text{subject to } \sum_{j \in N} r_{ij}\, t_j - D t_{n+1} = r_{io}$$

$$\text{and } t_j,\ j \in N + \{n+1\} \text{ non-negative integers.}$$

However, hand computation showed up the unsatisfactory nature of this approach for we are attempting to solve one integer programming problem within another and this process could be repeated yet again. In fact, we could be embarking on an infinite recursive process.

Instead, we adopted the fairly obvious and elementary method of choosing specific values for k, starting with k = 0 and increasing k in discrete unit steps, and attempting to solve the resulting diophantine equation in non-negative variables. The situation is simplified by one important fact, namely, that all the coefficients in the diophantine equation are non-negative since $r_{ij} \geq 0$. Formally then, the problem is to find non-negative integer values for t_j satisfying

$$\sum_{j \in N} r_{ij}\, t_j = r_{io} + kD \qquad (3.12)$$

where $r_{ij} \geq 0$, for a fixed integer value of $k \geq 0$.

3.4 Solution of the Diophantine Equation

In equation (3.12) the r_{ij} may be in any order and some may be zero, so for convenience we represent (3.12) by the congruence

$$\sum_{j=1}^{r} \alpha_j x_j \equiv \alpha_o \text{ modulo } D \tag{3.13}$$

where $\{1,\ldots,r\} = R \subset N$; α_j, $j \in \{0\} + R$, are positive integers and x_j are non-negative integer variables whose values are required. Furthermore, assume that $\alpha_j \leqslant \alpha_{j+1}$, $j = 1,\ldots,$ r-1. Now for some fixed value of k let the diophantine equation

$$\sum_{j \in R} \alpha_j x_j = \alpha_o + kD = \phi_r \tag{3.14}$$

define the problem P_r. Then set $x_r = \left[\frac{\phi_r}{x_r}\right]$ so that the remaining x_j must satisfy

$$\sum_{j=1}^{r-1} \alpha_j x_j = \phi_r - \alpha_r \left[\frac{\phi_r}{x_r}\right] = \phi_{r-1},$$

defining the problem P_{r-1}. Repetition of the process leads to

$$\alpha_1 x_1 = \phi_2 - \alpha_2 \left[\frac{\phi_2}{\alpha_2}\right] = \phi_1$$

defining the problem P_1, and we define

$$\phi_1 - \alpha_1 \left[\frac{\phi_1}{\alpha_1}\right] = \phi_o .$$

If at any stage ϕ_p is zero the congruence (3.13) is satisfied by the x_j already set for $j \geqslant p + 1$ and $x_i = 0$ for $j \leqslant p$. If the entire sequence P_j, $j = r, r-1,\ldots,1$ is produced and $\phi_o \neq 0$, then a new subsequence must be defined, starting at the last j for which $x_j > 0$. Suppose this was $j = q$ then reset $x_q = x_q - 1$ and $\phi_{q-1} = \phi_{q-1} + \alpha_q$ and proceed with the new subproblems P_j, $j = q-1, q-2,\ldots, 1$. Whenever $\phi_j = 0$ the corresponding r-vector is a lattice point on the hyperplane defined by (3.14). The recursive process can be continued until the search over the finite

hyperplane is complete i.e. the components x_r, x_{r-1},.....,x_2 are reduced to zero.

3.5 Illustration of Method

For the purpose of illustration, we chose one of the problems constructed by Thompson [34] and alleged to be difficult for the cutting plane method. The problem is to minimize x_3 subject to

$$5x_1 + 8x_2 - 7x_3 \geqslant -89,$$
$$-6x_1 + 5x_2 + x_3 \geqslant 11,$$
$$3x_1 - 5x_2 + 2x_3 \geqslant 29$$

with x_i non-negative integers.

In the following tableaux the numerators of the appropriate elements are displayed within the tableaux and the denominators D outside. The continuous optimal tableau is

	1	$-x_4$	$-x_5$	$-x_6$
x_0	-1321	15	49	73
x_1	121	-15	-19	-43
$\longrightarrow x_2$	79	-15	-31	-37
x_3	1321	-15	-49	-73

D = 90.

When seeking a row from which to generate a cutting plane we choose from among the basic variable rows that one with largest f_{io}. The selected generators are indicated by arrows and the pivots by asterisks.

In the above tableau we choose the x_2 row as generator and amend the tableau by the constraint row

$$s_1 = -(\tfrac{79}{90} + K) - \tfrac{75}{90}(-x_4) - \tfrac{59}{90}(-x_5) - \tfrac{53}{90}(-x_6).$$

Next, we set K = 0 and attempt to solve

$$75x_4 + 59x_5 + 53x_6 = 79$$

in non-negative integers x_4, x_5 and x_6. Clearly this has no solution. We proceed by setting K = 1, 2, 3 and attempting to solve the corresponding diophantine equations in turn. There are no solutions, but with K = 4 we have

$$75x_4 + 59x_5 + 53x_6 = 79 + 4.90 = 439,$$

which is satisfied by $x_4 = 2$, $x_5 = 4$, $x_6 = 1$ so producing the tableau

	1	$-x_4$	$-x_5$	$-x_6$	
x_0	-1321	15	49	73	
x_1	121	-15	-19	-43	
x_2	79	-15	-31	-37	D = 90.
x_3	1321	-15	-49	-73	
s_1	-439	-75*	-59	-53	

Performing the indicated pivot operation we produce

		1	$-s_1$	$-x_5$	$-x_6$	
	x_0	-1174	15	31	52	
	x_1	174	-15	-6	-27	
⟶	x_2	139	-15	-16	-22	D = 75.
	x_3	1174	-15	-31	-52	
	x_4	439	-90	59	53	

This tableau is infeasible and by the generator selection rule

the x_2 row is chosen for producing the new constraint

$$s_2 = -(\tfrac{64}{75} + K) - \tfrac{60}{75}(-s_1) - \tfrac{59}{75}(-x_5) - \tfrac{53}{75}(-x_6).$$

The first diophantine equation solvable in non-negative integers is derived from this when $K = 3$. The pivot is given by the s_1 coefficient and the pivot operation produces the following tableau from which we proceed with only brief comments.

	1	$-s_2$	$-x_5$	$-x_6$	
x_0	-997	15	13	31	
x_1	197	-15	7	-11	
→ x_2	169	-15	-1	-7	D = 60.
x_3	997	-15	-13	-31	
x_4	698	-90	118	106	
s_3	-(49+60K)	-45	-59*	-53	K = 4, i.e. $a'_{so} = -289$.

	1	$-s_2$	$-s_3$	$-x_6$	
x_0	-1043	5	13	19	
x_1	160	-20	7	-17	
→ x_2	171	-14	-1	-6	
x_3	1043	-5	-13	-19	D = 59.
x_4	118	-177	118	0	
x_5	289	45	-60	53	
s_4	-(53+59K)	-45*	-58	-53	K = 0.

		1	$-s_4$	$-s_3$	$-x_6$	
	x_0	-800	5	5	10	
	x_1	140	-20	25	5	
	x_2	143	-14	13	8	
→	x_3	800	-5	-5	-10	D = 45.
	x_4	249	-177	264	159	
	x_5	180	45	-90	0	
	s_5	-(35+45K)	-40*	-40	-35	K = 0.

		1	$-s_5$	$-s_3$	$-x_6$	
	x_0	-715	5	0	5	
	x_1	140	-20	40	20	
	x_2	138	-14	24	18	
→	x_3	715	-5	0	-5	D = 40.
	x_4	359	-177	392	279	
	x_5	125	45	-120	-35	
	s_6	-(35+40K)	-35	0	-35*	K = 0.

	1	$-s_5$	$-s_3$	$-s_6$	
x_0	-630	0	0	5	
x_1	105	-35	1	20	
x_2	105	-28	21	18	
x_3	630	0	0	-5	D = 35.
x_4	70	399	343	279	
x_5	140	70	-105	-35	
x_6	35	35	0	-40	

This tableau is optimal and feasible.

3.6 Optimal Cut in Practice

The method is clearly susceptible to precisely the same variants as those used in an earlier investigation of the Gomory cutting plane. In order to obtain a direct comparison between the appropriate cutting plane methods we chose to use those variants which were applied to the entire problem sample by the standard method. The complete sets of results are displayed in Tables 7 and 8 at the end of Chapter VII but summary performance figures together with those of the standard cut method are given in Table 3.2.

Although, on occasions, this method can produce an optimal cut which is far superior to the standard cut, this does not necessarily imply an accelerated convergence to the optimal solution. It is a temptation to draw an incorrect conclusion from the inequality constraint (3.11). One may immediately assume that since the cut that would be produced from this is at least as good as the Gomory cut then the continuous solution space must contract towards the convex hull of feasible solutions at least as quickly as it would have done in the application of Gomory cuts alone. However, this is not necessarily the case. If at some stage of the solution process the Gomory cut can be improved, then from that stage onwards the continuous convex solution space behaves quite differently from the way it would otherwise have behaved. This fact is demonstrated by some of the figures in Table 3.2. The code for the optimal cut method

Problem	Dimension: constraint	Dimension: variables	TOTAL NUMBER OF ITERATIONS. GENERATOR SELECTION 1. INCLUDING OBJECTIVE: Gomory	INCLUDING OBJECTIVE: Optimal	EXCLUDING OBJECTIVE: Gomory	EXCLUDING OBJECTIVE: Optimal	GENERATOR SELECTION 2. INCLUDING OBJECTIVE: Gomory	INCLUDING OBJECTIVE: Optimal	EXCLUDING OBJECTIVE: Gomory	EXCLUDING OBJECTIVE: Optimal
1	4	5	18	12	23	14	25	8	21	14
2	4	5	19	19	19	28	18	18	25	39
3	4	5	21	13	31	15	43	19	43	25
4	4	5	8	8	47	13	13	7	14	14
5	6	5	855	841	455	399	1431	397	73	68
6	6	5	559	559	753	33	80	80	116	82
7	4	5	1862	1545	1566	228	>2004	557	643	67
8	4	5	>2004	862	877	64	89	81	514	84
9	6	6	1922	729	>2006	46	558	469	819	53
10	4	5	7	7	10	10	8	8	14	14
11	7	7	9	9	10	10	9	9	14	14
12	7	7	21	21	35	24	25	25	30	27
13	3	4	30	29	93	16	17	19	21	23
14	15	15	251	251	100	100	41	29	50	83
15	2	2	>2002	4	> 2002	4	37	35	48	47
16	3	3	59	14	266	14	30	7	36	17
17	2	2	7	5	17	5	13	4	25	4
18	3	3	9	28	19		15	6	27	16
19	3	3	>2003	26	>2003	15	55	33	454	22
20	2	2	200	200	821	40	200	200	1595	55
21	2	2	324	6	762	103	305	89	1597	3

TABLE 3.2 (continued on following page)

Problem	Dimension		TOTAL NUMBER OF ITERATIONS.							
			GENERATOR SELECTION 1.				GENERATOR SELECTION 2.			
	constraint	variables	INCLUDING OBJECTIVE		EXCLUDING OBJECTIVE		INCLUDING OBJECTIVE		EXCLUDING OBJECTIVE	
			Gomory	Optimal	Gomory	Optimal	Gomory	Optimal	Gomory	Optimal
22	2	2	308	49	762	103	298	86	1673	103
23	8	8	73	67	73	67	46	48	34	33
24	2	2	7	6	10	4	10	6	9	4
25	3	2	5	5	7	5	5	5	7	5
26	5	2	8	8	8	8	8	8	8	8
27	3	3	4	4	4	4	4	4	4	4
28	2	5	4	4	5	5	4	4	5	4
29	5	2	10	10	10	10	12	12	12	12
30	3	3	8	4	8	4	9	5	10	5

Both codes use the random selection rule for breaking ties between columns.

GENERATOR SELECTION 1: Selects as new constraint generator that row with max. f_{io}

GENERATOR SELECTION 2: Selects that row with min f_{ik} where column k contains the smallest entry in the top row.

TABLE 3.2 Comparison of performances of the pure integer code and the optimal cut code.

gave inferior performances in 6 of the 120 cases. These occurred in two cases of problem 2, two cases of problem 13, one case of problem 18 and one case of problem 23. As with all the cutting plane codes,

the performance of the optimal cut code was unpredictable. It appeared to experience difficulties with a few problems which also presented difficulties to the standard method. This will be discussed in the next section. On the other hand, the improvement in performance on many problems was quite remarkable. Problems 15 and 19 which were unsolved by one variant of the standard cut code were solved in 4 and 26 iterations respectively by the optimal cut code. Apart from the six cases mentioned above, the optimal cut code did perform much better than the code using a standard Gomory cut. The use of this code does nevertheless give rise to certain difficulties which are discussed in the following section.

3.7 Difficulties and Possible Variations

As was mentioned in Chapter II, when discussing the pure integer code, the technique of representing fractions by storing their numerators and denominators separately may rapidly lead to very large numbers, exceeding the available computer capacity for fixed point arithmetic. This situation did not arise in the application of that code to the 30 problems. The progress of the optimal cut code was however impaired by this technique. One point where this sort of trouble can arise is during the derivation of the optimal cut. If, at such a stage, the D number is large, then increasing the constant term of the cut by multiples of D can soon cause the program to get out of control. So it is necessary to attempt to control D. Gomory [16] has suggested a method which ensures that the sequence of D numbers is monotonically decreasing. This may

be advantageous here because, in some of the problems solved, fairly rapid convergence occurs over discrete sections and associated with each of these is a decreasing sequence of D numbers. As D steadily decreases the value of the objective function also decreases until a state is reached where D is small relative to its value at the start of the sequence. The next pivot operation produces a tableau with $a_{io} < 0$ for at least one i and the restoration of $a_{io} \geq 0$, $i \in I$, may take some time. There are other ways of attempting to control D. We could, for example, set an upper bound on D so that whenever this limit is about to be exceeded we take some course of action to prevent this, such as limiting the search for the optimal cut or, perhaps even, choose a different generator. This limit on D could of course be fixed and supplied to the code as an input parameter or computed as required according to some heuristic.

The investigation into the determination of our optimal cutting plane for Gomory's pure integer method did lead to a most interesting feature of the diophantine equation solution stage. In working through some hand examples, the determination of the integer k, such that the associated diophantine equation was solvable in non-negative integers, proved to be laborious. However, we took advantage of our knowledge of the final solutions and applied the appropriate solution values to the non-basic variables in the cutting constraint. This led to the

correct determination of k. (The optimal cut method would continue from this stage by pivoting and producing yet more optimal cutting planes). This result then seemed to imply that once the diophantine equation was solved the original integer programming problem was also solved.

This investigation was applied to the cutting plane produced by any generator row but our observations led us to the fairly obvious idea of always using the functional hyperplane itself to generate a cut. That is, let us move the functional hyperplane parallel to itself, and as far as possible from its current position, into the convex region currently defined by the problem. The solution of the diophantine equation which produces this cut may, as was indicated in the preceding paragraph, provide the solution to the integer programming problem. On the other hand, the lattice point defined by this solution may be infeasible, in which case it will be necessary to search for further lattice points satisfying the diophantine equation in an attempt at finding a feasible lattice point. If none exists then the functional hyperplane may be moved further into the convex, by increasing the value of k, and the process repeated. If the lattice point established on the current cut does not happen to be feasible, the search for other lattice points which may be contained in the functional hyperplane in its current position can be laborious but, except in certain circumstances, it is finite

since the feasible region of the functional hyperplane is bounded $(0 \leqslant x_i \leqslant \left[\frac{r_{io} + kD}{a_{oi}}\right])$.

This method is a straightforward extension of the optimal cut method but its formalization should clarify certain points.

Suppose we have the tableau corresponding to the continuous optimal solution. Suppose further that this solution is infeasible, i.e. some $a'_{io} \not\equiv 0$ modulo D, $i \in I$. Then let the objective row be represented by

$$x_o = a'_{oo}/D + \sum_{j \in N} a'_{oj} (-t_j) /D \qquad (3.15)$$

where the a'_{oj}, $j \in \{0\} + N$, are the numerators of the correct coefficients with common denominator D. For the moment we assume that $a'_{oj} \not\equiv 0$, all j, i.e. that the current solution tableau is not dual degenerate. Now define the hyperplane

$$s_o = -r_{oo} - \sum_{j \in N} a'_{oj} (-t_j) - kD, \qquad (3.16)$$

which is a hyperplane parallel to the functional. When $k = 0$ and $a'_{oj} < D$, (3.16) defines the Gomory cutting plane that would be produced from the current objective row. When $k = 1, 2, \ldots,$ (3.16) defines successive positions of the hyperplane such that the space between two adjacent positions is empty of feasible lattice points until for some $k = k^*$ (3.16) is a hyperplane containing a feasible lattice point. Note that $k = k^*$ provides the optimum. Clearly, the aim is to minimize k subject to

$$\sum_{j \in N} a'_{oj} t_j = r_{oo} + kD \qquad (3.17)$$

$$t_j \geqslant 0, \quad j \in N, \qquad (3.18)$$

$$t_j \equiv 0 \text{ modulo } 1 , \tag{3.19}$$

$$x = a'_{io} + \sum_{j \in N} a'_{ij}(-t_j) \equiv 0 \text{ modulo } D, \quad i \in B, \tag{3.20}$$

where B is the set of indices of the current basic variables, and

$$x_i \geqslant 0. \tag{3.21}$$

The minimization of k subject to (3.17), (3.18) and (3.19) has already been dealt with in Section 3.4. There are various ways of deciding whether (3.20) and (3.21) are satisfied, the most straightforward being the application of the values of t_j corresponding to min k to the tableau rows representing the current basic variables.

In order to illustrate the method as given so far we present the solutions to two problems taken from [34]. The first is the one already used in Section 3.5 to illustrate the optimal cut method.

Minimize x_3 subject to

$$\begin{aligned} 5x_1 + 8x_2 - 7x_3 &\geqslant -89, \\ -6x_1 + 5x_2 + x_3 &\geqslant 11, \\ 3x_1 - 5x_2 + 2x_3 &\geqslant 29 \end{aligned}$$

with x_i non-negative integers.

As before, the numbers displayed within the following tableaux are the numerators of the appropriate elements, the common denominator D being displayed outside. The continuous optimal tableau is

	1	$-x_4$	$-x_5$	$-x_6$	
x_0	-1321	15	49	73	
x_1	121	-15	-19	-43	D = 90.
x_2	79	-15	-31	-37	
x_3	1321	-15	-49	-73	

Now (3.17) gives

$$s_0 = -29 -15(-x_4) -49(-x_5) -73(-x_6) -90k.$$

We are interested in the bounding hyperplane given by $s_0 = 0$

or $$15x_4 + 49x_5 + 73x_6 = 29 + 90k.$$

Applying the method of Section 3.4 we find that the minimum k satisfying this equation with $x_i \geq 0$ and integral is $k = 3$ and the associated solution is $x_4 = 2$, $x_5 = 4$, $x_6 = 1$. Substituting this solution in the tableau rows we obtain

$$x_1 = 121 -15(-2) -19(-4) -43(-1) = 270 \equiv 0 \text{ modulo } 90,$$

$$x_2 = 79 -15(-2) -31(-4) -37(-1) = 270 \equiv 0 \text{ modulo } 90,$$

$$x_3 = 1321 -15(-2) -49(-4) -73(-1) = 1620 \equiv 0 \text{ modulo } 90.$$

Thus we have the optimal solution, namely $x_0 = -18$, $x_1 = 3$, $x_2 = 3$, $x_3 = 18$, $x_4 = 2$, $x_5 = 4$, $x_6 = 1$.

This same problem, when solved by the optimal cut method, required six iterations and the solutions of five diophantine equations to reach the optimum from the above tableau.

As a further illustration, again we take a problem from [34], namely,

minimize x_2

subject to $66x_1 + 14x_2 \geq 1430,$

$$-82x_1 + 28x_2 \geq 1306,$$

x_1 and x_2 non-negative integers.

	1	$-x_1$	$-x_2$	
x_0	0	0	1	
x_3	-1430	-66*	-14	D = 1
x_4	-1306	82	-28	

	1	$-x_3$	$-x_2$	
x_0	0	0	1	
x_1	1430	-1	14	D = 66
x_4	-203456	82	-2996*	

	1	$-x_3$	$-x_4$	
x_0	-203456	82	66	
x_1	21746	-28	14	D = 2996.
x_2	203456	-82	-66	

This tableau represents the continuous optimum.

As before, we use (3.17) and extract from the objective row of this tableau the general equation

$$82x_3 + 66x_4 = -203456 - 2996\left[\frac{-203456}{2996}\right] + 2996\,k$$

i.e. $$82x_3 + 66x_4 = 272 + 2996\,k$$

The minimal k satisfying this equation with the condition that x_3 and x_4 are non-negative integers is $k = 1$ with $x_3 = 12$ and $x_4 = 80$. Applying these to the last tableau

$$x_1 = 21756 - 28(-12) + 14(-80) = 20972 \equiv 0 \text{ modulo } 2996,$$

$$x_2 = 203456 - 82(-12) - 66(-80) = 209720 \equiv 0 \text{ modulo } 2996.$$

All conditions are satisfied, so the optimal solution is

$$x_0 = 70,\ x_1 = 7,\ x_2 = 70,\ x_3 = 12,\ x_4 = 80.$$

In the application of the optimal cut method to some problems it was noticed that the technique sometimes became bogged down at integral levels of the functional as it was forced from the continuous solution towards the integer solution. For example, in one problem with functional values of 118.125 at the continuous optimum and 106 at the optimum respectively, the functional remained at the level 110 for 61 iterations during the solution process. It may well have been quicker to use the method just described. At least the functional value is decreased by significant amounts at each cycle and at each level the search for a feasible lattice point is limited. However, no sophisticated computer program has been written and a number of problems remain to be solved, in particular that of limiting the search for a feasible lattice point at fixed levels of the functional hyperplane,

CHAPTER IV

BRANCH AND BOUND METHODS

Another approach to the solution of integer programming problems is that typified by the so called "branch and bound" procedures. Since these involve constructing a tree of linear programming problems together with a directed search through this tree they may also be regarded as tree search procedures. The origin, or root node, of the tree is represented by the solution of the continuous problem presented when the feasibility requirements of integrality imposed on the original problem are ignored. If this solution should satisfy the integrality requirements then there is nothing further to consider. In general, some integer restricted variable, x_r, will be non-integral and consequently used to generate branches emanating from the current node. Each branch defines a linear programming subproblem the solution of which produces another node in the tree. The values of the objective function at these nodes provide bounds on x_o. The tree is not, of course, unique since it is dependent at any node upon the process of selecting x_r from among probably many variables whose values are infeasible. However, the common idea in the published methods is to attempt to develop only that part of the tree which contains the optimal solution and sufficient information to prove its optimality.

The original method in this class of techniques was proposed by Land and Doig [26]. Their method, which is applicable to mixed integer programming problems, devises a set of rules for constructing a tree which represents a succession of parallel shifts of the objective function in the direction of a decreasing maximand until it meets a point possessing the appropriate integral coordinates. In each of the continuous linear programmes of the tree some integer variable is forced to take an integer value. This can cause the generation of many branches from a node in the tree. Consequently, it is necessary to search over the range of integer values for which x_0 exceeds the currently best known feasible optimal value. This method has proved difficult to program for a computer.

Subsequent methods have been proposed and developed by Driebeck [13], Dakin [8] and Beale and Small [5]. The last two of these are very similar in concept and in contrast to the above method only produce two branches at a node by imposing bounds on x_r rather than forcing x_r to be integral valued. In general, at any node represented by the solution of some continuous linear programming subproblem an integer restricted variable x_r will be non-integral. Two new subproblems are defined, each consisting of the current constraints together with an added restriction on x_r. At the node, suppose $x_r = a_{ro} = [a_{ro}] + f_{ro}$, $0 < f_{ro} < 1$, then the constraint $x_r \leq [a_{ro}]$ is added to the current constraint set to define one problem and the alternative constraint $x_r \geq [a_{ro}] + 1$

is added to the set to define the other problem. These can then be regarded as defining two branches of the tree. Further branches can be developed each time a solution yields a non-integral value for an integer restricted variable. The solution of the original problem is complete when some branch contains a feasible optimal solution at a node while other branches terminate in problems with lower objective function values or with no solution. A description of this method now follows.

4.1 Dual Branching Tree Search Method

The methods of Dakin and Beale and Small are fundamentally similar to that of Land and Doig. One of the major differences occurs in the ways in which the basic idea is adapted for computer implementation. Dakin's aim, which was neatly achieved, was to provide a method for handling moderately sized problems completely within the high speed store of the computer. On the other hand Beale and Small developed a method capable of solving large problems but requiring the use of computer backing store. Computer considerations apart, both methods can be described as follows.

The object is to maximize some function of both discrete and continuous variables subject to linear inequality constraints. Let us suppose that x_j is an integer variable, then x_j cannot lie in the range

$$k < x_j < k + 1,$$

where $k \equiv 0$ modulo 1. Consequently, the convex region defined by the given system of linear inequalities may be divided into two non-overlapping convex regions defined by the original constraints and the additional constraints

$$\begin{aligned} &\text{(i)} \quad x_j \leq k \\ &\text{(ii)} \quad x_j \geq k + 1. \end{aligned} \tag{4.1}$$

The branch and bound method for solving the given problem is based on solving a series of ordinary linear programming problems in each of which all integer variables are bounded, that is, upper and lower bounds, as defined above, are imposed on the values of the integer variables. At any stage it is necessary to know the current best feasible solution and the associated objective function value x_{oc}. At the stage of imposing the bounds (4.1) on a selected variable, clearly, two problems are generated one of which is to be investigated immediately while the other is noted, or listed, for future investigation. In general then there exists a list of alternative problems which have been generated in earlier stages of the solution process. An overall picture of the method may be gained from the general flow chart below.

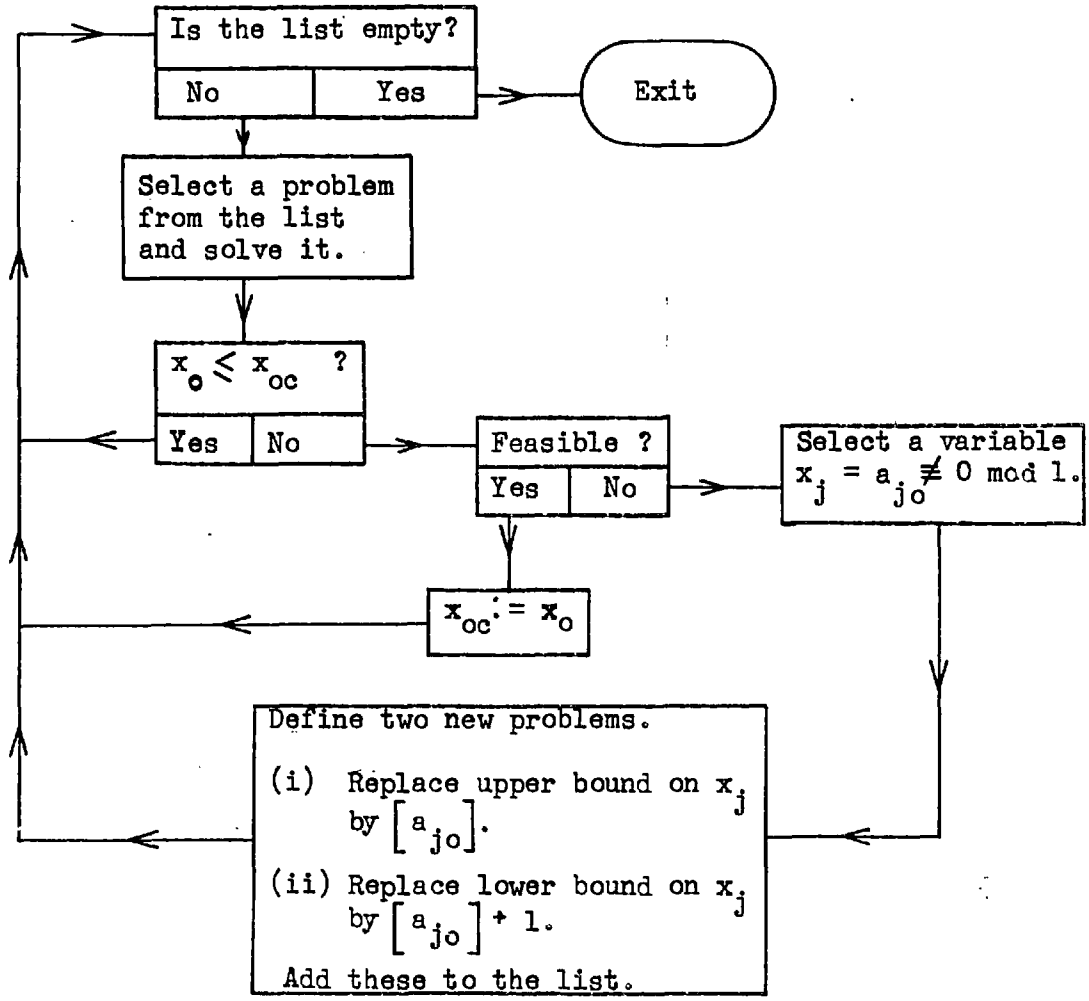


Fig. 4.1 FLOW CHART OF BRANCH AND BOUND ALGORITHM

Some slight elaboration on this flow chart will be advantageous. Beale suggests commencing the solution process by selecting plausible values for the integer variables in order to obtain a finite value for x_{oc}. The object then is to determine further plausible integer values of the appropriate variables which produce a functional value in excess of x_{oc}. The process generates a set of problems which must all be examined. The state which usually exists is one in which the current functional value exceeds x_{oc} but the solution is infeasible in that some or all of the variables required to be integral are not integral. Suppose that some such variable $x_i = a_{io} = \left[a_{io} \right] + f_{io}$, $f_{io} > 0$. Then the current single problem is replaced by two problems each of which differs from the current one in only one of the bounds placed on the variable x_i. In one problem x_i is given a lower bound of $\left[a_{io}\right]$ and in the other, x_i is given a lower bound of $\left[a_{io}\right] + 1$. For example, the problem with a range $0 \leq x_i \leq b$ is now replaced by two problems with ranges $0 \leq x_i \leq \left[a_{io}\right]$ and $\left[a_{io}\right] + 1 \leq x_i \leq b$ respectively. At such a stage, one of these new problems is examined immediately and the other is added to a list for later examination. Certain characteristics of this list, which make it computationally advantageous, are discussed in Chapter VII. Clearly, this solution process must terminate in a finite number of steps because, at worst, it would continue until the ranges of the

integer variables are reduced to zero in which case these variables must take integer values.

4.2 Selection of Branching Variable

At an optimal continuous stage many integer variables may be non-integral which implies that branching could occur on any of these and, of course, choosing different branching variables will produce different trees. This choice of branching variable is of major significance since proper selection can considerably reduce the amount of work that would otherwise be involved. In our application of this technique to the sample of thirty problems we finally based our selection, as do Beale and Small, on a strategy suggested by Little, Murty, Sweeney and Karel [27]. This selects a variable, one of whose bounds is un--likely to contain the final solution. The problem containing this particular bound is listed and the problem with the alternative bound is immediately investigated. However, prior to this final selection we attempted a number of variants with quite differing results. For example, we tried some of the selection criteria used in the cutting plane methods, namely, "of the integer variables which have not attained integrality select the first or the one with the smallest $f_{io} > 0$ or even the one with the largest $f_{io} > 0$". All of these displayed a behaviour very much inferior to two other variants which were attempted. One is the variant mentioned above, which we finally chose. The

other, which was based on a similar idea but achieved its purpose differently, was to choose that variable which produced the greatest decrease in the objective function and to use the alternative of the constraint which led to this situation. This required the nodal tableau to be augmented by appropriate branch constraints and associated with each constraint the selection of a pivot element by the dual simplex rule so that the objective function reduction factor could be obtained by a partial iteration. However, although this was successful we finally adopted the equally successful process which is described in the following chapter.

4.3 An Example

The following example serves to illustrate the method but no rule for the selection of the branching variable is specified. For ease of presentation, the tableaux are displayed in the condensed Tucker form and, furthermore, do not include the rows necessary to keep track of the ranges of values associated with the integer variables.

$$\text{Maximize } x_0 = 3x_1 + 4x_2$$

subject to

$$2x_1 + 5x_2 \leq 15,$$

$$2x_1 - 2x_2 \leq 5,$$

$$x_1, x_2 \geq 0 \text{ and integral.}$$

	1	$-x_1$	$-x_2$
x_0	0	-3	-4
x_3	15	2	5*
x_4	5	2	-2

The pivot element is selected by the primal simplex method rule and is identified by an asterisk.

	1	$-x_1$	$-x_3$
x_0	12	-7/5	4/5
x_2	3	2/5	1/5
x_4	11	14/5*	2/5

	1	$-x_4$	$-x_3$
x_0	35/2	1/2	1
x_2	10/7	-1/7	1/7
x_1	55/14	5/14	1/7

This tableau is primal and dual continuous feasible but the integrality requirements are not satisfied.

Suppose we branch on the variable x_2 and decide to explore $x_2 \geq 2$ and list $x_2 \leq 1$. Then we augment the above tableau by the row

$$s_1 = -\frac{4}{7} - \frac{1}{7}(-x_4) + \frac{1}{7}(-x_3)$$

and apply the dual simplex method to produce

	1	$-s_1$	$-x_3$
x_o	31/2	7/2	3/2
x_2	2	-1	0
x_1	5/2	5/2	1/2
x_4	4	-7	-1
s_2	-1/2	-5/2*	-1/2

Now branch on x_1 which cannot be increased because $a_{1j} \geq 0$, all $_j$ so explore $x_1 \leq \left[\frac{5}{2}\right] = 2$.

	1	$-s_2$	$-x_3$
x_o	74/5	7/5	4/5
x_2	11/5	-2/5	1/5
x_1	2	1	0
x_4	27/5	-14/5	2/5
s_3	-1/5	2/5	-1/5*

List $x_2 \geq 3$ and explore $x_2 \leq 2$.

	1	$-s_2$	$-s_3$
x_o	14	3	4
x_2	2	0	1
x_1	2	1	0
x_4	5	-2	2
x_3	1	-2	-5

This tableau is feasible so we set $x_{oc} = 14$ and extract the last listed problem which is defined by the previous infeasible tableau with the s_3 row replaced by one representing $x_2 \geq 3$ giving

	1	$-s_2$	$-x_3$
x_o	74/5	7/5	4/5
x_2	11/5	-2/5	1/5
x_1	2	1	0
x_4	27/5	-14/5	2/5
s_4	-4/5	-2/5*	1/5

	1	$-s_4$	$-x_3$
x_o	12	7/2	3/2
x_2	3	-1	0
x_1	0	5/2	1/2
x_4	11	-7	-1

This tableau is feasible but $x_o < x_{oc}$ so we extract the following listed problem.

	1	$-x_4$	$-x_3$
x_o	35/2	1/2	1
x_2	10/7	-1/7	1/7
x_1	55/14	5/14	1/7
s_5	-3/7	1/7	-1/7*

	1	$-x_4$	$-s_5$
x_o	29/2	3/2	7
x_2	1	0	1
x_1	7/2	1/2	1
x_3	3	-1	-7

This second tableau has $x_o > x_{oc}$ so a branch is necessary.

x_1 can only be decreased so explore $x_1 \leq 3$ by adding

$$s_6 = -\tfrac{1}{2} - \tfrac{1}{2}(-x_4) - 1(-s_5)$$

to the tableau and performing the pivot operation.

	1	$-s_6$	$-s_5$
x_o	13	3	4
x_2	1	0	1
x_1	3	1	0
x_3	4	-2	-5
x_4	1	-2	2

$x_o < x_{oc}$ and the list is empty so the solution is $x_1 = x_2 = 2$, $x_o = 14$.

The tree structure of this particular solution is displayed graphically in Figure 4.2. The shaded regions represent the feasible continuous space and the circled points are the optima. The order of search is indicated by the numerals.

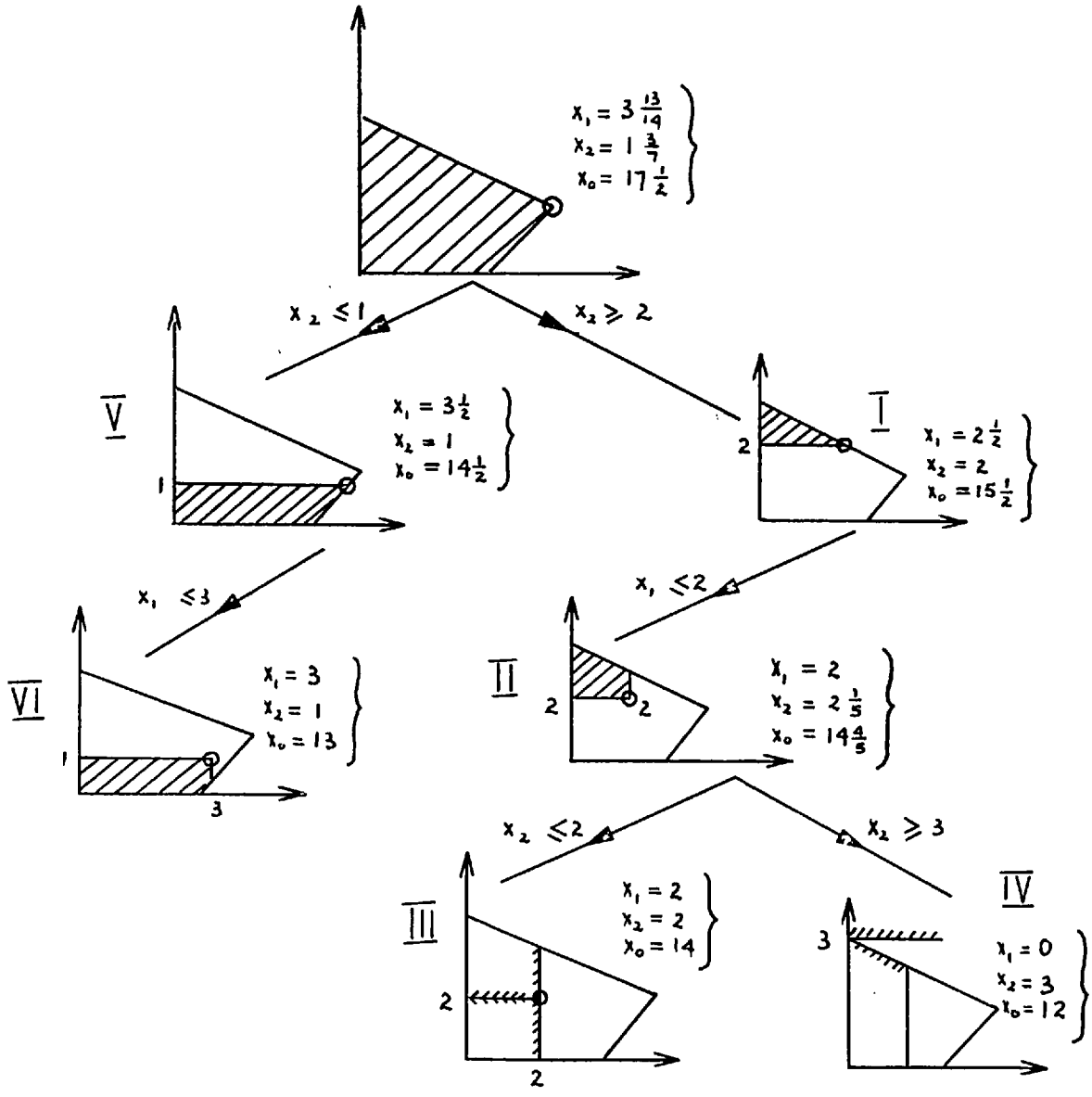


Fig. 4.2 SOLUTION TREE

Terminals III and IV illustrate situations which were not contained in the previous tableaux due to omitting the extra rows representing the ranges on the bounds of the integer variables. At terminal III, for example, the value 2 is active as both an upper bound and a lower bound on the integer variable x_2. In contrast to the method of Dakin, the above example has assumed that the tableau representing the situation at some node, from which branching has occurred, is available at any succeeding stages of the computation along these branches until such time that both branches have been exhausted. The tree is, of course, not unique since it is dependent upon the variable selected for branching.

4.4 Multiple Optima

The computational flow, if carried out exactly as described earlier in this chapter, could have a disadvantage in certain situations. This will be seen from the following discussion.

At the optimum stage of a continuous extremum problem the state of dual degeneracy, indicated by at least one $a_{oj} = 0$, $j \in N$, establishes the existence of multiple optimal solutions. The variable associated with the column of index j may now enter the basis without destroying optimality and without changing the value of the objective function. The existence of multiple optima may not be so readily indicated in a discrete extremum problem. In such a problem, the branch and bound method described in Section 4.1

partitions the continuous space enclosing the feasible lattice points into continuous subspaces by the introduction of bounds imposed on the integer variables. These bounds may, in some cases, introduce inconsistencies but these are always discovered by the method. Other bounds produce subspaces enclosing feasible sub-lattice points. The number of such subspaces may vary depending on the particular way in which branches are selected in the progression towards an optimal solution but this number is always finite. The method examines these subspaces in some order and produces, in effect, a series of optima. Throughout the method, the best feasible solution up to the current stage is retained and only those subspaces over which the optimal values of the objective function are in excess of this current best discrete optimum are investigated further. At some intermediate stage suppose we have the current best known feasible solution and further, suppose it to be an optimal solution but one of a set of multiple optima, in fact the first member of this set to be encountered. Of the subspaces not considered so far some will be associated with continuous optima violating the functional comparative criterion and will be discarded. Others will be associated with eligible continuous optima and of these some will enclose the remaining feasible optima. Each optimum must be visited because the continuous solution space enclosing it can only be partitioned into subspaces to be discarded and a subspace with an optimum equal to the current best. Thus, if

we are only interested in attaining an optimum, problems with multiple solutions will lead to unnecessary iterations. Such situations may be remedied by a simple device used in the extended method described in the following chapter.

As an illustration of this situation, consider the problem of maximizing

$$x_o = x_1 + x_2$$

subject to

$$10x_1 + 3x_2 \leqslant 55,$$
$$x_1 + 3x_2 \leqslant 12,$$

and x_1 and x_2 non-negative integers.

The following tableaux are in condensed form and again we emphasize that the rows representing the ranges on the integer variables are not recorded. The first tableau is the continuous optimal tableau and succeeding tableaux are accompanied by only brief comments.

	1	$-x_3$	$-x_4$
x_o	194/27	2/27	7/27
x_1	43/9	1/9	-1/9
x_2	65/27	-1/27	10/27
s_1	-7/9	-1/9*	1/9

Explore $x_1 \leqslant 4$, store $x_1 \geqslant 5$.

	1	$-s_1$	$-x_4$	
x_o	20/3	2/3	1/3	
x_1	4	1	0	
x_2	8/3	-1/3	1/3	
x_3	7	-9	-1	
s_2	-2/3	1/3	-1/3 *	Explore $x_2 \leqslant 2$, store $x_2 \geqslant 3$.

	1	$-s_1$	$-s_2$	
x_o	6	1	1	
x_1	4	1	0	
x_2	2	0	1	Active bounds; $x_1 \leqslant 4$, $x_2 \leqslant 2$.
x_3	9	-10	-3	
x_4	2	-1	-3	

The last tableau represents a feasible solution so set $x_{oc} = 6$ and extract the last listed subproblem which is derived from the second tableau together with the constraint $x_2 \geqslant 3$.

	1	$-s_1$	$-x_4$	
x_o	20/3	2/3	1/3	$x_o > x_{oc}$.
x_1	4	1	0	
x_2	8/3	-1/3	1/3	
x_3	7	-9	-1	
s_3	-1/3	-1/3*	1/3	Explore $x_2 \geqslant 3$.

	1	$-s_3$	$-x_4$
x_o	6	2	1
x_1	3	3	1
x_2	3	-1	0
x_3	16	-27	-10

Active bounds: $x_1 \leqslant 4$, $x_2 \geqslant 3$.

This provides a feasible solution with $x_o = x_{oc}$ so extract the last listed problem which is derived from the first tableau and the constraint $x_1 \geqslant 5$.

	1	$-x_3$	$-x_4$	
x_o	194/27	2/27	7/27	$x_o > x_{oc}$
x_1	43/9	1/9	-1/9	
x_2	65/27	-1/27	10/27	
s_4	-2/9	1/9	-1/9*	Explore $x_1 \geqslant 5$.

	1	$-x_3$	$-s_4$	
x_o	20/3	1/3	7/3	$x_o > x_{oc}$
x_1	5	0	-1	
x_2	5/3	1/3	10/3	
x_4	2	-1	-9	
s_5	-2/3	-1/3	-10/3*	Explore $x_2 \leqslant 1$.

	1	$-x_3$	$-s_5$	
x_o	31/5	1/10	7/10	$x_o > x_{oc}$
x_1	26/5	1/10	-3/10	
x_2	1	0	1	
x_4	19/5	-1/10	-27/10	
s_6	-1/5	-1/10*	3/10	Explore $x_1 \leqslant 5$, store $x_1 \geqslant 6$.

	1	$-s_6$	$-s_5$	
x_o	6	1	1	
x_1	5	1	0	
x_2	1	0	1	Active bounds: $x_1 \geqslant 5$, $x_2 \leqslant 1$, $x_1 \leqslant 5$.
x_4	4	-1	-3	
x_3	2	-10	-3	

This is again a feasible solution with $x_o = x_{oc}$ so extract from the list the tableau derived from the last tableau but one and the constraint $x_1 \geqslant 6$.

	1	$-x_3$	$-s_5$	
x_o	31/5	1/10	7/10	$x_o > x_{oc}$
x_1	26/5	1/10	-3/10	
x_2	1	0	1	
x_4	19/5	-1/10	-27/10	
s_7	-4/5	1/10	-3/10*	Explore $x_1 \geqslant 6$.

	1	$-x_3$	$-s_7$
x_0	13/3	1/3	7/3
x_1	6	0	-1
x_2	-5/3	1/3	10/3
x_4	11	-1	-9

No solution.

The solution process may now terminate because no further subproblems remain on the list.

Note that one of the multiple optima was obtained after just two iterations in the integer phase yet another four were required by this system to ascertain that no better solution existed.

4.5 Experience

Full details of our experience with this method are given in Chapter VII. At this point, since we have not yet described all aspects of the branch and bound method it is not possible to properly discuss the extent of the computation. However, the amount of work involved in progressing from a continuous optimal solution to a final solution and proving the optimality of this final solution is clearly dependent upon two factors, namely, the number of optimization subproblems which occur and the amount of computation within each of these. Due to the inclusion of the rows representing the ranges of values of the n' integer variables the amount of computation in an integer phase optimization is increased by a factor $n'(n + m + 1)^{-1}$ for each pivot operation.

The method successfully solved all problems of the sample and, taking the sample as a whole, appeared to be superior to the cutting plane method from the point of view of execution time. The average time taken over two complete runs of all 30 problems, each run representing a particular variant of the rule for breaking ties between eligible pivot columns, was 30.08 seconds compared with the best time of 32.6 seconds for the optimal cut method and the best time of 67.2 seconds for the standard cut method. The worst cutting plane time was 85.7 seconds, but it should be pointed out that this run included a problem which reached the imposed limit of 2000 iterations and still remained unsolved. Probably the most outstanding feature encountered on examination of the individual results is the comparative ease with which problems 5, 7, 8, 9, 15 and 19, all presenting great difficulty to the standard cutting plane techniques, were solved by this branch and bound method. The following table, in which the performances of one variant of the branch and bound method and four variants of the standard cutting plane method are compared, gives the number of iterations required to solve the sample of problems by the methods indicated. The branch and bound results column is extracted from the first branch and bound results table, Table 9. The best performance figures for 43% and equal best for 10% of the sample problems appear in the branch and bound results column.

In Table 4.1 the columns headed (a) and (b) represent two generator variants, viz: (a) selects row r where $f_{rk} = \min_{i \geq t} f_{ik}$, $a_{ok} = \min_{j > o} a_{oj}$; (b) selects row r such that $f_{ro} = \max_{i \geq t} f_{io}$, each variant being applied to the two situations t = o and t = 1.

Problem	Branch and Bound.	Standard cutting plane method.			
		(a)		(b)	
		Functional admissible	Functional inadmissible	Functional admissible	Functional inadmissible
1	64	21	17	14*	19
2	15	14*	21	15	15
3	20	39	39	17*	27
4	12	9	10	4*	43
5	16*	1427	69	851	451
6	13*	76	112	555	749
7	16*	> 2000	639	1858	1562
8	13*	85	510	> 2000	873
9	7*	552	813	1916	> 2000
10	8	4	10	3*	6
11	2	1*	6	1*	2
12	26	17	22	13*	27
13	43	15*	19	28	91
14	28	25*	34	235	84
15	5*	35	46	> 2000	> 2000
16	25*	26	32	55	262
17	27	11	23	5*	15
18	74	11	23	5*	15
19	93	52*	451	> 2000	> 2000
20	199*	199*	1594	199*	820
21	201	303	1595	322	760
22	201*	296	1671	306	760
23	9*	37	25	64	64
24	3*	8	7	5	8
25	3*	3*	5	3*	5
26	5*	5*	5*	5*	5*
27	2	1*	1*	1*	1*
28	2	1*	2	1*	2
29	6*	9	9	7	7
30	4*	7	8	6	6

* denotes the best performance.

TABLE 4.1 COMPARISON OF PERFORMANCES BETWEEN A BRANCH AND BOUND VARIANT AND FOUR VARIANTS OF THE STANDARD CUTTING PLANE METHOD.

CHAPTER V.

EXTENSION OF BRANCH AND BOUND METHOD.

The contents of this chapter are dependent upon the previous chapter, in fact the developments to be discussed resulted from a theoretical investigation of branch and bound methods. These developments were inspired by a statement made by Dakin [8] that his algorithm has the ability "……..to produce near optimal solutions fairly rapidly, even when it takes an unreasonably long time to obtain an optimal solution and prove that it is optimal". Our curiosity was further stimulated by another statement indicating that, in most problems he attempted, the first feasible solution encountered also proved to be the optimum. This prompted the idea that if in practice this situation proved to be generally true then it could be advantageous to establish optimality, or otherwise, for feasible points by some different process. The method to be described can, in a sense, be regarded as being independent of the branch and bound approach because it is applicable to any feasible starting point attained by whatever method, by some relaxation technique applied at the continuous optimum for example, but for simplicity and continuity we utilize the branch and bound technique for producing the first feasible solution.

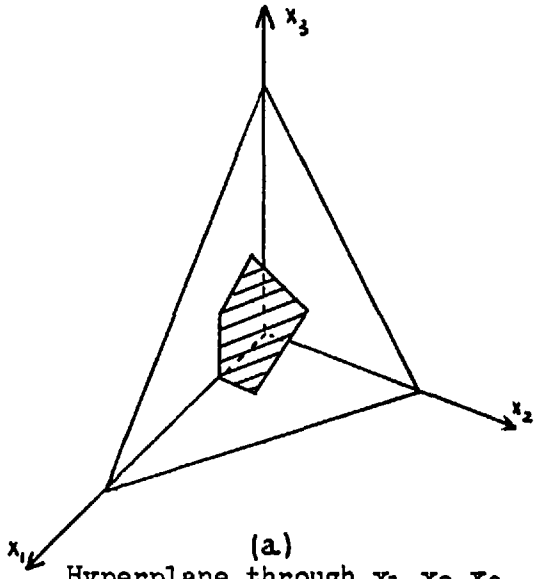
The chapter consists of seven sections. In the first we are concerned with obtaining a feasible point using the branch and bound approach. Since it is desirable to simulate the experience which led to the basic idea behind this investigation we present an argument for selecting a branching variable which appears to

define the best branch of the tree. The second and third sections describe the ideas fundamental to the method of progression from the first feasible solution point to the conclusion of the solution of the problem and are of primary importance. The fourth section describes how these ideas are brought together and used in order to carry out this progression. The remaining three sections discuss various aspects of the method. Section 5.5 discusses the types of difficulties that can arise in the method and presents ways of overcoming these. Section 5.6 proposes some possible variants of the method. Finally, a small example followed by a discussion on our experience with the method completes the chapter with Section 5.7.

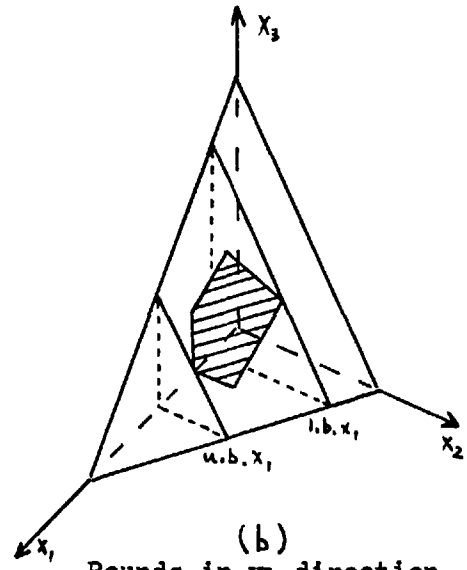
5.1 Variable Selection

Following the presentation of Land and Doig [26], let us consider the intersection of the functional hyperplane and the n-dimensional convex set over which it is to be optimized. A common convex set in (n - 1) dimensions is produced. In this (n - 1)-dimensional convex set each of the n variables has a maximum and a minimum value as illustrated by Figures 5.1 (a), (b), (c) and (d).

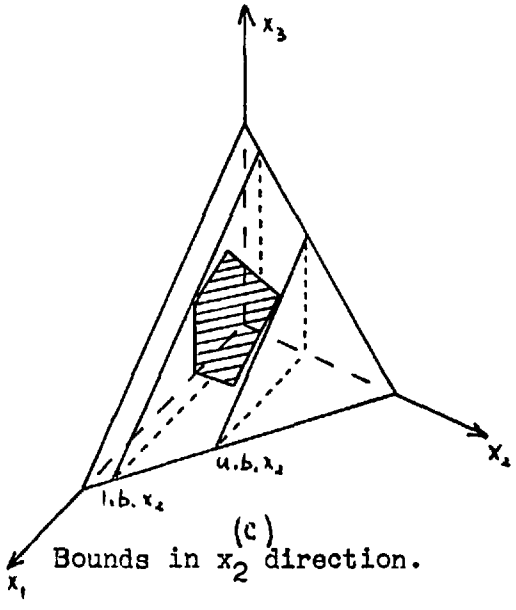
Knowing for a particular value of the functional the associated minimum and maximum values of each variable one also knows whether or not there is a possible integer value of each variable for that particular position of the functional hyperplane. If there is not at least one integral value for every variable x_j then clearly there exists no solution to the problem at such a value of the



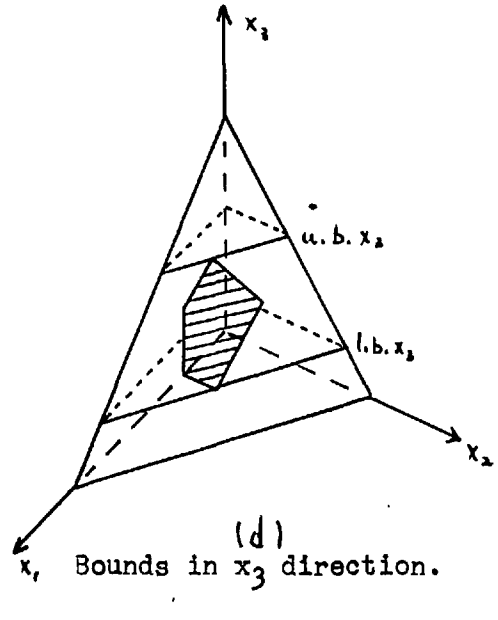
(a)
Hyperplane through x_1, x_2, x_3 cutting three dimensional convex.



(b)
Bounds in x_1 direction.



(c)
Bounds in x_2 direction.



(d)
Bounds in x_3 direction.

FIG. 5.1 Intersection of objective function and three dimensional space.

maximand. The converse of this result is not true however. Existence of integer values for all the variables x_j does not imply a solution at such a value of the maximand for the simple reason that these co-ordinates need not necessarily coincide on the hyperplane. Since, at a particular value of the functional, there is a unique maximum and minimum for any variable x_k, we can define two functional relationships min x_k and max x_k between x_k and the functional. We illustrate this point by means of the following example.

$$\begin{aligned} \text{Maximize} \quad & 3x_1 + 4x_2 \\ & 2x_1 + 5x_2 \leqslant 15, \\ \text{subject to} \quad & 2x_1 - 2x_2 \leqslant 5, \\ & 4x_1 - 4x_2 \geqslant -5, \\ & 4x_1 + 4x_2 \geqslant 5 \\ \text{and} \quad & x_1, x_2 \geqslant 0. \end{aligned}$$

The continuous optimal solution is given by the tableau

	1	$-x_4$	$-x_3$
x_0	35/2	1/2	1
x_1	55/14	5/14	1/7
x_2	10/7	-1/7	1/7
x_6	115/7	6/7	8/7
x_5	15	2	0
	c_1	c_2	c_3

(The c's are merely column identifiers for later use).

and is represented by the Figure 5.2.

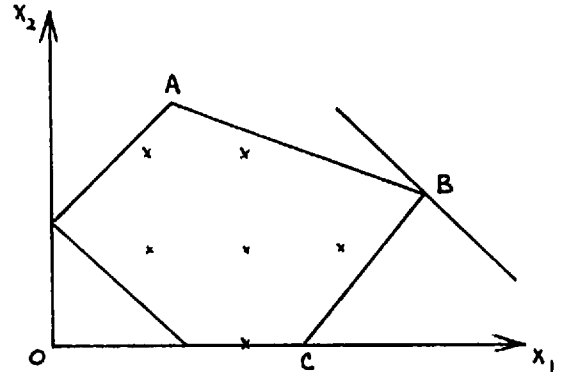


FIG. 5.2

Due to the integrality requirements x_0 must also be integral, so let us examine what happens as the functional is moved parallel to itself from B into the convex in such a way that at each step the functional attains an integral value. The intersection of the functional and BA and BC can be traced out as shown by the following table which illustrates the first four steps.

	BA path		column combination.	BC path		column combination.	Range	
x_0	x_1	x_2		x_1	x_2		x_1	x_2
17	$\frac{25}{7}$	$\frac{11}{7}$	$c_1 - c_2$	$\frac{27}{7}$	$\frac{19}{14}$	$c_1 - \frac{1}{2}c_3$	$\frac{25}{7} \leq x_1 \leq \frac{27}{7}$	$\frac{19}{14} \leq x_2 \leq \frac{22}{14}$
16	$\frac{20}{7}$	$\frac{13}{7}$	$c_1 - 3c_2$	$\frac{26}{7}$	$\frac{17}{14}$	$c_1 - \frac{3}{2}c_3$	$\frac{20}{7} \leq x_1 \leq \frac{26}{7}$	$\frac{17}{14} \leq x_2 \leq \frac{26}{14}$
15	$\frac{15}{7}$	$\frac{15}{7}$	$c_1 - 5c_2$	$\frac{25}{7}$	$\frac{15}{14}$	$c_1 - \frac{5}{2}c_3$	$\frac{15}{7} \leq x_1 \leq \frac{25}{7}$	$\frac{15}{14} \leq x_2 \leq \frac{30}{14}$
14	$\frac{10}{7}$	$\frac{17}{7}$	$c_1 - 7c_2$	$\frac{23}{7}$	$\frac{13}{14}$	$c_1 - \frac{7}{2}c_3$	$\frac{10}{7} \leq x_1 \leq \frac{23}{7}$	$\frac{13}{14} \leq x_2 \leq \frac{34}{14}$

TABLE 5.1 x_1 and x_2 ranges.

At specific values of the functional x_0 the corresponding min x_1, max x_1 and min x_2, max x_2 are given by the range columns of Table 5.1.

Consider the convex polyhedron in three dimensional space (x_0, x_1, x_2) defined by the example constraints without the integrality requirements. A sample of its boundary points is illustrated in the Figure 5.3.

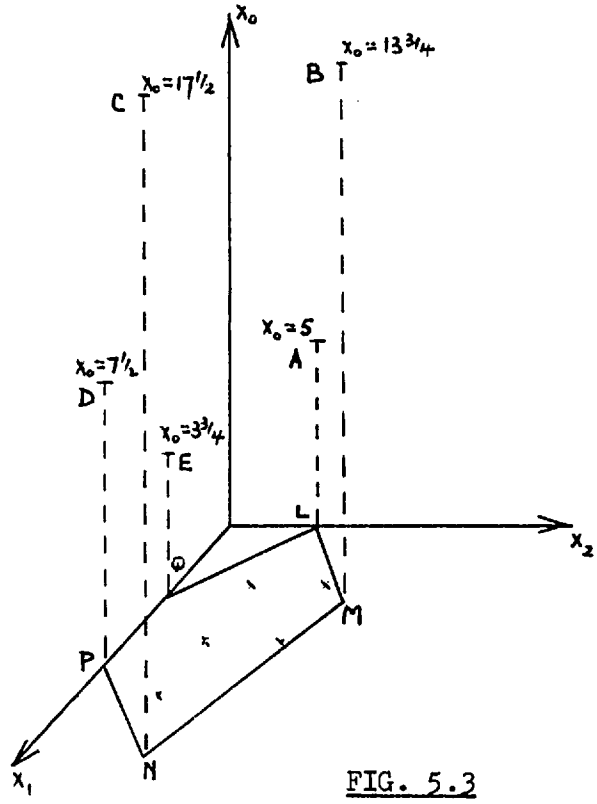


Figure defining the polyhedron

$$x_0 - 3x_1 - 4x_2 = 0,$$
$$2x_1 + 5x_2 \leqslant 15,$$
$$2x_1 - 2x_2 \leqslant 5,$$
$$-4x_1 + 4x_2 \leqslant 5,$$
$$-4x_1 - 4x_2 \leqslant -5,$$
$$x_1,\ x_2 \geqslant 0$$

with vertices A,B,C,D,E,L,M,N,P,Q.

FIG. 5.3

If we project this three dimensional space onto (x_0, x_1), say, then the convex polygon bounded by min x_1 and max x_1, specific points of which are given in Table 5.1, is produced as illustrated in Figure 5.4.

The highest point of the polygon, with $x_0 = 17\frac{1}{2}$, corresponds to the continuous optimal solution of the original problem.

Thus far, we have merely illustrated some ideas which can be generalized to n dimensions. With these preliminaries complete let us now return to the major problem of this section.

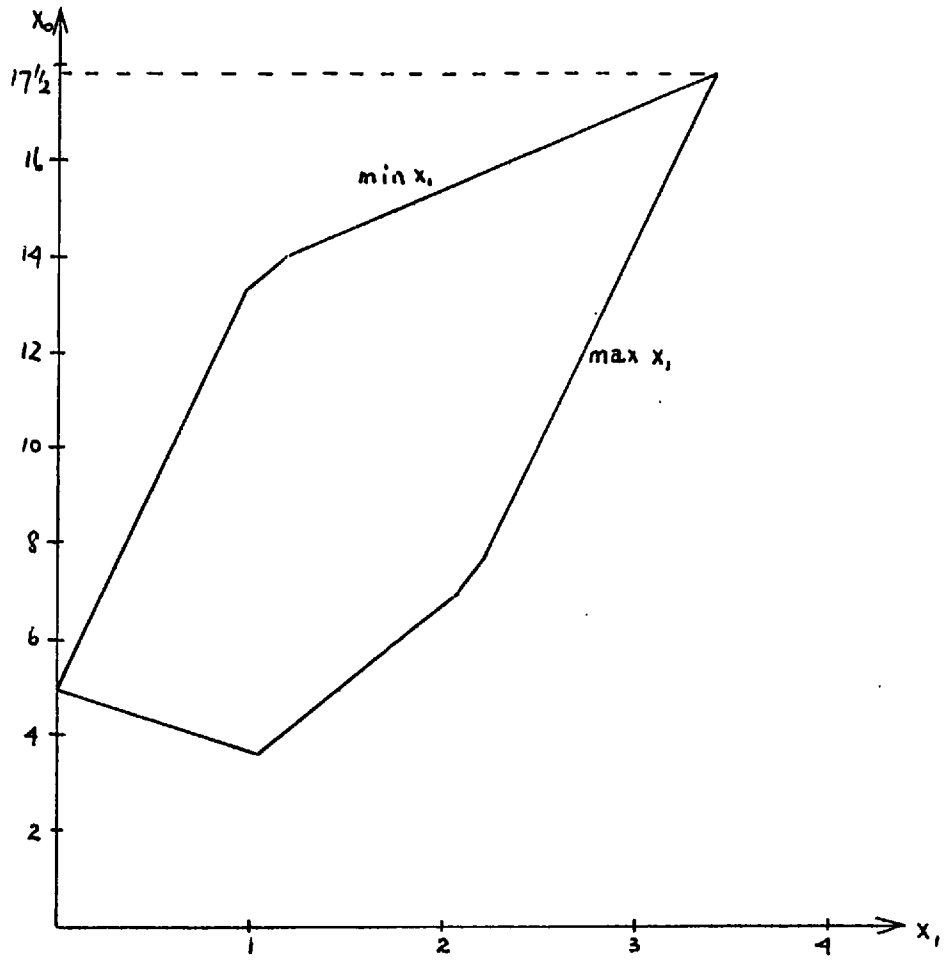


FIG. 5.4 Projection of (x_0, x_1, x_2) space onto (x_0, x_1) space.

Suppose that we have the optimal solution of a subproblem generated by the branch and bound method and further, that a subset of the variables required to take integer values have not yet attained integrality. In the manner illustrated above let us project the (n + 1)- dimensional space of variables (n) and the objective onto the two-dimensional space (x_0, x_i), where x_0 is the objective function variable and x_i is one of the basic variables. The figure produced will be a convex bounded by the lines min x_i and max x_i. Suppose the current tableau to

be expressed by

$$x_o = a_{oo} + \sum_{j \notin B} a_{oj}\,(-x_j),$$

$$x_i = a_{io} + \sum_{j \notin B} a_{ij}\,(-x_j), \quad i \in B, \text{ the set of indices of the current basic variables.} \qquad (5.1)$$

Let x_{oc} be the best value to date of x_o consistent with an integer solution. The Figure 5.5 represents the projection of the $(x_o, x_1, \ldots\ldots, x_n)$space defined by (5.1) onto (x_o, x_i).

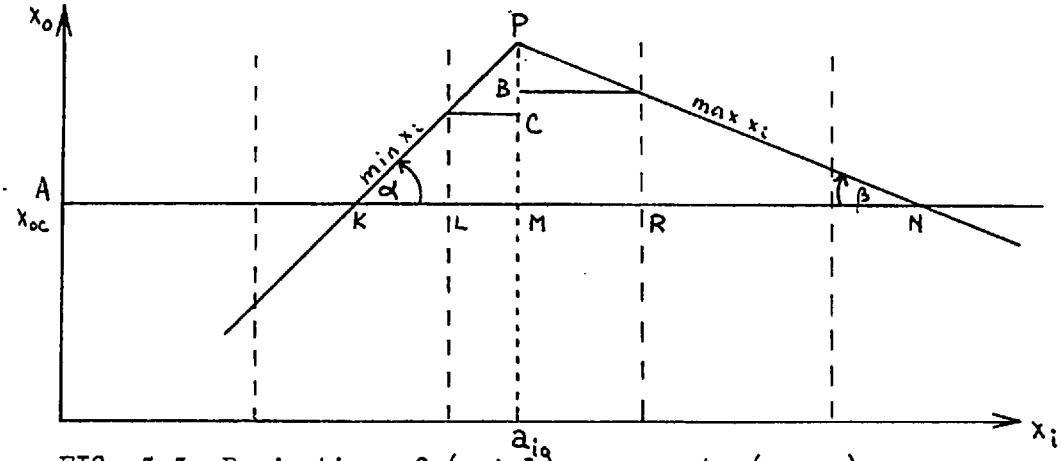


FIG. 5.5 Projection of $(n + 1)$ space onto (x_o, x_i).

P represents the current continuous optimum and x_{oc} is the best feasible optimum to date.

$PK \equiv \min x_i$, $PN \equiv \max x_i$.

L and R are the intersections of AN and the integral lattice lines bounding a_{io}. Since we have a solution at x_{oc} the only problems worth considering are those for which $x_o > x_{oc}$. Now, from the figure, for x_o to exceed x_{oc} the variable x_i must satisfy

$$AK < x_i < AN.$$

(Note. This isn't the case for the situations $\alpha > \frac{\pi}{2}$ or $\beta > \frac{\pi}{2}$. In such situations M lies outside the range KN and in the one case x_i is bounded below by PM ($x_i > AM$) and in the other x_i is bounded

above by PM (x_i <AN)).

Now
$$AK = AM - KM = AM - \frac{PM}{\tan\alpha} = \left[a_{io}\right] + f_{io} - \frac{a_{oo} - x_{oc}}{\tan\alpha}.$$

So
$$x_i > \left[a_{io}\right] - \frac{a_{oo} - x_{oc} - f_{io}\tan\alpha}{\tan\alpha}.$$

But $x_i \equiv 0$ modulo 1,

hence
$$x_i \geq \left[a_{io}\right] - \left[\frac{a_{oo} - x_{oc} - f_{io}\tan\alpha}{\tan\alpha}\right]. \tag{5.2}$$

Again,
$$AN = AM + MN = AM + \frac{PM}{\tan\beta} = \left[a_{io}\right] + \frac{a_{oo} - x_{oc} + f_{io}\tan\beta}{\tan\beta}.$$

Therefore
$$x_i < \left[a_{io}\right] + \frac{a_{oo} - x_{oc} + f_{io}\tan\beta}{\tan\beta},$$

but $x_i \equiv 0$ modulo 1,

and so
$$x_i \leq \left[a_{io}\right] + \left[\frac{a_{oo} - x_{oc} + f_{io}\tan\beta}{\tan\beta}\right]. \tag{5.3}$$

Thus, if x_o is to exceed x_{oc} we must have

$$\left[a_{io}\right] - \left[\frac{a_{oo} - x_{oc} - f_{io}\tan\alpha}{\tan\alpha}\right] \leq x_i \leq \left[a_{io}\right] + \left[\frac{a_{oo} - x_{oc} + f_{io}\tan\beta}{\tan\beta}\right]. \tag{5.4}$$

We will now attach meanings to α and β in terms of the tableau. We need consider only one of these, say α.

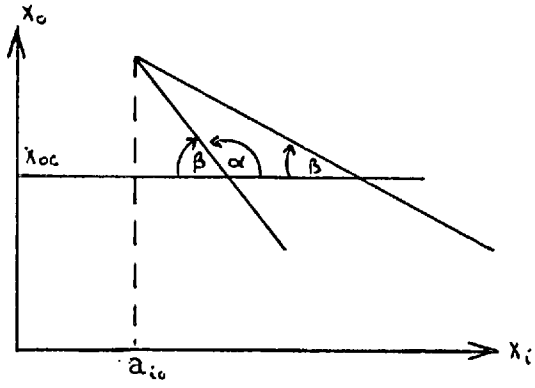
Now, $\tan\alpha = \frac{PM}{KM}$ = change in x_o per unit change in x_i, and so in terms of the cost ranging procedure of linear programming, $\tan\alpha$ is the marginal cost of changing x_i. In this particular instance, $\tan\alpha$ is the marginal cost of decreasing x_i and $\tan\beta$ is the marginal cost of increasing x_i. In terms of the tableau elements we have

$$\tan\alpha = \min_{j \mid a_{ij} > 0} \left\{\frac{a_{oj}}{a_{ij}}\right\} \tag{5.5}$$

$$\text{and} \quad \tan\beta = \min_{j|a_{ij}<0} \left\{ \frac{a_{oj}}{|a_{ij}|} \right\} . \qquad (5.6)$$

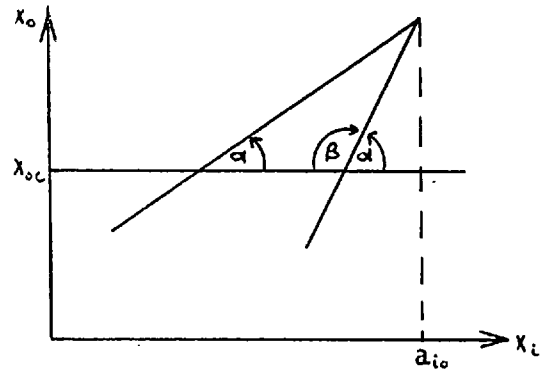
It may be worth noting that (5.5) is undefined for $a_{ij} < 0$ and (5.6) is undefined when $a_{ij} > 0$.

When considering x_i suppose that all $a_{ij} < 0$ then (5.5), and consequently a lower bound in (5.4), have not been defined. What this situation means in fact is that α exceeds $\frac{\pi}{2}$, in which case $(\pi - \alpha)$ becomes essentially another β. However, the correct β is selected by (5.6) and the upper bound of (5.4) will be correctly defined. Under these circumstances let $\tan\alpha$ be infinite then (5.4) will be defined completely. Similarly, if all $a_{ij} > 0$ then $\tan\beta$ and the upper bound of (5.4) are undefined. This situation is catered for by defining $\tan\beta$ to be infinite under these circumstances. Both situations are illustrated



$\tan\alpha$ undefined

FIG. 5.6(a)



$\tan\beta$ undefined.

FIG. 5.6(b)

in the Figures 5.6(a) and (b). All this means is that in the first case x_i cannot be decreased below its current value and in the second case x_i cannot be increased above its current value. The strategy employed in [27] in order to select a branching variable

in the travelling salesman problem was to branch on that variable which made the worse of the upper and lower bounds on the range of the variable as bad as possible. This worse alternative is unlikely to produce the final solution; it is given by

$$\begin{aligned} &\max (PB, PC) = \max (MR \tan\beta , LM \tan\alpha) \\ &= \max \{ (1-f_{io}) \tan\beta , f_{io} \tan\alpha \}. \end{aligned} \tag{5.7}$$

(When $\beta > \frac{\pi}{2}$ MR is undefined, when $\alpha > \frac{\pi}{2}$ LM is undefined and in either case no alternative exists).

Therefore, efficient selection of a branching variable can be achieved by (5.5), (5.6) and (5.7). To each integer variable x_i which has not attained integrality, so that $x_i = a_{io} + f_{io}$, $f_{io} > 0$, apply (5.5) and (5.6) then compute $(1 - f_{io}) \tan\beta$ and $f_{io} \tan\alpha$. Finally, select that variable which satisfies (5.7).

5.2 Bound Contraction

As mentioned earlier, investigators of tree search methods have reported that the optimal solution often appears to be reached at an early stage in the progress through the solution process. At this stage many subproblems may, however, remain and must all be examined prior to termination of the search. If this is in fact the case it may be advantageous to proceed as follows . Obtain an integer solution, probably discarding alternative problems en route if the branch and bound technique is used to attain this first feasible solution. Employ this feasible solution and the functional to construct a new constraint which will exclude this solution and produce a new

convex. Two possibilities can exist, either we already have the optimal solution or the new convex is not empty and contains the desired lattice point. If an approach along these lines is adopted it will thus be necessary to determine whether or not the new convex is empty. We now investigate this aspect.

Suppose the constraints defining the convex enclosing the feasible solution space, if it exists, are

$$\sum_{j \in N} a_{ij} x_j \leqslant a_{io}, \quad i = 1, \ldots, t; \tag{5.8}$$

Suppose further that x_j is bounded thus

$$\underline{\phi}_j \leqslant x_j \leqslant \overline{\phi}_j \quad .$$

Now let either bound on x_j be represented by

$$\phi_{ij} = \begin{cases} \overline{\phi}_j \text{ if } a_{ij} < 0, \\ 0 \text{ if } a_{ij} = 0, \\ \underline{\phi}_j \text{ if } a_{ij} > 0 \quad . \end{cases} \tag{5.9}$$

Then for any x_j in (5.8) we have

$$a_{ij} \phi_{ij} \leqslant a_{ij} x_j. \tag{5.10}$$

Now, in order to consider a particular variable x_k, from the set of constraints (5.8) extract the i-th and rewrite it as

$$\sum_{j \neq k} a_{ij} x_j + a_{ik} x_k \leqslant a_{io} \cdot$$

Then, using this and (5.10), we obtain

$$\begin{aligned} a_{ik} x_k &\leqslant a_{io} - \sum_{j \neq k} a_{ij} x_j \\ &\leqslant a_{io} - \sum_{j \neq k} a_{ij} \phi_{ij} \\ &\leqslant a_{io} - \sum_{j \in N} a_{ij} \phi_{ij} + a_{ik} \phi_{ik}. \end{aligned} \tag{5.11}$$

Let $$\delta_i = a_{io} - \sum_{j \in N} a_{ij} \phi_{ij}$$

then (5.11) may be rewritten as

$$a_{ik}\, x_k \lesseqgtr \delta_i + a_{ik}\, \phi_{ik}. \tag{5.12}$$

Now, when $a_{ik} > 0$ inequality (5.12) can be written as

$$x_k \lesseqgtr \frac{\delta_i}{a_{ik}} + \phi_{ik} \quad ,$$

which by (5.9) becomes

$$x_k \lesseqgtr \frac{\delta_i}{a_{ik}} + \underline{\phi}_k \quad .$$

This can be expanded to give

$$x_k \lesseqgtr \bar{\phi}_k + (\underline{\phi}_k - \bar{\phi}_k + \frac{\delta_i}{a_{ik}}) \quad . \tag{5.13}$$

When $a_{ik} < 0$ inequality (5.12) can be transformed by a similar process to produce

$$\begin{aligned} x_k &\gtreqless \frac{\delta_i}{a_{ik}} + \bar{\phi}_k \\ &\gtreqless \underline{\phi}_k + (\bar{\phi}_k - \underline{\phi}_k + \frac{\delta_i}{a_{ik}}) \quad . \end{aligned} \tag{5.14}$$

Let $\Delta_{ik} = \begin{cases} \underline{\phi}_k - \bar{\phi}_k + \dfrac{\delta_i}{a_{ik}} & \text{for } a_{ik} > 0, \\ \bar{\phi}_k - \underline{\phi}_k + \dfrac{\delta_i}{a_{ik}} & \text{for } a_{ik} < 0. \end{cases}$

Then, using (5.13) and (5.14), we see that the variable x_k must satisfy

$$x_k \lesseqgtr \bar{\phi}_k + \Delta_{ik} \text{ if } a_{ik} > 0 \tag{5.15}$$

$$\text{or} \quad x_k \gtreqless \underline{\phi}_k + \Delta_{ik} \text{ if } a_{ik} < 0. \tag{5.16}$$

Clearly then an improved bound may be obtained for x_k because if $\Delta_{ik} < 0$ (5.15) may be used to decrease its upper bound while if $\Delta_{ik} > 0$ (5.16) may be used to increase its lower bound. Furthermore, if $\delta_i < 0$, it can be seen from (5.13) or (5.14) that contradictions will occur because, for example, we could have

$$x_k \gtreqless \underline{\phi}_k$$

and from (5.13)

$$x_k \leq \phi_k - \alpha, \quad \alpha > 0,$$

which are contradictory.

5.3 Convex Isolation

The development of the above bound contraction process has presupposed the existence of bounds on the integer variables. Generally we may assume that bounds for the integer variables in a problem are not known but once the continuous optimal vertex has been determined we can compute such bounds in the following manner.

Firstly, it should be noted that we can often take advantage of the branch and bound method itself in obtaining bounds. Suppose, within the complete solution process, we are at a stage between the production of the continuous optimum and the first feasible solution. If during the branching variable selection step defined in Section 5.1 we have that $\tan\alpha$ or $\tan\beta$ is infinite and no other subproblems have been listed, then we can define a bound on the variable concerned. For example, if $x_k = a_{ko} \not\equiv 0$ modulo 1 and $\tan\alpha$ is infinite then we know that a lower bound on this variable is given by $x_k \geq \lfloor a_{ko} \rfloor + 1$. Similarly, if $\tan\beta$ is infinite an upper bound would be given by $x_k \leq \lfloor a_{ko} \rfloor$. Of course, if any previously listed subproblem existed at this stage these bounds would be applicable only on the particular branch with which we are concerned. They may also apply on branches containing the other subproblems but at this stage we cannot decide this from the current problem. Situations in which this retaining of bounds en route to the first feasible solution is

advantageous are discussed later in this chapter.

To return to the idea introduced earlier, suppose that we have produced the continuous optimum and the first feasible solution. The object now is to use this first feasible point in order to isolate a region of the original convex. To do this we use the objective function and the feasible solution to define a new functional constraint. This, as the name may imply, is a hyperplane which is parallel to the objective function hyperplane and is positioned so as to have a constant term which exceeds the feasible optimal functional value. As an example suppose our current feasible point is $(x_1^*, x_2^*, x_3^*, x_4^*)$ and that the functional to be maximized is

$$x_0 = a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 .$$

Then at $(x_1^*, x_2^*, x_3^*, x_4^*)$

$$x_0 = a_1 x_1^* + a_2 x_2^* + a_3 x_3^* + a_4 x_4^* = k(\text{say}).$$

If this functional is to be improved its value must be increased by some amount $d > 0$ so that the functional constraint is given by

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 \geq k + d.$$

The value of d is dependent upon the type of problem being considered. For a mixed integer problem all we can say here is that d is positive and may be as large as possible subject to the condition that no feasible lattice point, other than the first obtained, is excluded from the new convex region. Although we have successfully solved some small mixed integer problems by hand our computer code has been written to cater for the pure integer problems of the data sample. Due to this, we now confine our attention to pure integer problems.

Let

$$x_o = a_1 x_1 + a_2 x_2 + \ldots\ldots + a_n x_n ,$$

where $x_i \equiv 0$ modulo 1.

Now, to use a term from number theory, the aggregate $\sum_{i=1}^{n} a_i x_i$ forms a modulus, that is a system S of numbers such that the sum and difference of any two members of S are themselves members of S. Let d be the greatest common divisor of $a_1, a_2, \ldots, a_n$ then the modulus $\sum_{i=1}^{n} a_i x_i$ is the aggregate of multiples of d so that any change in the magnitude of x_o will be some integral multiple of d. Consider the point - lattice $\mathcal{L}$ and suppose that the objective function hyperplane defined by $x_o = \sum_{i=1}^{n} a_i x_i$ is at some point P of $\mathcal{L}$ such that the value of the objective function is $x_o = k$, say. It is now necessary to move the functional away from P in such a direction that the value of x_o is increased above k and by such a distance that the space between the two adjacent functional positions contains no member of $\mathcal{L}$. Clearly, by the above, the magnitude of such a distance will be a function of d and, in fact, the minimum increase in x_o is d.

So we define our functional constraint to be

$$\sum_{j \in N} a_{oj} x_j \geq x_{oc} + d , \qquad (5.17)$$

where d is the greatest common divisor of the objective function coefficients.

The continuous optimum is defined by a subset of the hyperplanes defining the original convex and can be identified by the variables which are nonbasic in the primal and dual continuous feasible tableau.

These hyperplanes emanating from the continuous optimal vertex together with the functional constraint define our new convex region, which we call the isolated convex. The vertices of this isolated convex are used to define a set of lattice planes bounding any lattice points which the isolated convex may contain. Suppose that the co-ordinates of the vertices of the isolated convex are $(x_{i1}, \ldots\ldots, x_{in})$, $i = 1, \ldots\ldots, v$, one of which is, of course, the continuous optimal vertex. Then the lattice planes are defined by

$$\eta_{j1} \leqslant x_j \leqslant \eta_{j2}, \quad j = 1, \ldots, n, \tag{5.18}$$

where
$$\eta_{j1} = \min_i x_{ij} \text{ if } \min_i x_{ij} \equiv 0 \text{ modulo } 1,$$
$$= \left[\min_i x_{ij}\right] + 1 \text{ if } \min_i x_{ij} \not\equiv 0 \text{ modulo } 1$$

and
$$\eta_{j2} = \left[\max_i x_{ij}\right], \quad i = 1, \ldots, v.$$

5.4 Search for a Superior Feasible Point

The ideas fundamental to this method have now been described and so can be brought together in a way which facilitates the search for a superior feasible point. To review the situation, let us assume that the following steps have already been carried out.

(i) The continuous optimum has been found with a continuous maximal objective function value of x_o'.

(ii) The first feasible solution, with an objective function value of x_{oc}, and possibly a set of integer variable bounds have been obtained by the branch and bound method.

(iii) A functional constraint excluding this feasible point has been defined.

Clearly, the first thing to determine at this point is whether or not

$$d > x_o' - x_{oc}, \tag{5.19}$$

where d is the greatest common divisor of the objective function coefficients. If (5.19) is true then the isolated convex R that would be produced by the intersection of the functional constraint and the constraints defining the continuous optimal vertex would be entirely infeasible. The current solution is then the best. If (5.19) is untrue, the isolated convex will define a continuous space which may or may not contain a feasible subspace and we proceed with the following step.

(iv) Determine the points of intersection of the functional constraint and those defining the continuous optimal vertex. Use these vertices, together with the continuous optimal vertex, to determine bounds on the integer variables. From this set of bounds and any specified by (ii) extract the best to define L, a convex set bounded by planes of the fundamental lattice.

We now want to examine $L \frown R$ and ascertain whether or not it contains a feasible lattice point and if so find one. To do this we employ the contraction process described in Section 5.2. Firstly though we should examine L itself because, by our definition, a contradiction can arise in the determination of L. If, for example, the vertices of the isolated convex are given by

$(x_{i1}, \ldots, x_{in})$, $i = 1, \ldots, t$ then let us consider just the first component of each and suppose that $x_{i1} = K + \epsilon_i$, $i = 1, \ldots, t$, where K is a positive integer and $0 < \epsilon_i < 1$. Then, by (5.18) we obtain

$$\eta_{11} = \left[\min_i x_{i1} \right] + 1 = K + 1,$$

$$\eta_{12} = \left[\max_i x_{i1} \right] = K$$

and $K + 1 \leqslant x_1 \leqslant K$

which is a contradiction. This implies that R contains no lattice point and that the current feasible solution is optimal.

Now we assume that L is non-empty, in which case it must contain members of the point-lattice $\mathcal{L}$, and proceed to describe the contraction process.

Suppose any bounds produced prior to step (iv) are given by

$$l_j \leqslant x_j \leqslant n_j.$$

Let (5.8) define the isolated convex R and let (5.18) define L, where it is now assumed that

$$\eta_{j1} = \max \left\{ \min_i x_{ij}, l_j \right\} \text{ if } \min_i x_{ij} \equiv 0 \text{ modulo } 1,$$

$$= \max \left\{ \left[\min_i x_{ij} \right] + 1, l_j \right\} \text{if } \min x_{ij} \not\equiv 0 \text{ modulo } 1$$

and $$\eta_{j2} = \min \left\{ \max_i x_{ij}, n_j \right\}.$$

For ease of reference R and L are given by

$$\sum_{j \in N} a_{ij} x_j \leqslant a_{io}, \quad i = 1, \ldots, t, \tag{5.20}$$

and

$$\eta_{j1} \leqslant x_j \leqslant \eta_{j2}, \tag{5.21}$$

respectively.

For each constraint (5.20) compute

$$\delta_i = -\sum_{j \in N} a_{ij}\, \phi_{ij} + a_{io}\ , \qquad (5.22)$$

where ϕ_{ij}, as defined by (5.9), uses the bounds given by (5.21). If $\delta_i < 0$ then the application of (5.13) or (5.14) would produce a contradiction which implies that $\mathcal{L} \frown R$ is empty and consequently the current solution is optimal. If $\delta_i \geqslant 0$ then we attempt to contract L by applying (5.13) and (5.14) to $L \frown R$ as follows

$$x_j \leqslant \eta_{j1} + \frac{\delta_i}{a_{ij}} \quad \text{if } a_{ij} > 0$$

$$\text{and} \qquad x_j \geqslant \eta_{j2} + \frac{\delta_i}{a_{ij}} \quad \text{if } a_{ij} < 0 \ , \ j \in N.$$

If, for some i, any change occurs in the bounds then the new values displace the present ones before i is incremented. This implies that (5.22) always operates on the currently best bounds. A complete pass over i = 1,....,t is called a cycle. The occurrence of a bound change does, of course, introduce the possibility of alternative paths in the process. A change in either bound on some variable x_j at a stage i = s, say, could affect some δ_i, $i < s$. We could take account of this situation by returning either to the beginning of the cycle, i = 1, or to some stage, i = p, such that δ_p is affected but δ_i, $i < p$, is unaffected. On the other hand we can ignore it and continue the cycle with i = s + 1,....,t and, for simplicity, this is what we have decided to do. This cycle process is repeated until a cycle is executed which involves no change in the bounds on any of the x_j and this then is as far as the bound contraction process can proceed. The following situations can exist at the end of a cycle during which no contradiction has

arisen and no bound has changed.

(i) $L \frown R$ has been contracted to a region $L^1 \frown R$ in which L^1 is given by a range similar to (5.21) such that

$$\eta_{j2} - \eta_{j1} \geq 1 \tag{5.23}$$

for at least one $j \in N$.

(ii) $L \frown R$ has been contracted to a lattice point.

In the event of (i) we define L_1 and L_2 where

$$L_1 \frown L_2 = 0 , \quad L_1 \smile L_2 \subset L^1$$

so that L_1 and L_2 are two non-intersecting subregions of L^1. In fact, to be more specific, we may select any variable for which (5.23) is true and dichotomize its present range

$$\eta_{k1} \leqslant x_k \leqslant \eta_{k2}$$

to the two ranges

$$\eta_{k1} \leqslant x_k \leqslant \xi_k \tag{5.24}$$

and

$$\xi_k + 1 \leqslant x_k \leqslant \eta_{k2} \tag{5.25}$$

where $\xi_k = \eta_{k2} - 1 - \left[\frac{\eta_{k2} - \eta_{k1}}{2}\right]$. No feasible space is omitted by this process since $\xi_k < x_k < \xi_k + 1$ is empty of lattice points. Thus, L_1 and L_2 have the same bounds as L^1 in all variables except x_k. We now attempt to contract $L_1 \frown R$ in the same manner as before and if $L_1 \frown R$ contains no lattice points then we repeat the process with $L_2 \frown R$. The process of contraction will thus terminate in one of two ways. Either we prove that $L \frown R$ is empty of feasible lattice points or the situation (ii) occurs.

For a reason to be discussed in the next section, in the event of (ii) we must apply a partial feasibility test to the lattice point produced by the contraction process. If it is infeasible and no L_i resulting from dichotomies remain then the current feasible solution is optimal. If, however, it is feasible then it provides a solution superior to the current one. This in turn supplies a new x_{oc} to step (ii) given at the start of this section and the process outlined above is recommenced from step (iii) using the new superior feasible point.

5.5 Difficulties Within the Method

We first present the reason for the partial feasibility test mentioned in the previous section and commence by recalling the process of defining the isolated convex. The constraints defining the continuous optimal vertex are extracted from the tableau. These and the functional constraint define the bounding hyperplanes of the isolated convex. Now, clearly, this convex may or may not be a subset of the original convex. If it is then any enclosed point-lattice is feasible, if not then infeasible lattice points may be admitted. This situation is illustrated by the following problem and the associated diagram (Figure 5.7).

$$\begin{aligned} \text{Maximize} \quad & 4x_1 + x_2 \\ \text{subject to} \quad & 3x_1 + 2x_2 \leqslant 17, \\ & 2x_1 - x_2 \leqslant 7, \\ & 2x_1 - 3x_2 \leqslant 4, \end{aligned}$$

$$x_1 + x_2 \geq 3,$$
$$-2x_1 + 6x_2 \leq 3,$$
$$x_1, x_2 \geq 0 \text{ and integral.}$$

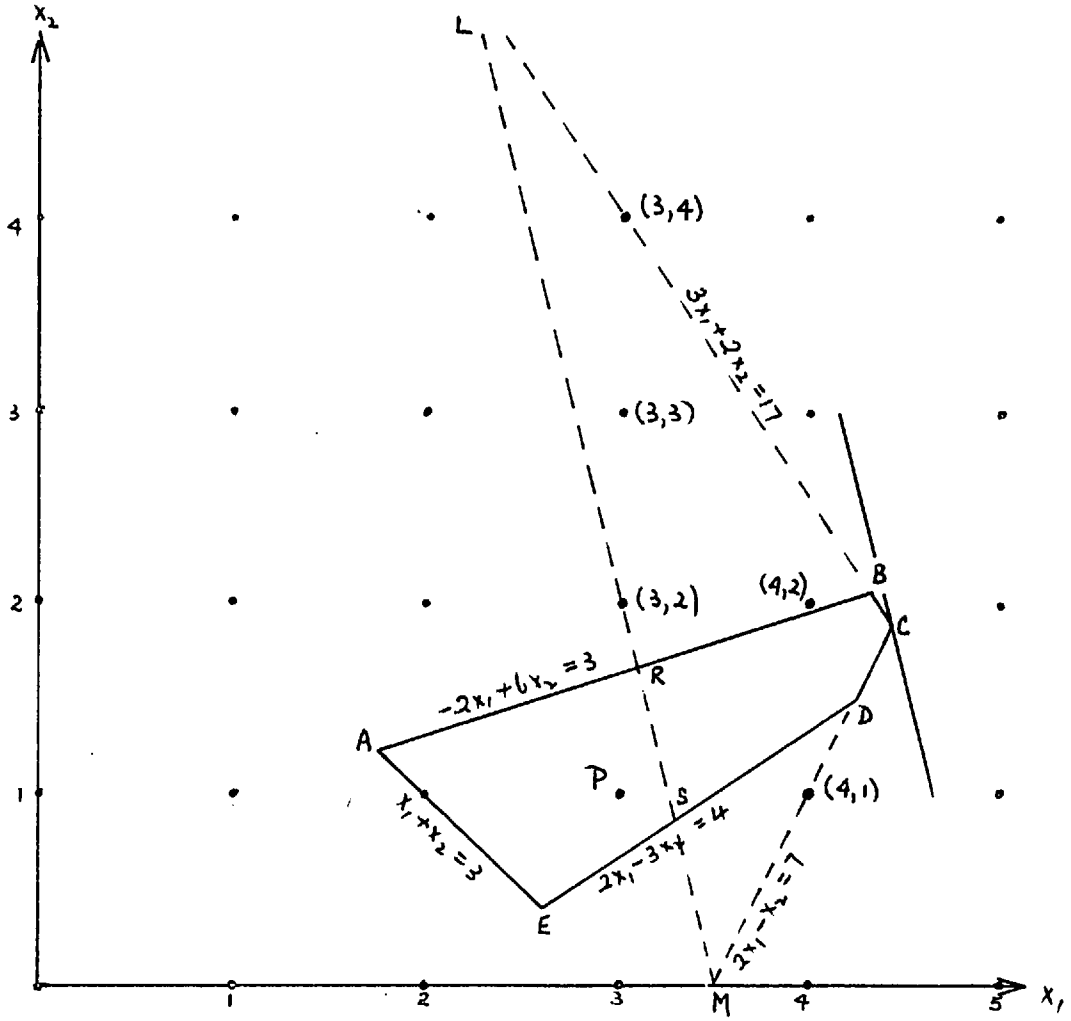


FIG. 5.7

In this graphic representation of the problem and solution, ABCDE represents the original convex enclosing the feasible space. C is the continuous optimal vertex and P the feasible optimum. LM is the functional constraint intersecting the constraints BC and DC, which define C, in L and M respectively. The region RBCDS, in which we are interested, is empty but, by our definition, the

isolated convex LCM has admitted the many infeasible points (3,4), (3,3), (3,2), (4,2) and (4,1). When it was realized that such a situation could arise the possibility of an excessive amount of feasibility testing had to be recognized. Consequently, our construction of this and similar problems led us eventually to take advantage of the branching variable selection process and to the retention of the bounds this produced. Fortunately, such a scheme appears successful in practice and excessive feasibility testing has not arisen in any of the problems which we have attempted. In the above problem, for example, during the progress from C to P the bounds obtained are successively $x_1 \leqslant 4$, $x_2 \leqslant 1$ and $x_1 \leqslant 3$. The isolated convex would produce $3 \leqslant x_1 \leqslant 4$, $0 \leqslant x_2 \leqslant 5$. Combining these two sets of information we obtain $3 \leqslant x_1 \leqslant 3$, $0 \leqslant x_2 \leqslant 1$ and the application of the bound contraction process to this set and the isolated convex reveals immediately that the functional constraint is violated.

Another source of difficulty is the determination of the vertices of the isolated convex. This determination is not in itself difficult, the vertices being obtained quite simply, but difficulties do arise if one or more of the vertices has coordinates with one or more negative components. Such a vertex is, of course, infeasible. In the early stages of development of this method all we did in order to remedy this situation was to set the negative valued components to zero and leave the others unaltered. In many situations this could lead to poor bounds, as illustrated in Figure 5.8. The convex defined by the constraints

is represented by CDEFG and the circled points are feasible lattice points.

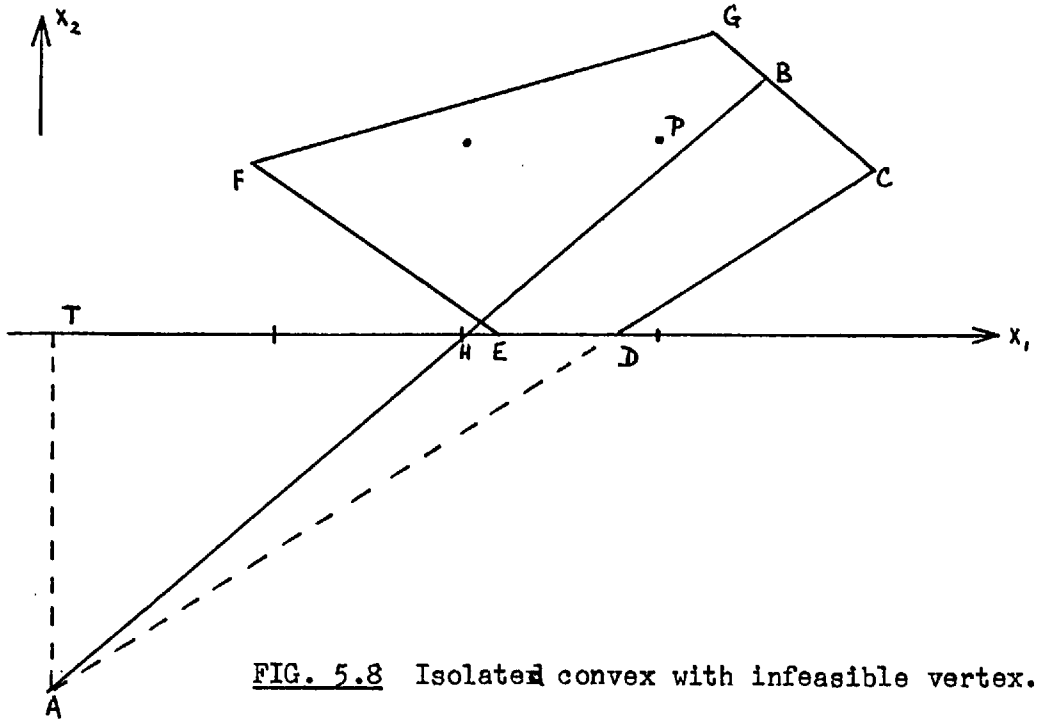


FIG. 5.8 Isolated convex with infeasible vertex.

Suppose C is the continuous optimal vertex and P the feasible optimum. AB represents the functional constraint defined so as to exclude P. It intersects the hyperplanes defining C in A, with co-ordinates (a_1, a_2) where $a_2 < 0$, and B.

B would provide a good bound on x_2 but the bounds obtained from A would be weak. Using the technique mentioned earlier for dealing with $a_2 < 0$, we produce the co-ordinates $(a_1, 0)$, the point T on the x_1 - axis. When constructing the bounds on x_1 and x_2 we then obtain $x_1 \geq \lceil a_1 \rceil + 1$ and $x_2 \geq 0$. Inspection of the figure indicates that, in fact, $x_1 \geq \lceil a_1 \rceil + 3$ and $x_2 \geq 0$.

Due to the possibility of such poor bounds resulting from the above method we have, in practice, further modified the technique

for dealing with a vertex having any negative co-ordinates.

In general, the isolated convex is represented by (n + 1) equations in n variables and the solutions of all square subsets of n equations produce the vertices. The solution of one of these subsets is, of course, unnecessary since the continuous vertex is already available from the continuous optimal tableau. If a vertex with a negative co-ordinate, say x_k, is produced then the particular subset of n equations associated with this vertex is augmented by the equation $x_k = 0$ and the new system solved n equations at a time. If many co-ordinates in the initial vertex are negative we choose for equating to zero that variable whose co-ordinate value is most negative. If this second operation should again produce negative co-ordinates we adopt the earlier remedy and set the appropriate values to zero.

In terms of Figure 5.8, the vertex A is obtained from the solution of the two equations representing BA and CD. The x_2 co-ordinate of A is negative so we determine, in turn, the intersections of $x_2 = 0$ and BA and $x_2 = 0$ and CD giving the points H and D respectively. Then the bound construction process will derive a better bound from H than it would have done from T.

This method has succeeded well in practice. A number of problems in the data sample presented these difficulties and the code was able to cope with them all.

5.6 Variants in Certain Areas of the Method

Two principle stages of the method, as already presented, are

susceptible to variation. These are (i) the determination of the vertices of the isolated convex and (ii) the action to be taken when a new feasible lattice point is found.

Although no details were given, it was indicated in Section 5.5 that the vertices of the isolated convex can be determined by solving all the square subsystems of n equations in n variables taken from a certain set of n + 1 equations in the same n variables. This amounts to inverting the matrix of coefficients associated with each vertex of the intersection of the functional constraint and the hyperplanes defining the continuous optimal vertex. Section 5.5 also pointed out difficulties that can arise with this technique. In practice, this is the technique we have adopted but there are at least two other possible methods of dealing with the isolated convex. From the theoretical point of view these are also more sophisticated since they use only principles common to linear programming. Furthermore, the difficult situations already referred to are automatically overcome.

The first approach would be to optimize the null vector over the new convex region defined by the functional constraint and the hyperplanes defining the continuous optimal vertex. If this was done for the situation illustrated in Figure 5.8 the points D and H would be determined instead of A, due to the condition $x_i \geq 0$ being implied by the simplex method.

The second approach would be to minimize and maximize each integer variable in turn over the new convex region defined by the functional constraint and the hyperplanes emanating from the

continuous optimal vertex. Again, due to the simplex method implications, this approach would not obtain A but would obtain H as a result of minimizing x_1.

Both of these approaches may introduce sophistication at the expense of computational effort; this aspect is further considered in Chapter VII.

After the production of the isolated convex R and the hyperrectangle L the region $L \frown R$ is searched for a feasible lattice point. If one is found it provides a better value of the objective function than any currently retained lattice point. In the general situation we may assume that hypotheses have been made en route to this point, i.e. the region $L \frown R$ has been partitioned in the manner described in Section 5.4. At the stage of finding a new feasible lattice point and hence defining a new functional constraint we can either (i) continue the examination of remaining hypotheses or (ii) recommence the process of determining the vertices of the new isolated convex to define a new R, use this to define a new L and by the bound contraction process seek a new lattice point in $L \frown R$. In order to adhere to the basic logic of the technique as a whole the second alternative was initially chosen. After applying the code to the sample of 30 problems the code was, however, modified to cater for the first alternative as well.

To conclude this section we mention one other part of the technique where variation is possible. The two variants pre-

sented above were regarded as major because they were concerned with variations within the logical structure of the technique. This final variant concerns the situation of having to select a variable, from among possibly many, in the event of $L \frown R$ being contracted to a region enclosing more than one member of the point - lattice $\mathcal{L}$. The code caters for the following three variants.

Of the variables associated with non-zero ranges select x_k

(i) where $(\eta_{k2} - \eta_{k1}) = \min (\eta_{j2} - \eta_{j1})$,

(ii) where $(\eta_{k2} - \eta_{k1}) = \max (\eta_{j2} - \eta_{j1})$,

(iii) where $(\eta_{k2} - \eta_{k1}) = \max (\eta_{j2} - \eta_{j1})$ and continue to use this variable until both its bounds are coincident,

for dichotomizing.

5.7 The Method in Practice

Before proceeding to discuss our general experience with the code we present a simple example for illustrative purposes.

$$\text{Maximize} \quad x_0 = 3x_1 + 4x_2 + x_3$$

$$\text{subject to} \quad x_1 + \tfrac{5}{2}x_2 + \tfrac{5}{2}x_3 \leqslant \tfrac{15}{2},$$

$$x_1 - x_2 \leqslant \tfrac{5}{2}$$

and x_1, x_2 and x_3 non-negative integers.

Introducing non-negative slack variables x_4 and x_5 this is, of course, seen to be a mixed integer problem but one which can be trivially transformed to a pure integer problem. However, such

a transformation is not necessary since the variables in the objective function are all integer variables and a functional constraint excluding some given lattice point can always be defined.

The solution below starts with the continuous optimal tableau in explicit form. As in the previous chapter, the rows representing the ranges on the integer variables are not recorded.

	1	$-x_5$	$-x_4$	$-x_3$	
x_0	35/2	1	2	4	
x_1	55/14	5/7	2/7	5/7	
x_2	10/7	-2/7	2/7	5/7	
x_3	0	0	0	-1	
x_4	0	0	-1	0	
x_5	0	-1	0	0	
s_1	-4/7	-2/7 *	2/7	5/7	Explore $x_2 \geq 2$.

	1	$-s_1$	$-x_4$	$-x_3$	
x_0	31/2	7/2	3	13/2	
x_1	5/2	5/2	1	5/2	
x_2	2	-1	0	0	
x_3	0	0	0	-1	
x_4	0	0	-1	0	
x_5	2	-7/2	-1	-5/2	
s_2	-1/2	-5/2 *	-1	-5/2	Explore $x_1 \leq 2$.

	1	$-s_2$	$-x_4$	$-x_3$	
x_0	74/5	7/5	8/5	3	
x_1	2	1	0	0	
x_2	11/5	-2/5	2/5	1	
x_3	0	0	0	-1	
x_4	0	0	-1	0	
x_5	27/10	-7/5	2/5	1	
s_3	-1/5	2/5	-2/5	-1 *	Explore $x_2 \leqslant 2$.

	1	$-s_2$	$-x_4$	$-s_3$	
x_0	71/5	13/5	2/5	3	
x_1	2	1	0	0	
x_2	2	0	0	1	
x_3	1/5	-2/5	2/5	-1	
x_4	0	0	-1	0	
x_5	5/2	-1	0	1	
s_4	-1/5	2/5	-2/5 *	1	Explore $x_3 \leqslant 0$.

	1	$-s_2$	$-s_4$	$-s_3$
x_0	14	3	1	4
x_1	2	1	0	0
x_2	2	0	0	1
x_3	0	0	1	0
x_4	1/2	-1	-5/2	-5/2
x_5	5/2	-1	0	1

This tableau is feasible and optimal.

The next stage is to define the isolated convex.

We first construct the functional constraint

$$x_0 = 3x_1 + 4x_2 + x_3 \geqslant 14 + 1.$$

Then the isolated convex is bounded by the hyperplanes

$$x_3 = 0 \; ; \qquad x_3 = 0 \qquad \text{(i)}$$

$$x_4 = 0 \; ; \qquad x_1 + \tfrac{5}{2}x_2 + \tfrac{5}{2}x_3 = \tfrac{15}{2} \qquad \text{(ii)}$$

$$x_5 = 0 \; ; \qquad x_1 - x_2 = \tfrac{5}{2} \qquad \text{(iii)}$$

$$x_0 = 15 \; ; \qquad 3x_1 + 4x_2 + x_3 = 15 \qquad \text{(iv)}$$

Equations (i), (ii) and (iii) define the continuous optimal vertex.

Equations (i), (ii) and (iv) define the point $(\frac{15}{7}, \frac{15}{7}, 0)$.

Equations (i), (iii) and (iv) define the point $(\frac{25}{7}, \frac{15}{14}, 0)$.

Equations (ii), (iii) and (iv) define the point $(\frac{195}{56}, \frac{55}{56}, \frac{5}{8})$.

So the ranges on the integer variables are

$$\frac{15}{7} \leqslant x_1 \leqslant \frac{55}{14},$$

$$\frac{55}{56} \leqslant x_2 \leqslant \frac{15}{7},$$

$$0 \leqslant x_3 \leqslant \frac{5}{8},$$

from which we obtain the L - region

$$3 \leqslant x_1 \leqslant 3,$$

$$1 \leqslant x_2 \leqslant 2,$$

$$0 \leqslant x_3 \leqslant 0.$$

The convex region R is defined by

$$x_1 + (5/2)\,x_2 + (5/2)\,x_3 \leqslant 15/2,$$

$$x_1 - x_2 \leqslant 5/2,$$

$$-x_3 \leqslant 0,$$

$$-3x_1 - 4x_2 - x_3 \leqslant -15$$

and the bound contraction process, defined in Section 5.2, is now used to contract $L \frown R$.

To make this next step easier to follow we recall, from Section 5.2, that

$$\delta_i = a_{io} - \sum_{j \in N} a_{ij} \phi_{ij} \qquad (5.26)$$

where the a_{ij}, $j \in \{0\} + N$, are the constant term and coefficients of the i-th constraint and ϕ_{ij} is the lower bound on x_j if $a_{ij} > 0$ and the upper bound on x_j if $a_{ij} < 0$.

If $a_{ik} > 0$, a revised upper bound on x_k is

$$x_k \leq \phi_{ik} + \frac{\delta_i}{a_{ik}} , \qquad (5.27)$$

where ϕ_{ik} is the existing lower bound,

while if $a_{ik} < 0$, a revised lower bound is

$$x_k \geq \phi_{ik} + \frac{\delta_i}{a_{ik}} , \qquad (5.28)$$

where ϕ_{ik} is the existing upper bound on x_k.

Now return to the first constraint defining R and use (5.26), (5.27) and (5.28) to compute

$$\delta_1 = \tfrac{15}{2} - (1)(3) - (\tfrac{5}{2})(1) - (\tfrac{5}{2})(0) = 2,$$

and so $x_1 \leq 3 + 2 = 5$,

$x_2 \leq 1 + \frac{2}{5/2} \Rightarrow x_2 \leq 1$, giving a new upper bound on x_2,

and $x_3 \leq 0 + \frac{2}{5/2}$.

This has changed the L - region to

$$3 \leq x_1 \leq 3 ,$$

$$1 \leq x_2 \leq 1 ,$$

$$0 \leq x_3 \leq 0 ,$$

and we continue to apply (5.26), (5.27) and (5.28) to the remaining constraints in R, using the current L - region. The computation proceeds as follows.

$$\delta_2 = \tfrac{5}{2} - (1)(3) + (1)(1) = \tfrac{1}{2},$$

$$x_1 \leqslant 3 + \frac{1/2}{1},$$

$$x_2 \geqslant 1 - \tfrac{1}{2}.$$

$$\delta_3 = 0,$$

$$x_3 \geqslant 0.$$

$$\delta_4 = -15 + (3)(3) + (4)(1) + (1)(0) = -2,$$

which indicates, as shown in Section 5.2, that a contradiction must occur (e.g. if we continued we would obtain $x_1 \geqslant 3 + \frac{-2}{-3} \Longrightarrow x_1 \geqslant 4$ which contradicts the current upper bound on x_1). This in turn implies that the isolated convex contains no member of the point - lattice, so that the current solution (14; 2,2,0, $\frac{1}{2}$, $\frac{5}{2}$) is optimal.

The above solution has been carried out in a straightforward manner appropriate for such an illustrative example. However, it does not fully represent all the detail of the computer code. Those points which have been omitted are discussed in Chapter VII.

Our experience with the code for this method is similar to that with the others. All the problems of the data sample have been attempted and, as with the branch and bound method, all were successfully solved. To make a worthwhile detailed comparison between the performance of the branch and bound method and that

of the method described in this chapter is a little difficult. Both codes behave in a similar manner up to the stage of finding the first feasible optimum. Thereafter the branch and bound code continues with pivot operations whilst the convex isolation and bound contraction method determines bounds on the integer variables before the repetitive bound contraction cycles are commenced.

Table 5.1 contains the appropriate performance figures for three variants of one version of the code. No variant behaved consistently better than any other but the second, which selects to dichotomize that variable associated with the largest range, performed slightly better than the other variants in most problems. For immediate reference the appropriate branch and bound performance figures have been included in Table 5.1. Except for problems 4, 5 and 6 the modified code has behaved very satisfactorily even on the basis of a one to one correspondence between a contraction cycle and a pivot operation. On such a basis the modified code clearly suffers because in a problem of m inequality constraints in n variables a contraction cycle involves $n(n + 1)$ multiplications, $n(n + 1)$ divisions and $(n + 1)^2$ additions whilst a branch and bound integer phase pivot operation requires $n(m + 2n + 1)$ multiplications, $(m + 2n + 1)$ divisions and $n(m + 2n + 1)$ additions. Problem 5, which represents the worst observed performance of the modified code relative to the branch and bound code, required an execution time of 1.53 seconds

PROBLEM	ITERATIONS TO FIRST FEASIBLE	INTEGER PHASE PERFORMANCE FIGURES							
		CONTRACTION CYCLES						BRANCH & BOUND ITERATIONS	
		VERSION 0							
		VARIANT 0		VARIANT 1		VARIANT 2			
		(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
1	13	0	19	0	22	0	25	0	55
2	7	0	0	0	0	0	0	0	12
3	11	0	13	0	11	0	13	0	13
4	12	11	11	13	13	15	15	1	4
5	7	132	240	23	215	336	528	8	13
6	11	0	90	0	77	0	94	0	6
7	7	132	240	23	215	336	528	8	13
8	11	0	90	0	77	0	94	0	6
9	9	0	5	0	5	0	5	0	4
10	7	0	0	0	0	0	0	0	5
11	9	0	0	0	0	0	0	0	1
12	13	0	0	0	0	0	0	0	21
13	12	8	12	40	44	52	56	21	33
14	33	0	0	0	0	0	0	0	18
15	6	0	0	0	0	0	0	0	1
16	14	0	2	0	2	0	2	0	15
17	28	0	0	0	0	0	0	0	1
18	67	12	15	13	16	13	16	4	12
19	58	0	61	0	58	0	59	0	38
20	200	0	0	0	0	0	0	0	0
21	203	0	0	0	0	0	0	0	0
22	203	0	0	0	0	0	0	0	0

TABLE 5.1 (continued on following page)

PROBLEM	ITERATIONS TO FIRST FEASIBLE	INTEGER PHASE PERFORMANCE FIGURES							
		CONTRACTION CYCLES						BRANCH & BOUND ITERATIONS	
		VERSION 0							
		VARIANT 0		VARIANT 1		VARIANT 2			
		(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
23	13	0	0	0	0	0	0	0	5
24	7	2	2	2	2	2	2	1	2
25	5	0	0	0	0	0	0	0	0
26	8	0	0	0	0	0	0	0	0
27	4	0	0	0	0	0	0	0	1
28	5	0	0	0	0	0	0	0	0
29	9	0	0	0	0	0	0	0	0
30	5	0	0	0	0	0	0	0	1

VERSION 0: Construct a new isolated convex each time a feasible lattice point is produced.

Footnotes (ii), (iii), (iv), (v) of Table 5.2 are also applicable here.

TABLE 5.1 Performances of three variants of the first version of the convex isolation and bound contraction code and an appropriate variant of the branch and bound code.

using the modified code, compared with 0.65 seconds using the branch and bound code. Apart from this and problem 6, the execution times with the modified code were considerably shorter than those with the branch and bound code. In general, it appeared that the execution

time per contraction cycle was small (2×10^{-3} seconds for a 10 x 6 problem tableau). However, the computation involved in determining the isolated convex region appeared to increase the mean cycle time by as much as a factor 10.

Table 5.2 gives the performance figures of three variants of the second version of the modified code. Clearly, only those problems in which the first feasible solution is not optimal are considered.

PROBLEM	INTEGER PHASE CONTRACTION CYCLES VERSION 1					
	VARIANT 0		VARIANT 1		VARIANT 2	
	(a)	(b)	(a)	(b)	(a)	(b)
4	11	23	13	28	15	35
5	132	147	23	193	336	344
7	132	147	23	193	336	344
13	8	13	22	22	20	20
18	12	12	13	13	13	13
24	2	2	2	2	2	2

(i) VERSION 1: Examines remaining hypotheses each time a feasible lattice point is produced.

(ii) VARIANT 0: Selects the smallest range for dichotomy.

(iii) VARIANT 1: Selects the largest range for dichotomy.

(iv) VARIANT 2: Selects the largest range for dichotomy and continues to dichotomize on this variable until its range becomes zero.

(v) Column (a) gives the number of operations between the first feasible solution and the optimum.

Column (b) gives the number of operations between the first feasible solution and the conclusion.

TABLE 5.2 Performances of three variants of the second version of the convex isolation and bound contraction code.

As represented by Table 5.2 VERSION 1 appears to have favoured the first variant, which gave the best performance on all six problems; it has proved to be considerably better than VERSION 0 in just problems 5 and 7.

Within our experience discussed above, the convex isolation and bound contraction method has behaved very satisfactorily and in many cases has performed most favourably in comparison with other methods. We conclude that it merits further consideration and investigation. At least, some of its ideas could perhaps be incorporated with advantage in the branch and bound method.

CHAPTER VI

OTHER METHODS.

In Chapter I we mentioned four classes of techniques capable of effectively solving integer programming problems. In the preceding chapters we have discussed in detail two of these. Although our practical experience has been confined to techniques within these two classes we consider it necessary, if only for the sake of completeness, to devote this chapter, the concluding chapter on techniques, to a presentation of some of the methods falling into the remaining two classes, namely, primal and partial enumeration methods. We describe two primal methods, one an all integer integer method and the other a mixed integer method. In addition to completing the four classes, we discuss an approximate method. Although, perhaps, approximate methods should not have a place in a thesis concerned, in the main, with exact methods for solving linear programming problems in integers it is considered that, from the point of view of practice, some mention should be made of them because various investigators of approximate methods have reported success in some cases. In particular, Cooper and Drebes [6], who developed an approximate method for solving fixed charge problems, state "..... the heuristic methods produce optimal solutions in well over 90 per cent of the several hundred problems investigated and very close to optimal (a few per cent) in the remaining cases". However, since this particular approximate method is designed for a certain type of problem we present the

method of Reiter and Rice [31], which has general application.

As stated above, we have had no practical experience with any of these methods so this chapter is entirely discursory.

6.1 Primal Methods

We commence this section with pure integer methods.

An optimal solution to an integer programming problem satisfies the two conditions of (i) optimality and (ii) feasibility. By our definition of feasibility this latter condition does, in fact, imply the two conditions (ii) non-negativity and (iii) integrality. The order in which these three conditions are attained may differ between different methods. For example, the All Integer Integer Method of Gomory, presented in Chapter II, generates a sequence of basic solutions which are optimal and integral but not non-negative until the attainment of the final solution. On the other hand, the other cutting plane methods of Gomory, also presented in Chapter II, proceed through sequences of solutions which are optimal and non-negative but non-integral before reaching the final solution. In contrast to these procedures, Young [39] has developed a primal algorithm which, prior to reaching the final solution, generates basic solutions which are integral, non-negative and non-optimal.

Let us briefly revise Gomory's All Integer Integer Method. Firstly the problem is to maximize

$$x_o = a_{oo} + \sum_{j \in N} a_{oj} (-t_j)$$

$$\text{subject to } x_i = a_{io} + \sum_{j \in N} a_{ij}\,(-t_j) \quad , \quad i \in I,$$

$$\text{and } x_i \geqslant 0 \text{ and integral.}$$

We note that (i) x_i, $i \in I = \{1,\ldots,m,\ldots,m+n\}$, consists of all the variables, including slack variables; (ii) t_j, $j \in N$, represents those x_i, $i \in I$, which are currently nonbasic; and (iii) the a_{ij} are also integral.

The adjoined cut as defined by Gomory is

$$s = \left[\frac{a_{ro}}{\lambda}\right] + \sum_{j \in N} \left[\frac{a_{rj}}{\lambda}\right](-t_j) + \left[\frac{1}{\lambda}\right](-t_r) \tag{6.1}$$

where the generator r is selected such that $a_{ro} < 0$ and at least one $a_{rj} < 0$, $j \in N$.

The dual simplex method is used so that λ is determined in order that

(i) the pivot element be -1

and (ii) λ be as small as possible.

The adjoined row then becomes the pivot row with pivot -1.

Gomory's finiteness proof is based on the fact that

$$\left[\frac{a_{ro}}{\lambda}\right] < 0.$$

Young [39] uses these ideas and presents what he calls a rudimentary primal algorithm the steps of which proceed thus.

(i) Select an entry variable t_j by the simplex criterion.

(ii) Select a source row by $\min_{a_{ij} > 0} (a_{io}/a_{ij})$.

(iii) Set $\lambda = a_{rj}$, the natural pivot.

(iv) Adjoin the cut (6.1) to the system. It can be assumed that $a_{rj} > 1$ and that x_r does not therefore appear in the cut.

(v) Perform the pivot step.

This procedure produces a cut which qualifies as a pivot row with pivot 1. However, proof of finiteness is difficult because of the possible occurrence of

$$\frac{a_{ro}}{a_{rj}} < 1 \tag{6.2}$$

for every permissible selection of a pivot column. When this situation occurs the cut has $\left[\frac{a_{ro}}{a_{rj}}\right] = 0$ and s enters the basis at zero level and after the pivot step x_j replaces s in the basis at zero level.

The primal algorithm of Young avoids, whenever possible, selection of an entry variable x_j which will lead to (6.2). If such selection is unavoidable, an entry variable implying (6.2) is selected and a cut having

$$s = \left[\frac{a_{ro}}{a_{rj}}\right] = 0 \tag{6.3}$$

is generated. Thus the major task associated with this algorithm is establishing a guarantee that (6.3) will occur for at most a finite sequence of successive cycles. Although the primal algorithm basically follows the rudimentary primal algorithm certain elaborations complicate and constrain the selection of the entry variable and of the source row. These serve to guarantee a finite algorithm and to avoid arbitrary restriction of choice beyond that to attain a finite algorithm.

The rudimentary algorithm is illustrated by the simple problem of maximizing

$$3x_1 + x_2$$

$$\text{subject to } 2x_1 + 3x_2 \leqslant 6$$

$$2x_1 - 3x_2 \leqslant 3 \ .$$

The solution process is represented by the following tableaux which are presented in the condensed form of Tucker. The source rows are indicated by arrows and the pivots by asterisks.

	1	$-x_1$	$-x_2$
x_0	0	-3	-1
→ x_3	6	2	3
x_4	3	2	-3
s_1	2	0	1*

If x_1 enters the basis then x_4 must exit and $g_v/a_{vj} = 3/2 > 1$. If x_2 enters the basis then x_3 must exit and $g_v/a_{vj} = 6/3 > 1$. So we choose x_2 arbitrarily and form a cut $s_1 \geqslant 0$.

	1	$-x_1$	$-s_1$
x_0	2	-3	1
→ x_3	0	2	-3
x_4	9	2	3
x_2	2	0	1
s_2	0	1*	-2

Add a cut $s_2 \geqslant 0$.

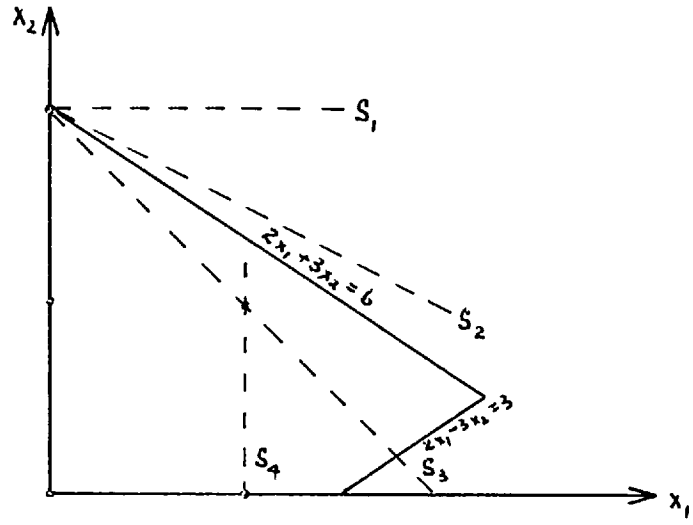
	1	$-s_2$	$-s_1$
x_0	2	3	-5
x_3	0	-2	1*
x_4	9	-2	7
x_2	2	0	1
x_1	0	1	-2

No cut

		1	$-s_2$	$-x_3$	
	x_0	2	-7	5	
	s_1	0	-2	1	
→	x_4	9	12	-7	Add a cut $s_3 \geq 0$.
	x_2	2	2	-1	
	x_1	0	-3	2	
	s_3	0	1*	-1	

		1	$-s_3$	$-x_3$	
	x_0	2	7	-2	
	s_1	0	2	-1	
→	x_4	9	-12	5	
	x_2	2	-2	1	Add a cut $s_4 \geq 0$.
	x_1	0	3	-1	
	s_2	0	1	-1	
	s_4	1	-3	1*	

	1	$-s_3$	$-s_4$	
x_0	4	1	2	
s_1	1	-1	1	Solution.
x_3	4	3	-5	
x_2	1	1	-1	
x_1	1	0	1	
s_2	1	-2	1	
x_3	1	-3	1	



$$s_1 = 2 - x_2 \geq 0$$

$$s_2 = -x_1 - 2x_2 + 4 \geq 0$$

$$s_3 = -x_1 - x_2 + 2 \geq 0$$

$$s_4 = -x_1 + 1 \geq 0$$

FIG. 6.1 Geometric representation of problem and its solution.

Whereas the previous algorithm is an all integer integer algorithm Harris [23] has developed a primal algorithm for solving mixed integer problems. Briefly, the linear programming problem, which is initially defined over a particular region of integer and continuous space, and whose objective function can be considered as having a direction of increasing "profit", is transformed to a problem defined over the integer space. Such a transformation introduces the property that over this space the profit increases from all directions towards one point. The object then is to progress towards this point from some arbitrary point in the integer space. This is achieved by the gradual and systematic contraction of the space of integer feasible solutions. Progress through the algorithm is controlled by the successive introduction of constraining hyperplanes within the integer space. The introduction of each such hyperplane effects the exclusion of a subspace of the current space of integers

and leads to the selection of the integer point, within the reduced space, from which the algorithm should proceed. The integer space is contracted step by step in this manner until the space enclosed by these, and initially defined, hyperplanes is empty. The attainment of this situation completes the algorithm.

Consider the feasible integer space to be embedded in a continuous space and bounded by a feasible convex boundary. If a linear programme is defined over the continuous space then there is an associated corresponding continuous linear programme at each point of the integer space. These programmes will have the same matrix elements but the vector representing the right hand side of the constraints and the objective will change. Each such linear programme, which is feasible, has an optimal value of its objective function. This value will be defined to be the "profit" of the appropriate point in integer space. An integer point whose continuous linear programme is infeasible will have no associated profit. This point would be outside the convex feasible boundary of the integer space.

The basic principle of the algorithm is best introduced by considering the geometrical representation of a simple problem. The purpose here is to indicate that the problem can be transformed to a situation in which the profit increases towards a focus from any direction rather than the usual situation where the profit continues to increase in one direction. This is brought about by projecting the space common to the objective and the continuous

feasible space onto the integer space. To illustrate, consider Figure 6.2 which represents a three dimensional convex defined over the integer space x_1, x_2 and the continuous space x_3. The direction x_3 is regarded to be very profitable. The unbracketed numbers in the figure denote the profits at the corresponding points. At the point with maximum x_3 the profit is 8. However, this value is really of no concern since the only points of interest are those such that $(x_1,x_2) = (0,0),(1,0),(1,1),(0,1)$. Assume that the projection onto the integer space of the point with maximum x_3 occurs at $x_1 < \frac{1}{2}$, $x_2 < \frac{1}{2}$ then the closest integer point to it has profit 3 and a unit change in either variable will produce the profit 1. Thus the point with profit 3 is a local optimum. However, the diagonally opposite integer point produces a profit of 5. Inspection of the integer space indicates that only the four points $(0,0),(1,0),(1,1)$ and $(0,1)$ need be considered, these have associated profits of 3, 1, 5 and 1 respectively. By considering only these integer points nothing more can be said at this stage, however, if one allows points other than the integer points to have associated profits, then the inclusion of these will permit lines of equal profit to be produced. Such lines will be called iso-profit contours. Figure 6.3 represents the iso-profit contours of this particular example superimposed on the integer space x_1, x_2 and extended into the surrounding area. Progression towards the innermost contour from any direction results in an increase in profit.

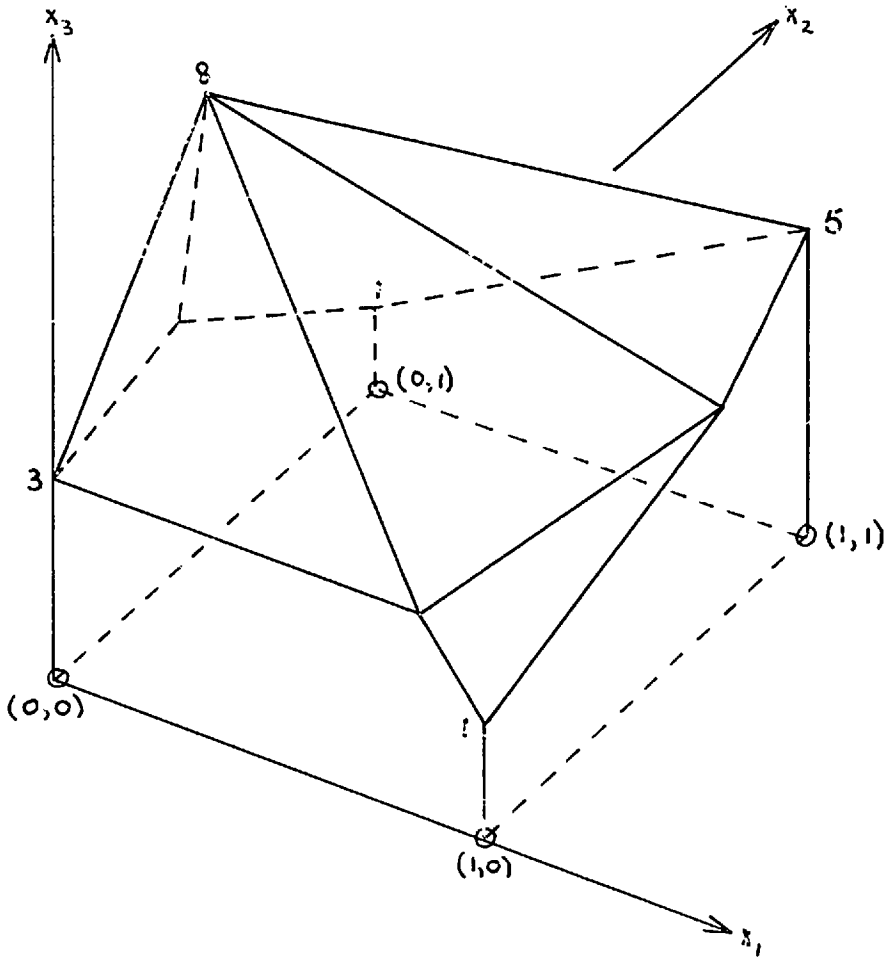


FIG. 6.2

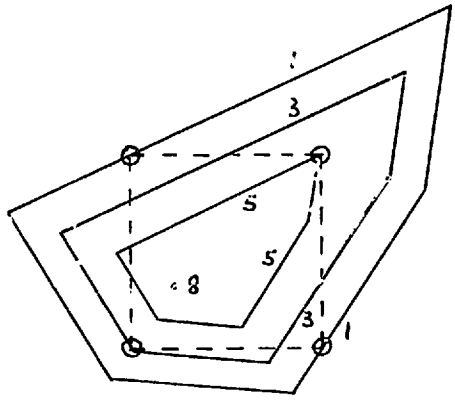


FIG. 6.3

The point with profit 8 may be regarded as a focus towards which the profit will increase from all directions. In order to show that an iso-profit contour is simply a convex boundary of a feasible region smaller than the initial feasible region consider the feasible mixed integer space to be bounded by very small facets then an ellipsoid may be taken as representative of such a space. This is illustrated by Figure 6.4.

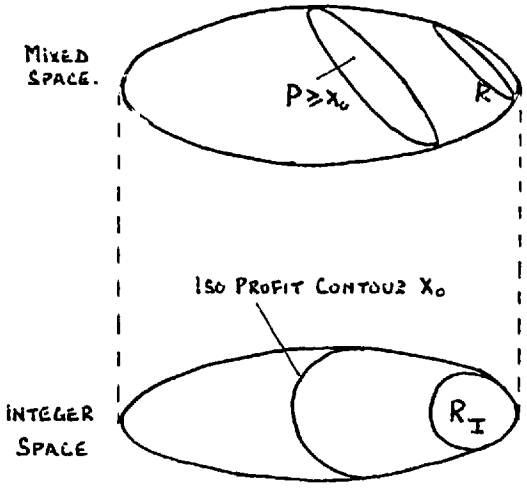


FIG. 6.4

Let $P(x_i)$ be the profit associated with a point $\bar{x}_i$, $\bar{x}_i$ a vector. Introduce the linear profit restriction $P \geq x_o$ such that it divides the feasible mixed integer space E into two subsets R and E-R. The subset R is the set of points $\bar{x}_i$ such that $P(x_i) \geq x_o$, $\bar{x}_i \in R$. Projection of R onto the integer space produces a region R_I containing integer points having profits exceeding x_o. Points lying within the original feasible boundary in integer space but lying outside R_I have profits less then x_o. The new boundary in the integer space is in fact the iso-profit

contour x_o because points on this boundary have profits equal to x_o. Changing the value of x_o will clearly produce a family of iso-profit contours. As x_o increases the boundary of the feasible region containing integer points whose profits exceed x_o converges to its limit.

As mentioned earlier, to each point in integer space there corresponds a continuous linear programme and each of these linear programmes which is feasible has an optimal value of its objective function. Clearly, a process is required whereby one may proceed from one point to another in the integer space in such a way that the value of the objective function improves and the solution remains or becomes feasible. Initially, the aim is to find a feasible point in integer space and to determine the value of the objective function at this point. The starting point for such a process could be arbitrarily selected in which case the origin is the natural choice. If the point is infeasible (in which case, during attempted optimization at this point, progress through the simplex algorithm will be halted when a row is obtained in which the coefficients of the continuous variables are positive but the constant term of the constraint is negative) the situation provides a linear restriction on the integer variables. It is the first linear boundary of the feasible integer space. Boundaries are introduced such that at least the current point is excluded from the convex region of feasible solutions. This linear restriction on the integer variables is used temporarily as a control function of the integer problem. Its value must be

increased to become non-negative. Once this integer constraint attains feasibility so does the previously infeasible continuous constraint. This feasible integer constraint is retained to prevent re-entry into this area of the infeasible integer space. Once a feasible integer point is available the continuous linear programme at this point is optimized.

The value of the objective function provides a means of defining a further constraint on the problem. All continuous problems throughout the feasible integer space could be solved so producing a set of optima $x_o(i)$. Then the solution to the original problem produces the optimum

$$x_o = \max_i x_o(i).$$

Once an $x_o(k)$ is obtained, the restriction

$$x_o \geqslant x_o(k)$$

is introduced and implies that provided feasibility is retained a current solution producing an x_o smaller than $x_o(k)$ will not be accepted. During progress through the algorithm x_o^{max} denotes the highest profit attained up to the stage under consideration. If this restriction had a negative value the continuous problem would be infeasible because the profit restriction would have no negative continuous variable coefficient. The integer constraint preventing this will be the linear segment of the x_o^{max} iso-profit contour to be added to the integer problem for use initially as control. Now by changing the value of x_o^{max}, the value of this restriction will change producing a family of iso-profit contours.

Increasing the value of this restriction by proceeding to a new integer point will increase the value of the objective function.

Successive introduction of boundaries causes the feasible space to converge to the optimal solution. The process does in fact continue until the existing space bounded by feasibility boundaries and iso-profit contours has been contracted to the situation where no integer point is contained.

It has been indicated that the solution space of integers is gradually diminished by the introduction of two types of boundaries, one to attain feasibility, the other to produce new profit. Furthermore, it was stated that the introduction of such boundaries caused the exclusion of an integer space point and necessitated the selection of another point within the remaining space. The method of effecting these steps will now be explained.

Firstly, let us slightly modify our standard notation so as to make presentation of this method somewhat simpler. Let us denote the integer variables by higher case letters and the continuous variables by lower case letters. Then let us consider the situation when infeasibility occurs. In such a case some row of the tableau would be defined

$$\sum_{i \in S_I} a_{ri} X_i + \sum_{j \in S} a_{rj} x_j = a_{ro} \qquad (6.4)$$

where S_I and S are sets of indices defining integer space and continuous space respectively, $a_{ro} < 0$ and $a_{rj} \geq 0$ with at least one $a_{rj} > 0$. Then the feasibility constraint to be added to the

problem is

$$\sum_{i \in S_I} a_{ri} X_i - f = a_{ro} \tag{6.5}$$

where f is the new feasibility variable.

Two circumstances can arise whereby a profit restriction has to be added to the problem. The first arises when optimization at a point is about to produce a value of the objective function which is below the current value of x_o^{max}. The second arises when optimization has been successful and x_o^{max} has been reset to the new high value. Suppose that

$$\sum_{i \in S_I} a_{oi} X_i + \sum_{j \in S} a_{oj} x_j = x_o - x_o^{o} \tag{6.6}$$

represents the unsatisfied profit restriction that would have resulted from the succeeding iteration in the first case and the current objective function of the second case. Then the profit constraint to be added to the problem is

$$\sum_{i \in S_I} a_{oi} X_i - p + x_o^{o} = 0. \tag{6.7}$$

The introduction of a constraint of this type causes the exclusion of a subspace of the current solution space and the problem now is to find an integer lying within the reduced space or to establish that none exists. The procedure for carrying out this step will now be given. This will be followed by a discussion on the development of such a procedure.

The situation now is that a feasibility or profit constraint has been added to the tableau. The integer linear programme is approached as if it were an ordinary linear programme and a pivot

row r is selected on the basis

$$\frac{a_{ro}}{a_{rk}} = \min_{i} \frac{a_{io}}{a_{ik}} ,$$

where the column k has been selected on the basis

$$a_{ok} = \min_{j \in S_I'} a_{oj} ,$$

in which $S_I' \subset S_I$ is the set of indices corresponding to the integer variables which are currently nonbasic. Now the row

$$\sum_{j \in S_I'} a_{rj} X_j - y = a_{ro} , \tag{6.8}$$

where y is the basic variable, is used to construct the restriction

$$\sum_{j \in S_I'} \left[\frac{a_{rj}}{a_{rk}}\right] X_j - Y = \left[\frac{a_{ro}}{a_{rk}}\right], \tag{6.9}$$

which is then added to the tableau as the new pivot row with a pivotal element of unity. The primal simplex algorithm can now be used to find an integer point within the solution space currently defined, or failing this, optimality of the present integer point will be established.

The development of this restriction will now be discussed.

Clearly, once an integer point has been selected it will not be known whether such a point satisfies the integer constraints of profit and feasibility until the tableau integer columns have been combined with the column of constant terms to produce the column of constant terms of the continuous problem at that point. At the same time, this will produce the constant terms of these constraints. Now, at the stage under consideration, the last integer point examined satisfied all the constraints but

the introduction of a profit or feasibility constraint has since disqualified it. While retaining feasibility, this disqualifying constraint must be increased in value until it is satisfied by the new integer point. Consequently, it may be regarded as a control row for maximizing. The problem now resembles an ordinary linear programme except that movement is restricted to integer points. Movement in an ordinary linear programme is restricted to the vertices of a polytope. By introducing a new restriction to the integer problem a vertex can be created at an integer point. An ordinary simplex iteration will then effect movement to the integer point. If the new restriction does not exclude any feasible integer point then its inclusion cannot affect the integer point solution to this problem. Several such restrictions will serve to create a vertex at the solution integer point, and optimality will be established in the usual way after a number of ordinary simplex iterations.

Now each new restriction will be used as a pivot row so its elements will dictate the multiples of the pivot column to be combined with the other integer columns and with the column of constant terms. If it can be so arranged that this new row has a pivotal element of 1 and other elements integral then only integral multiples of the necessary columns can be formed ensuring that any changes to the values of the integer variables will be integral. The construction of a restriction

satisfying this requirement and the one that no integer point must be removed will now be discussed by means of an example. Suppose an ordinary linear programme approach to the integer linear programme produces a pivot row

$$1.2\,Y_1 - .7\,Y_2 + 3.9\,Y_3 = 2.3$$

where Y_i are integer variables and the pivot element is the coefficient of Y_1. Proceeding with the pivotal operation would indicate that Y_1 is about to increase from 0 to $2.3/1.2 = 1.916$. Since Y_1 is an integer variable the new restriction to be constructed should limit Y_1 to $\lfloor 1.916 \rfloor = 1$. Thus far, two factors about the new restriction are known, the pivot and the constant term. The new restriction can therefore be constructed in the following manner. Divide the selected row by the pivot element to obtain

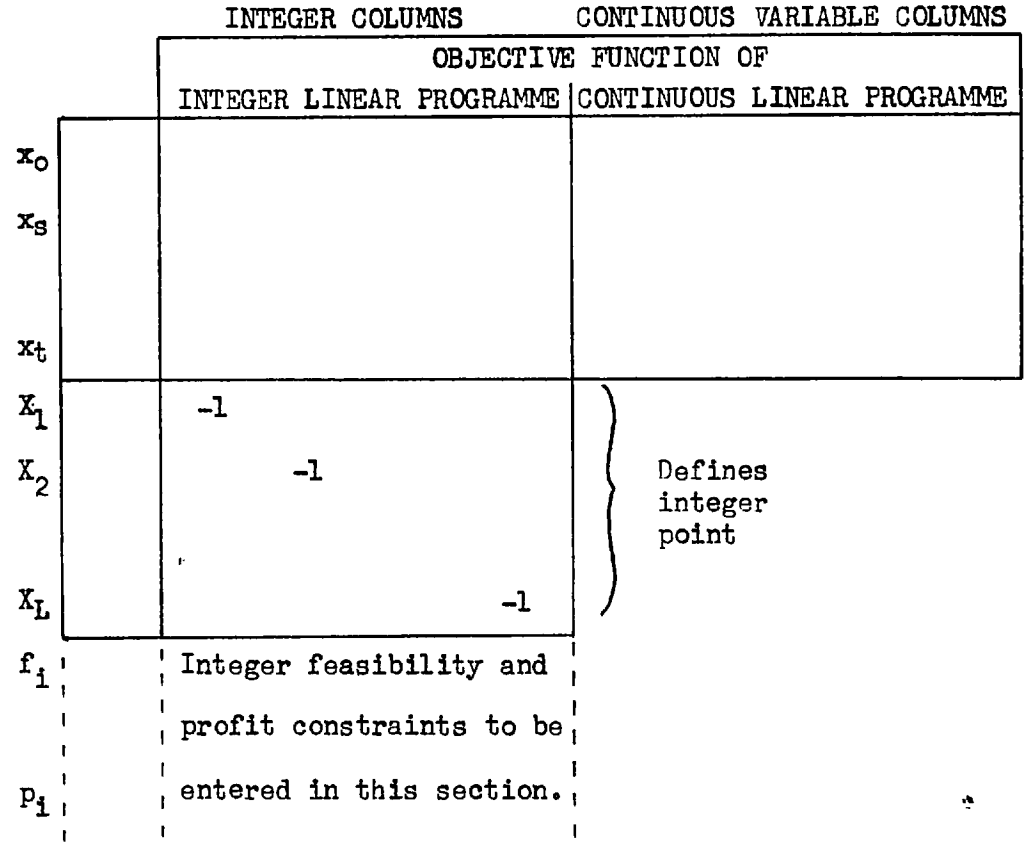
$$Y_1 - .583Y_2 + 3.25Y_3 = 1.916.$$

By integerizing the coefficients this can be made less restrictive and an integer combination of integers equal to 1.916 would be produced so, clearly, the whole row could be integerized to give

$$Y_1 - Y_2 + 3Y_3 = 1\,,$$

which is the required restriction with the unit pivot element being the coefficient of Y_1. Finally, since pivoting in this way never removes from the basis a variable which will be later required this pivot row may be discarded after use.

In conclusion to this section it will be helpful to present the steps of the computation and a worked example. Every possible situation can, by combining integral multiples of the integer columns, be represented within a tableau structure which, since it differs from our familiar tableau, is presented here



Commencing at the origin of integer space, the computational flow would proceed as follows.

(i) Attempt to find the profit at the current integer point by solving the continuous linear programme at this point. If infeasibility occurs proceed from step (ii). If it is found that the value of the objective function can be

reduced below x_o^{max} proceed from step (iv). If optimization has been successful reset x_o^{max} and proceed from step (iv).

(ii) Annexe the feasibility constraint (6.5) to the tableau and continue to step (iii).

(iii) Employ the integer subsection of the tableau to obtain a pivot row (6.8) and from this construct the integer restriction (6.9) and annexe this to the tableau as the new pivot row. Perform the pivot operation. If infeasible then the mixed integer problem is infeasible so the process must terminate. If no point is produced then the solution is complete and is provided by the current integer point. If a point is found drop the pivot row from the tableau and proceed from (i) with the new integer point.

(iv) Construct a profit constraint from the appropriate tableau row, annexe to the tableau and proceed from (iii).

Now consider the problem of maximizing

$$11X_1 + 2X_2 + 6x_3 + x_4$$

subject to

$$6X_1 + 4X_2 + x_3 - 3x_4 \leqslant 6,$$

$$X_1 + 3X_2 + 2x_3 + x_4 \leqslant 3,$$

$$X_1, X_2 \geqslant 0 \text{ and integral},$$

$$x_3, x_4 \geqslant 0.$$

The initial tableau at the origin of integer space is

	1	$-X_1$	$-X_2$	$-x_3$	$-x_4$
x_o	0	-11	-2	-6	-1
x_5	6	6	4	1	-3
x_6	3	1	3	2*	1
X_1	0	-1	0		
X_2	0	0	-1		

First we solve the continuous linear programme to give

	1	$-X_1$	$-X_2$	$-x_3$	$-x_4$
x_o	9	-8	7	3	2
x_5	9/2	11/2	5/2	-1/2	-7/2
x_3	3/2	1/2	3/2	1/2	1/2
X_1	0	-1	0		
X_2	0	0	-1		

So the linear programme at the point (0,0) has an optimal value of 9. Following the algorithm we set $x_o^{max} = 9$ to give the objective

$$x_o - x_o^{max} = 0 - 8(-X_1) + 7(-X_2) + 3(-x_3) + 2(-x_4)$$

and the profit restriction

$$p_1 = 0 - 8(-X_1) + 7(-X_2).$$

Increasing the value of this restriction by moving to a new integer point will increase the value of the objective function. With the above two modifications we approach the integer linear programme as if it were an ordinary linear programme. We do not

consider the row representing x_5 because a continuous variable has a negative coefficient so we select

$$x_3 = \frac{3}{2} + \frac{1}{2}(-X_1) + \frac{3}{2}(-X_2)$$

as the pivot row with pivot $\frac{1}{2}$. We integer modify this row and add this and the profit restriction to the tableau which now becomes

	1	$-X_1$	$-X_2$	$-x_6$	$-x_4$	
$x_o - x_o^{max}$	0	-8	7	3	2	
x_5	9/2	11/2	5/2	-1/2	-7/2	
x_3	3/2	1/2	3/2	1/2	1/2	
X_1	0	-1	0			
X_2	0	0	-1			
p_1	0	-8	7			Control.
X_3	3	1*	3			Pivot.

Performing the indicated pivot operation and discarding the pivot row produces

	1	$-X_3$	$-X_2$	$-x_6$	$-x_4$
x_o	33	8	31	3	2
x_5	-12	-11/2	-14	-1/2	-7/2*
x_3	0	-1/2	0	1/2	1/2
X_1	3	1	3		
X_2	0	0	-1		
$x_o^{max} + p_1$	33	8	31		

giving the new integer point (3,0). Application of the

dual, which retains optimality but decreases the objective gives

	1	$-X_3$	$-X_2$	$-x_6$	$-x_5$
x_o	183/7	34/7	23	19/7	4/7
x_4	24/7	11/7	4	1/7	-2/7
x_3	-12/7	-9/7	-2	3/7	1/7
X_1	3	1	3		
X_2	0	0	1		
$x_o^{max}+p_1$	33	8	31		

This tableau indicates the situation defined by (6.4), namely, there exists some $a_{ro} < 0$ with $a_{rj} \geq 0$ and at least one $a_{rj} > 0$ for $j \in S$ where S is the set of indices defining continuous space. Hence we must extract from the x_3 row the feasibility constraint

$$x_3 = -\frac{12}{7} - \frac{9}{7}(-X_3) - 2\,(-X_2)\,.$$

Now, as before, we select the second column by

$$a_{ok} = \min_{j \in S_I'} a_{oj}$$

and use

$$\frac{a_{ro}}{a_{rk}} = \min_{i} \frac{a_{io}}{a_{ik}}$$

to determine the pivot row X_1. The row x_4 could not be a candidate because of a negative coefficient associated with a continuous variable. We add the feasibility constraint and the integer modified pivot row to the tableau which now becomes

	1	$-X_3$	$-X_2$	$-x_6$	$-x_5$	
x_o $-x_o^{max}$	120/7	34/7	23	19/7	4/7	
x_4	24/7	11/7	4	1/7	-2/7	
x_3	-12/7	-9/7	-2	3/7	1/7	
X_1	3	1	3			
X_2	0	0	1			
p_1	24	8	31			
f_1	-12/7	-9/7	-2	Control.		
P_1	3	1*	3	Pivot.		

Performing the indicated pivot operation and dropping the pivot row produces

	1	$-P_1$	$-X_2$	$-x_6$	$-x_5$
x_o	81/7	-34/7	59/7	19/7	4/7
x_4	-9/7	-11/7	-5/7	11/7	-2/7
x_3	15/7	9/7	13/7	3/7	1/7
X_1	0	-1	0		
X_2	0	0	-1		
x_o^{max} p_1	9	-8	7		
f_1	15/7	9/7	13/7		

Since $x_o^{max} = 9$ the p_1 value is now zero which implies that the present integer point lies on the x_o^{max} iso-profit contour. The distance from this boundary must be increased. We choose the second column as pivot column by the usual process and then select the f_1 row

to provide the constraint

$$F_1 = \left[\frac{15/7}{9/7}\right] + \left[\frac{9/7}{9/7}\right](-P_1) + \left[\frac{13/7}{9/7}\right](-X_2)$$

$$= 1 + 1\,(-P_1) + 1\,(-X_2)\,,$$

with the coefficient of $(-P_1)$ being the pivot. Execution of the pivot operation produces the tableau

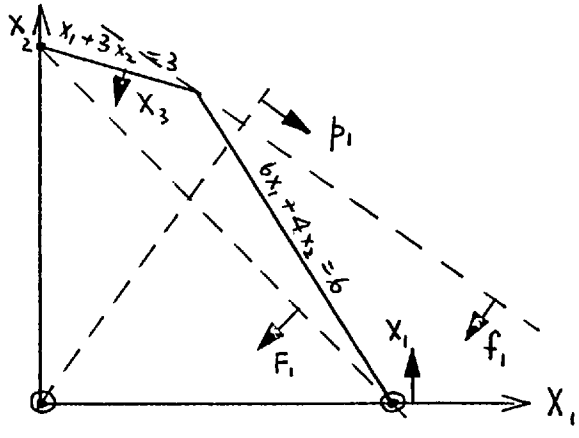
	1	$-F_1$	$-X_2$	$-x_6$	$-x_5$
x_o	115/7	34/7	93/7	19/7	4/7
x_4	2/7	11/7	6/7	1/7	-2/7
x_3	6/7	-9/7	4/7	3/7	1/7
X_1	1	1	1		
X_2	0	0	-1		
x_o^{max} p_1	17	8	15		
f_1	6/7	-9/7	4/7		

Now set $x_o^{max} = \frac{115}{7}$ and form the profit constraint

$$p_2 = 0 + \frac{34}{7}(-F_1) + \frac{93}{7}(-X_2).$$

This is already optimal so the solution is complete.

The integer space for the above example is geometrically represented in Figure 6.5.



p_1: $8X_1 - 7X_2 \geq 0$

X_3: $3 - X_1 - 3X_2 \geq 0$

f_1: $-12/7 + 9/7X_3 + 2X_2 \geq 0$

$\Longrightarrow 9X_1 + 13X_2 - 15 \leq 0$

P_1: $3 - X_3 - 3X_2 \geq 0 \Longrightarrow X_1 \geq 0$

F_1: $1 - P_1 - X_2 \geq 0 \Longrightarrow X_1 + X_2 \leq 1$

FIG. 6.5

6.2 Partial Enumeration

We present a brief description of an enumerative technique and illustrate it by means of a simple example.

We assume the values of all variables to be either 0 or 1, that all constraints are of the form

$$y_i = a_{io} + \sum_{j \in N} a_{ij} x_j \geq 0$$

and that the objective function to be maximized is

$$y_o = a_{oo} + \sum_{j \in N} a_{oj} x_j .$$

We may assume that all $a_{oj} \leq 0$, $j \in N$, because if some $a_{oi} > 0$, $i \in S \subset N$ then we can replace x_i, $i \in S$, by $1 - x_i$. At any stage in the computation we have a current best known solution with its corresponding value of y_o, say y_{oc}. We also have a set of subproblems in each of which the variables are divided into the following three sets

S_o = the set of variables to which the value zero is assigned,

S_1 = the set of variables to which the value one is assigned, and

S_2 = the set of unassigned variables provisionally set equal to zero. Initially, all variables $\in S_2$, that is the solution is $(0, a_{io})$.

Suppose the current values of $Y = (y_o, y_1, \ldots.)$ are $V = (v_o, v_1, \ldots.)$. Suppose the maximum possible values of $y_1, y_2, \ldots.$ obtainable by changing the set S_2 are $u_1, u_2, \ldots.$ then

$$u_i = v_i + \sum_j \max (0, a_{ij})$$

where j ranges over all values such that x_j is in S_2.

Initially, we may set $y_{oc} = -\infty$ and there exists one subproblem with all variables in S_2. Then the enumerative technique would proceed in the following manner.

(i) Take a subproblem from the list. If $v_o \leq y_{oc}$ then abandon it. Otherwise, if all $v_i \geq 0$ for $i \geq 1$ we have a new best known solution.

(ii) Otherwise, if $a_{oj} \leq y_{oc} - v_o$ for any j, put x_j in S_o and revise the value of the u_i. (x_j can only be zero otherwise the functional would be decreased).

(iii) If any $u_i < 0$ the subproblem cannot have a feasible solution (if the maximum value of y_i is negative then y_i remains negative) and should be dropped. If $u_i + a_{ij} < 0$ for some i we may put x_j in S_o and revise the u_i accordingly since the subproblem would be infeasible with $x_j = 1$. ($u_i + a_{ij} < 0, (u_i > 0)$, implies that $a_{ij} < -u_i$ so if it is active, that is, if $x_j = 1$, then max. $y_i < 0$ which indicates infeasibility).

(iv) If $u_i - a_{ij} < 0$ put x_j in S_1, since the problem is infeasible with $x_j = 0$, ($a_{ij} > u_i > 0$).

(v) If no x_j can be assigned to S_1 in step (iv) then a branch must be made. Take some variable in S_2 and assign it to S_1, but at the same time set up a new subproblem that is the same as the current one except that this variable is assigned to S_o.

The following example will illustrate the technique.

Maximize $x_1 - x_2 + x_3$

subject to

$$x_1 + x_2 + x_3 \leqslant 1,$$

$$2x_2 + x_3 \leqslant 7$$

and $x_i = 0$ or 1.

To start put $x_1 = 1 - x_1'$

and $x_3 = 1 - x_3'$

and rewrite the problem as follows.

Maximize $y_o = 2 - x_1' - x_2 - x_3'$

subject to

$$y_1 = -1 + x_1' - x_2 + x_3'$$

and $y_2 = 6 \quad -2x_2 + x_3'$.

Solution 1. $x_1' = 0, x_2 = 0, x_3' = 0$ all in S_2.

$v_o = 2, v_1 = -1, v_2 = 6.$

$v_o \not\leqslant y_{oc}\ (= -\infty),$

$v_i \not\geqslant 0.$

$y_{oc} - v_o = -\infty.$

$$\left.\begin{aligned} u_1 &= v_1 + \{\max(0,1) + \max(0,-1) + \max(0,1)\} = v_1 + 2 = 1 \\ u_2 &= v_2 + \{\max(0,0) + \max(0,-2) + \max(0,1)\} = v_2 + 1 = 7 \end{aligned}\right\} u_i > 0.$$

$$\left.\begin{aligned} u_1 + a_{11} &= 2, & u_1 + a_{12} &= 0, & u_1 + a_{13} &= 2, \\ u_2 + a_{21} &= 7, & u_2 + a_{22} &= 5, & u_2 + a_{23} &= 8. \end{aligned}\right\} u_i + a_{ij} \not< 0.$$

$$\left.\begin{aligned} u_1 - a_{11} &= 0, & u_1 - a_{12} &= 2, & u_1 - a_{13} &= 0, \\ u_2 - a_{21} &= 7, & u_2 - a_{22} &= 9, & u_2 - a_{23} &= 6. \end{aligned}\right\} u_i - a_{ij} \not< 0 \text{ so branch.}$$

Take x_3 from S_2 and place in S_1; i.e. set $x_3 = 1$. (a)

Also, take x_3 from S_2 and place in S_o and list this situation. (b)

Select subproblem (a).

$$x_1' = 0, \quad x_2 = 0, \quad x_3' = 1; \quad v_o = 1, \; v_1 = 0, \; v_2 = 7.$$
$$S_2 \qquad S_2 \qquad S_1$$

$v_i \geqslant 0$ so we have a new solution and $y_{oc} = 1$.

Now select subproblem (b).

$$x_1' = 0, \quad x_2 = 0, \quad x_3' = 0; \quad v_o = 2, \; v_1 = -1, \; v_2 = 6.$$
$$S_2 \qquad S_2 \qquad S_o$$

$v_o \not\leqslant y_{oc}$, $v_i \not\geqslant 0$.

$y_{oc} - v_o = 1 - 2 = -1$ (j now 1 or 2).

$a_{o2} = -1 = y_{oc} - v_o$ so put x_2 in S_o (j now 1).

$$\left.\begin{aligned} u_1 &= v_1 + \max(0,1) = v_1 + 1 = 0, \\ u_2 &= v_2 + \max(0) = v_2 = 6. \end{aligned}\right\} \; u_i \not< 0.$$

$$\left.\begin{aligned} u_1 + a_{11} &= 1, \\ u_2 + a_{21} &= 6. \end{aligned}\right\} \; u_i + a_{ij} \not< 0.$$

$$\left.\begin{aligned} u_1 - a_{11} &= -1, \\ u_2 - a_{21} &= 6. \end{aligned}\right\} \; u_i - a_{ij} < 0 \text{ when } i = 1 \text{ so put } x_1 \text{ in } S_1.$$

$$x_1' = 1, \quad x_2 = 0, \quad x_3' = 0; \quad v_o = 1.$$

The list is now empty so the solution is

$$x_1' = 0, \quad x_2 = 0, \quad x_3' = 1, \; y_o = 1, \; y_1 = 0, \; y_2 = 7.$$

Reverting to the original variables the solution is

$$\underline{\underline{x_1 = 1, \quad x_2 = 0, \quad x_3 = 0.}}$$

6.3 An Approximate Method

To remain within the general context of this thesis, the method to be discussed here is a particular application of the general method developed by Reiter and Rice [31] for approximating the solution of mixed integer nonconcave programming problems in bounded variables.

From the given hyperrectangle defined by the bounded variables a point is sampled at random. If this point lies outside the convex solution space the method enables one to progress from this point to a feasible lattice point. Once a feasible lattice point has been determined the objective function is employed in order to define a direction of search for improved lattice points. This direction, and later a succession of modified directions, is followed until a constraining hyperplane prohibits further progress from the last encountered lattice point, which is then said to be locally optimal. The sequence of random selection, attaining feasibility, and climbing the lattice to a local optimum is repeated and the complete process continued until many local optima are obtained. A search among these produces the best, which is then an approximation to the global optimum. This approach is made more sophisticated by the introduction of a learning procedure which restricts the area of search for feasible lattice points.

Suppose an integer programming problem is defined as the maximization of

$$x_o = \sum_{j \in N} a_{oj} x_j , \qquad (6.10)$$

subject to

$$\sum_{j \in N} a_{ij} x_j \leqslant a_{io}, \quad i \in M, \tag{6.11}$$

and $$u_j \leqslant x_j \leqslant v_j, \quad j \in N. \tag{6.12}$$

Then the details of the solution process outlined above are as follows.

(a) Choose a random starting point.

Generate a permutation I of the integers

1, ..., n

$$I = \{ i_1, i_2, \ldots, i_n \}.$$

Sample an integer value x_{i_1} from (6.12).

Now define

$$T_1 = (0, \ldots, 0, x_{i_1}, 0, \ldots, 0),$$

$Z_2 = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 occurs in the i_2th position.

Then

$$T_2 = T_1 + b_2 Z_2.$$

Now apply T_2 to (6.11) to determine the upper and lower bounds on b_2, which must also satisfy $u_{i_2} \leqslant b_2 \leqslant v_{i_2}$. If the constraints are violated or if the range for b_2 does not contain an integer point we must start again with a new x_{i_1}. If the range does contain an integer, select an integer from it by uniform sampling. With this value calculate T_2 then

$$T_3 = T_2 + b_3 Z_3$$

where Z_3 is a vector with a 1 in co-ordinate i_3 and zero elsewhere. Continue until b_n is sampled and T_n calculated. Then T_n is a feasible lattice point which we denote by T^+.

(b) Improving the value of the objective.

Let $R(x)$ denote the gradient of the objective function

$$R(x) = \nabla\left(\sum_{j \in N} a_{oj}\, x_j\right) = (a_{o1}, a_{o2}, \ldots, a_{on}).$$

Define

$$Y(d) = T^+ + d.R(T^+) = T^+ + d.R = T^+ + d(a_{o1}, \ldots, a_{on}). \quad (6.13)$$

R in general would not be integer valued so we normalize R to obtain R^+ as follows.

$$R^{\#} = \min_{R_j \neq 0} |R_j| = \min_{a_{oi} \neq 0} |a_{oi}|, \quad (6.14)$$

$$R_i^+ = \left[\frac{R_i}{R^{\#}} \pm .5\right]$$

$$= \left[\frac{a_{oi}}{R^{\#}} \pm .5\right] \quad (6.15)$$

where $i \in N$ and we apply the addition if $R_i \geq 0$ or the subtraction if $R_i < 0$.

Then

$$Y(d) = T^+ + d.R^+$$

defines the path of search for improved values of the objective function.

Set $d = 0, 1, 2, \ldots$ and evaluate $Y(d)$ in turn until infeasibility occurs. When this happens set d^* equal to the previous d value. If d^* is not zero set $T^+ = Y(d^*)$ and repeat the procedure. If d^* is zero set that element of R of smallest nonzero absolute value

equal to zero to obtain R^1 and proceed as before with R^1 in place of R. If R^1 leads to no improvement set the smallest nonzero absolute value element of R^1 to zero to obtain R^2 and proceed with R^2 in place of R. Continue until R^w is obtained with only one nonzero element. If this results in no improvement try R^{w+1},...., R^{w+k} where k + 1 is the number of nonzero elements in R and R^{w+j} is a vector with 1 in the co-ordinate corresponding to the partial derivative in R which ranks j + 1 in absolute value for $j \leq k$. Ties are broken in order of subscript. If R^{w+k} results in no improvement then T^+ is locally maximal. If any R^j leads to improvement then we set T^+ equal to the $Y(d^*)$ and repeat the procedure.

The procedure as outlined above will handle pure problems. The only modifications necessary for the execution of mixed problems occur in (6.14), (6.15) where normalization only occurs for integer variables, and in the determination of d for (6.13) in the situation where R^j contains zeros in all integer co-ordinates. (6.13) is applied to the constraints to determine non-integer d.

Reiter and Rice [31] do suggest alternative ways of determining a feasible lattice point and of computing d. In some small hand examples we found the method of weighted perpendiculars, the alternative method of searching for a feasible lattice point from a currently infeasible point $X^+ = \{ x_j^+ \}$, unsatisfactory, particularly in cases in which the cone generators

$$-Z = \frac{\sum_{(i/S_i>0)} S_i \left\{ \nabla_i \left(\sum_{j \in N} a_{ij} x_j \right) \right\}_{X=X^*}}{\sum_{(i/S_i>0)} S_i} \qquad (6.16)$$

, where ∇_i denotes the vector of partial derivations of the i th constraint with respect to the x_j's,

form an obtuse angle and the weights

$$S_i = \sum_{j \in N} a_{ij} x_j^* - a_{io}, \qquad i \in M , \qquad (6.17)$$

are unequal. As an example, consider the situation in Figure 6.1

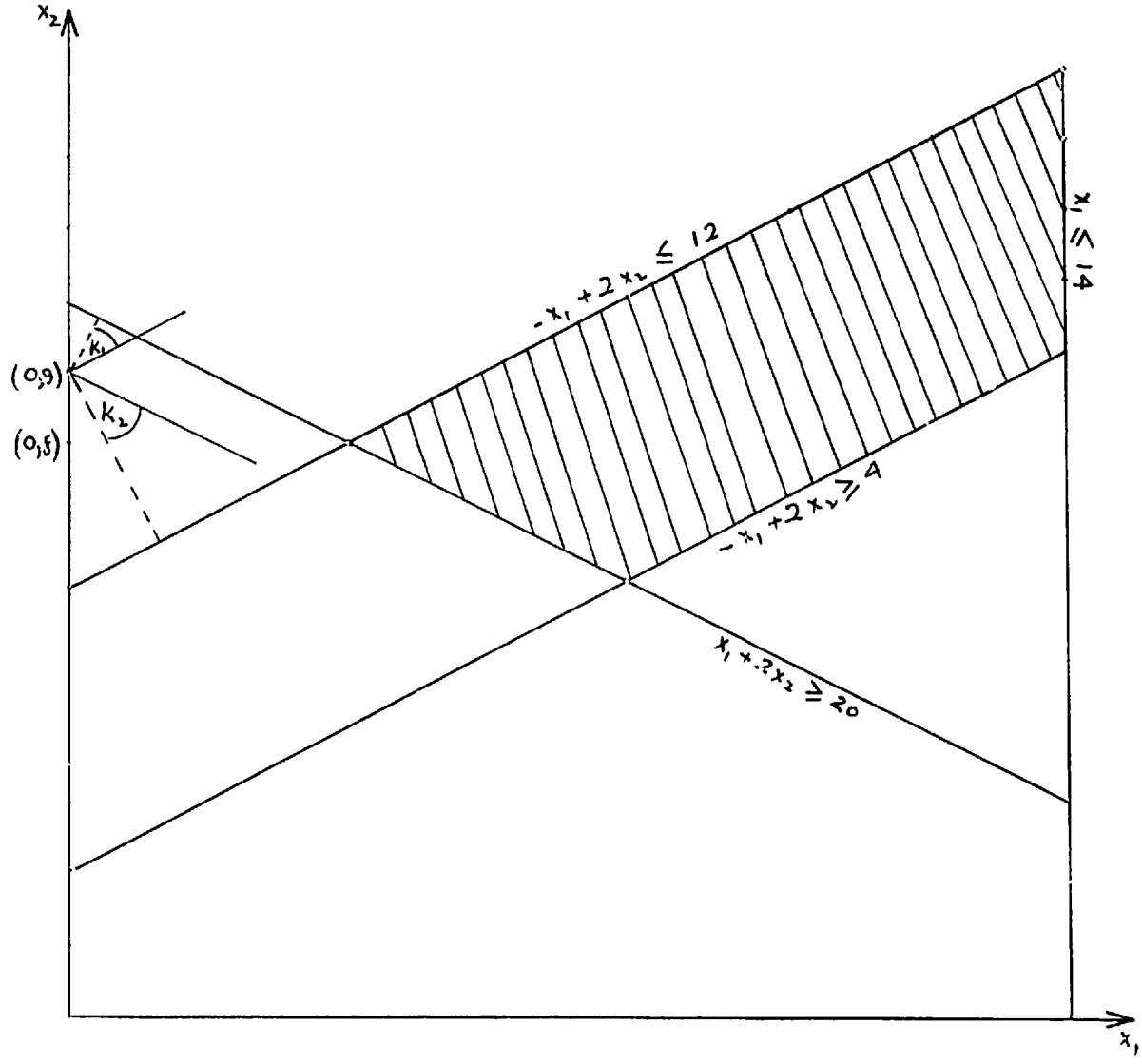


FIG. 6.1

Suppose we started with the point (0,9) then the cone generated by (6.16) would enclose the subcones K_1 and K_2 of totally infeasible directions. In fact, the weights associated with this starting point are

$$S_1 = -1(0) + 2(9) - 12 = 6,$$

$$S_2 = -1(0) - 2(9) + 20 = 2.$$

Then

$$Z = -\left\{\frac{6(-1,2) + 2(-1,-2)}{6+2}\right\} = (1, -1).$$

To determine the actual direction of search for a feasible lattice point Z would now be normalized by a process similar to (6.14) and (6.15) to obtain

$$Z^* = (1, -2).$$

Consider now the simple problem presented by the maximization of $x_1 + x_2$ subject to the space defined in the previous illustration which is bounded by

$$0 \leqslant x_i \leqslant 15, \quad i = 1, 2.$$

Suppose the permutation I gave $I = (i_1, i_2) = (2,1)$. Then sample an integer value $x_{i_1} = x_2$ from (0,15) say $x_2 = 15$.

Then define

$$T_1 = (0,15) \text{ and } Z_2 = (1,0)$$

so $$T_2 = T_1 + b_2 Z_2 = (0,15) + b_2(1,0) = (b_2,15).$$

The conditions

(i) $$0 = u_{i_2} \leqslant u_{b_2} \leqslant b_2 \leqslant v_{b_2} \leqslant v_{i_2} = 15$$

and (ii) $\begin{bmatrix} -1 & 2 \\ 1 & -2 \\ -1 & -2 \\ 1 & 0 \end{bmatrix} T_2 \leqslant \begin{bmatrix} 12 \\ 4 \\ 20 \\ 14 \end{bmatrix}$

must also be satisfied so if we apply (ii) to T_2 we have

$$-b_2 + 2.15 \leqslant 12 \Longrightarrow b_2 \geqslant 18 \text{ which contradicts (i)}$$

so a new sampling is taken for x_{i_1}, say $x_2 = 7$.

Then

$$T_1 = (0,7),\ Z_2 = (1,0)$$

and $T_2 = (0,7) + b_2(1,0) = (b_2,7)$.

Now $0 \leqslant b_2 \leqslant 15$

and
$$\left.\begin{aligned} -b_2 + 2.7 &\leqslant 12 \Rightarrow b_2 \geqslant 2, \\ b_2 - 2.7 &\leqslant -4 \Rightarrow b_2 \leqslant 10, \\ b_2 &\leqslant 14 \\ -b_2 - 2.7 &\leqslant -20 \Rightarrow b_2 \geqslant 6, \end{aligned}\right\} \Rightarrow 6 \leqslant b_2 \leqslant 10.$$

Now sample an integer point from this range, say $b_2 = 8$, then

$$T_2 = (0,7) + 6(1,0) = (6,7),$$

which is feasible, so $T^+ = (6,7)$.

Now $R(x) = (1,1)$, $R^+ = (1,1)$

so $Y(d) = T^+ + dR^+ = (6,7) + d(1,1)$

$d = 1$; $Y(d) = (7,8)$,

$d = 2$; $Y(d) = (8,9)$,

$d = 3$; $Y(d) = (9,10)$,

$d = 4$; $Y(d) = (10,11)$,

$d = 5$; $Y(d) = (11,12)$, which is infeasible, hence $d^* = 4$.

$T^+ = (10,11)$.

$Y(d) = (10,11) + d(1,1)$.

$d = 1$; $Y(d) = (11,12)$, which is infeasible, hence $d^* = 0$.

Replace $R(T^+) = (1,1)$ by $R^1(T^+) = (0,1)$ and repeat.

$Y(d) = (10,11) + d(0,1)$

$d = 1$; $Y(d) = (10,12)$, which is infeasible, hence replace R by $R^2=(1,0)$

$Y(d) = (10,11) + d(1,0)$

$d = 1$; $Y(d) = (11,11)$,

$d = 2$; $Y(d) = (12,11)$,

$d = 3$; $Y(d) = (13,11)$,

$d = 4$; $Y(d) = (14,11)$,

$d = 5$; $Y(d) = (15,11)$, which is infeasible so $d^* = 4$

and $T^+ = (14,11)$ is a new feasible point hence

recommence with this and $R = (1,1)$.

$Y(d) = (14,11) + d(1,1)$.

$d = 1$; $Y(d) = (15,12)$, which is infeasible.

Replace R by $R^1 = (0,1)$.

$Y(d) = (14,11) + d(0,1)$.

$d = 1$; $Y(d) = (14,12)$,

$d = 2$; $Y(d) = (14,13)$,

$d = 3$; $Y(d) = (14,14)$, which is infeasible so $d^* = 2$.

$T^+ = (14,13)$.

Reset $R = (1,1)$

then $Y(d) = (14,13) + d(1,1)$

$d = 1$; $Y(d) = (15,14)$ which is infeasible so $R^1 = (0,1)$

$Y(d) = (14,13) + d(0,1)$

$d = 1$; $Y(d) = (14,14)$ which is infeasible so $R^2 = (1,0)$

$Y(d) = (14,13) + d(1,0)$

$d = 1$; $Y(d) = (15,13)$ which is infeasible and all R sets have been used so (14,13) is locally maximal.

Although this point is, in fact, the global maximum the solution process has no way of knowing this. The current solution would be stored and the whole process repeated from a new sampled starting point. The process can be terminated after an arbitrarily defined number of local maxima has been obtained.

Remarks

As stated earlier, we have had no practical experience with any of the methods described in this chapter. Furthermore, we have encountered evidence of results produced by computer programs for just one of the above described methods, namely, the approximate method. However, for the results reported it was not known if the best local optima tabulated were also global optima.

CHAPTER VII

PRACTICE

In this chapter we discuss the implementation of the techniques presented in Chapters II through V and the results obtained from the solution of problems drawn primarily from [21] and [34]. Some additional problems were generated so as to cause difficulty for the method of Chapter V. However, as development of the method and its code progressed the initial difficulties were overcome. Nevertheless, these particular problems have been included in most data samples. We present the computational steps for each method, formally define various row selection rules and enlarge upon earlier discussions on the results. Some repetition of items covered in earlier chapters is inevitable if we are to avoid constantly referring back to these chapters.

All codes have been written in FORTRAN for use with the ICT Atlas Computer at the University of London. The specifications, source and object decks of these codes are available at the Institute of Computer Science. However, they should be regarded primarily as experimental codes, particularly those related to Chapters III and V, because possible modifications have been discussed earlier and, it is hoped, will soon be under investigation.

In the following codes all problems are regarded as maximization problems. The tableaux are accepted in the condensed Tucker form, with the objective function occupying the leading row and the constant terms occupying the leading column. Each code generates the trivial

rows $x_j = -1(-x_j)$ and constructs the enlarged Tucker tableau by inserting these between the objective row and the initial constraints (except in code 1 where they succeed the objective and initial constraints) so that all variables have row representation and the row order of the variables is maintained throughout.

The order of the discussion of the codes follows that of the corresponding techniques as presented in the earlier chapters.

In the following, the m + n variables, excluding the objective x_o, consist of m basic variables and n nonbasic variables.

Furthermore, let us define the set of rows with negative constant terms, but which excludes the objective row, as

$$I^- = \left\{ i \,\middle|\, a_{io} < 0 \text{ and } i \neq 0 \right\}.$$

Associated with one such row i let the columns containing negative elements in this row be defined by the set of column indices

$$J_i^- = \left\{ j \,\middle|\, a_{ij} < 0 \text{ and } j \neq 0 \right\}.$$

Let the s-th member of a set R be denoted by R(s) and let the order, or dimension, of this set be denoted by q(R).

7.1 Cutting Planes

7.1.1 Code 1. The All Integer Integer Method

As may be expected, this is by far the easiest code to develop due, primarily, to the exclusion of all rounding errors. The only restrictions imposed by this code are concerned with upper limits on storage and iteration count. The code has been written to cater for four variants of the definition of the generator row. Furthermore, the standard or modified version, as defined in Chapters II

and III respectively, can be selected by means of an input parameter. The four generator variants are defined below.

(i) Select a row of index r such that

$$r = \min_{i \in I^-} (i).$$

(ii) During the p-th iteration, select a row of index r such that

$$r = I^-(s)$$

where $s \equiv p \text{ modulo } q(I^-)$, $1 \leqslant s \leqslant q(I^-)$.

(iii) Select a row $r \in I^-$ which produces the lexicographically largest pivot column.

(iv) Select a row r such that

$$q(J_r^-) = \min_{i \in I^-} (q(J_i^-)).$$

Of the methods under discussion, this is the only one which has been coded to allow for lexicography and as such requires the initial tableau to possess a certain format. This initial state criterion is that the tableau be lexicographically dual feasible, that is, the first non-zero elements of columns $j \in N$ must be positive. Clearly then, in the case of dual degeneracy, the tableau (1.8) will not satisfy this requirement. Instead, the trivial rows $x_j = -1(-x_j)$ are interchanged with the basic rows. If the starting criterion is still not satisfied, the method of the artificial constraint could be utilized in order to achieve the required state. However, the code assumes that the starting criterion is satisfied and then proceeds with the following steps.

(i) Select a row r by one of the variants defined above.

(ii) Select a column $k \in J_r^-$ given by

$$a_{sk} = \min_{j \in R_s} a_{sj} \text{ such that } q(R_{s+1}) = 1$$

where $s \in \{0\} + I$, $I = \{1, \ldots, m, \ldots, m+n\}$, $R_0 = J_r^-$ and R_{s+1} is the set of columns satisfying $\min_{j \in R_s} (a_{sj})$.

(iii) Compute $\lambda = \max\left\{\frac{-a_{rj}}{[a_{oj}/a_{ok}]}\right\}$, $a_{ok} \neq 0$,

$= \max(-a_{rj})$, $a_{ok} = 0$.

(iv) Perform the transformation

$$a'_{ij} = a_{ij} + \left[\frac{a_{rj}}{\lambda}\right] a_{ik}.$$

The modification of the cut, as was discussed in Chapter III, is achieved by extending step (iii) in the following manner. Let us define the λ produced by step (iii) as λ_{min}.

Then compute

$$\bar{\lambda} = \min\left\{\frac{a_{io}}{[a_{io}/\lambda_{min}]+1} - \delta, \min_{a_{ij}>0}\left(\frac{a_{ij}}{[a_{ij}/\lambda_{min}]}\right)\right\}, \quad \delta > 0$$

and replace λ_{min} by $\max(\lambda_{min}, \bar{\lambda})$.

This code, with the four variants and using the standard and modified cuts, was applied to 13 test problems. The results are displayed in Table 2. A number of problems gave difficulty as is indicated by the iteration count of 2000 being reached before a solution is obtained. A major fault of this method is that when such an exit occurs the current tableau satisfies the starting criterion of lexicographical dual positivity and primal infeasibility in which case there is no indication even of an approximate solution. Problem 23 proved to be the most difficult for this code, just one

variant producing a solution within the restriction of 2000 iterations. From the point of view of iteration count there appears little to choose between variants (i) and (iii), namely, selecting the first admissible generator encountered and selecting that row which produces the lexicographically largest pivot column. However, it is worthwhile drawing attention to the times required for executing the entire set of 13 problems. The set, subject to the first variant, required a computing time of 18.54 seconds while subject to the third variant the time was 32.7 seconds. The first variant clearly has the fastest cycle time per iteration. The work involved in (iii) in determining the complete set of generators and choosing appropriately from among them has caused a 76% increase in execution time relative to (i). The second variant, namely selecting the generator cyclically, could be regarded as generally less effective than the others but it is interesting to note that it did succeed in solving problem 18 in 27 iterations. Based on this small experiment the fourth variant quite clearly emerges as the most effective since it did not fail on any problem and only on two problems did it not require the least number of iterations.

7.1.2 Code 2. The Pure Integer Method

This has been coded in fixed point arithmetic employing the method of using a numerator and denominator for tableau entries, as given in [15], and consequently has similar limitations to those discussed in that paper. At this point, it may be appropriate

to convert the tableau and the transformation formulae used previously into the form employed in this and some other codes.

Suppose that the problem is still defined by (1.10), (1.11), (1.12) and (1.13). However, let us now change tableau (1.8) as follows. Let D be the least common multiple of all denominators of the fractions representing the real coefficients a_{ij}, then a general element of the original tableau is $a_{ij} = a'_{ij}/D$. Multiplying all elements by D we obtain the new problem tableau

$$\begin{array}{c|cccccc} & 1 & -x_1 & -x_2 & \cdots & -x_n \\ \hline x_o & a'_{oo} & a'_{o1} & a'_{o2} & \cdots & a'_{on} \\ x_1 & 0 & -D & 0 & \cdots & 0 \\ \vdots & \vdots & & & & \\ x_n & 0 & 0 & 0 & & -D \\ x_{n+1} & a'_{n+1,o} & a'_{n+1,1} & a'_{n+1,2} & \cdots & a'_{n+1,n} \\ \vdots & \vdots & & & & \\ x_{n+m} & a'_{n+m,o} & a'_{n+m,1} & a'_{n+m,=} & \cdots & a'_{n+m,n} \end{array} \qquad (7.1)$$

with an associated denominator D so that, for example, the value of x_{n+1} is $a'_{n+1,o}/D$ and its equation is

$$x_{n+1} = a'_{n+1,o}/D + \sum_{j=1}^{n} a_{n+1,j}(-x_j)/D.$$

For convenience let us drop the primes. After the attainment of primal dual continuous feasibility the steps of the code proceed as follows.

(i) Select a row r such that $a_{ro} \not\equiv 0$ modulo D.

(ii) Augment tableau (7.1) by the row

$$a_{sj} = D\left[\frac{a_{rj}}{D}\right] - a_{rj} \quad , \quad j = 0,1,\ldots,n.$$

(iii) Apply the dual simplex algorithm until $a_{io} \geq 0$, $1 \leq i \leq m+n$.

(iv) Test for feasibility. If the tableau is feasible, i.e. $a_{io} \equiv 0$ modulo D, then the first column defines the optimal solution. Otherwise repeat the cycle (i) to (iv).

This method is capable of modification in a number of ways, first with regard to the selection of the generator row r, second with regard to the problem of two or more nonbasic variables qualifying for entry into the basis, and third with regard to the problem of dual degeneracy, which is largely dealt with by the previous variation class.

Let us define $I' = \left\{ i \middle| a_{io} \not\equiv 0 \text{ modulo } D \right\}$.

Then, in connection with the first class we have used the following four variants on some problems.

(i) Select a row r such that

$$r = \min_{i \in I'}(i).$$

(ii) Select a row r with

$$f_{ro} = \max_{i \in I'}(f_{io}).$$

(iii) Select a row r with

$$-\bar{f}_{ro} = \min_{i \in I'}(-\bar{f}_{io}).$$

(iv) Select a row r with

$$f_{rk} = \min_{i \in L}(f_{ik}), \quad L = \left\{ i \middle| a_{ij} \not\equiv 0 \text{ modulo } D \right\}$$

where the column k satisfies the condition

$$a_{ok} = \min_{j \in N}(a_{oj}).$$

This set of four rules for selecting a generator row is itself susceptible to variation in that the set I' can be restricted.

I', as defined, is the set of rows with a_{io} non-integer and $i \in \{0\}+ I$, $I = \{i \mid 1 \leqslant i \leqslant m+n\}$, if the objective function is to be integral but $i \in I$ otherwise. In the code we have catered for the possibility of admitting all rows as potential generators but we have not used this variant in our experiments and have restricted I' to the set $i \in \{0\}+ N$. This means that we have neglected the slack rows when seeking a generator. Furthermore, in our experiments which have been confined to problems in which the objective function is also required to be integral, we have subjected the set $i \in \{0\} + N$ to the two variants defined by this set itself and this set with $\{0\}$ excluded. This amounts to admitting the objective row as a possible generator by the former and barring it from inclusion in the generator search by the latter variant.

A pivot column may not be uniquely defined during the process of selection in the dual simplex method. To cater for such an event we have tried four variants. Let there be b columns satisfying the entry variable rule and let their indices be defined by the set $J' \subset N$. Let $J'(t)$ denote the t-th member of J'. The selection rules are as follows.

(i) Select column k such that

$$k = \min_{j \in J'}(j) \quad .$$

(ii) On the p-th occasion of non-uniqueness select column k such that

$$k = J'(t)$$

where $t \equiv p$ modulo b , $1 \leqslant t \leqslant b$.

(iii) Select at random from among the tying columns.

(iv) Select a column k such that

$$a_{rk} = -D$$

and $$a_{rj} = 0.$$

If none exists use (iii).

Choosing the first column (variant (i)) satisfying the criterion for an entry variable is common practice. However, since the code is not dependent on 1-feasibility, difficulties can arise due to the occurrence of dual degeneracy, a not uncommon situation in integer programming. Due to this, selection of the same column can cause repetition of bases and prohibit progress towards a feasible solution and we attempted to avoid this situation by the introduction of the variants (ii), (iii) and (iv). Variants (ii) and (iii) are quite straightforward and fairly obvious possible choices but variant (iv), which has no real logical basis for its definition, was a choice taken in some hand computations where it seemed preferable to introduce one of the original variables into the basis rather than one of the slack variables. Machine computations certainly favoured variant (iii), which selected randomly from among the tying columns.

Finally, the problem of dual degeneracy. The variants of class one would sometimes select the objective row as generator row, and this could generate a hyperplane parallel to itself as the new cutting constraint, in which case the succeeding transformation would produce (n-1) zeros in the leading row of the tableau. This, clearly, eventually leads to the situation of class two. In an attempt at avoiding

dual degeneracy we restricted the range over which the class one variants could search, that is, we excluded the objective function from the set I'.

All variations of the first two classes were applied to a small set of problems in an initial attempt at deciding upon some reasonable combination of the generator selection and column selection rules. The necessity for a column selection variant was due to the fact that this code does not use a lexicographical approach and many of the problems to which the code is to be applied are degenerate. The results obtained from the first two problems of this initial set are given in Tables 3 and 4. The results from these two problems indicate the behaviour typical within the complete set and appeared to indicate that variant (ii) or (iv) of the first class combined with variant (iii) of the second class should probably be favoured, with no preference at this early stage between generator variants (ii) or (iv).

Both of these variant combinations were used in the application of this code to the complete set of 30 problems defined later. We performed two runs through the set, in one the objective function was admitted as a possible generator, in the other it was excluded. The results of these runs are exhibited in Tables 5 and 6. An upper limit of 2000 iterations was again imposed on each problem by means of an input parameter. As indicated by the tables, the behaviour is unpredictable and quite erratic. If we regard as a success, a problem which has been solved within the iteration restriction and if we accept

the number of iterations required to solve a problem as an indication of performance, with the understanding that the fewer the iterations the better the performance, then purely from the point of view of the number of improved performances the generator variant (iv) was slightly more satisfactory than variant (ii). The same appears to be true for the set of results obtained by excluding the objective function from consideration. Although dual degeneracy occurs in most problems, the lack of general improvement in performance when attempting to avoid degeneracy was surprising. In particular, it is interesting to note how very badly this change affected problems 20, 21 and 22 especially in Table 5, which we have regarded as representing the better performance. However, it is also interesting to note that the whole set of 30 problems took 85.7 seconds to solve by variant (iv) with no restriction on the objective function while the set took 67.2 seconds to solve when excluding the objective. Probably the most striking example indicating the vast differences encountered by using different generators is problem 15. It is a small two variables two constraints problem, it was unsolved by the maximum fractional part variant after 2000 iterations yet was solved in 37 iterations by using the alternative variant.

7.1.3 Code 3. Mixed Integer Routine

Although this code has not been used in the comparison tests it was used during the development of codes 2 and 4. In some of the problems attempted by codes 2 and 4 the denominator D became large

and the accuracy of the computation diminished. Code 3 was then employed on these same problems and eventually indicated areas for improvement in codes 2 and 4 such that better accuracy could be achieved.

In the mixed integer routine control of rounding errors does present a problem. Such control is, in this code, achieved by using a parameter BOUND which may be regarded as an indirect measure of the accuracy expected in a problem. For example, if BOUND = 0.4999999 then the routine assumes that any number within 0.5-BOUND =.0000001 of an integer may be regarded as that integer. This parameter is used in a number of places throughout the code to determine whether or not numbers are integral, and in comparison tests where the equality branch of a three way decision, even though of extreme importance, may never attain the state of equality without the introduction of some such tolerance limit.

Another minor difference, though not a difficulty, between this and the other codes is that in order to define correctly the cut to be introduced to the current system, we must know the mode of the nonbasic variables. This question is answered by the following process. Suppose the integer variables occupy the rows 1,2,....,t of a tableau and let the tableau columns $j \in N$ have associated with them identifiers J(j) such that J(j) = 1 implies that the column of index j refers to an integer variable and J(j) = 0 implies that the column refers to a continuous variable, that is, the modes of the members of the set of nonbasic variables are defined by J(j), $1 \leqslant j \leqslant n$. The actual column variables are not required to be known by name but during the con-

struction of a new constraint from a selected generator it is necessary to know whether they are of integer or continuous mode. During the pivotal operation some variable is to enter the basis to the exclusion of another. Since the row order in the tableau is maintained the procedure automatically and correctly adjusts the appropriate variables, but it is still necessary to know how the modes of the column variables have changed, if at all. For example, suppose the pivotal element is a_{rk}, then this means that the r-row variable is to leave the basis and the k-column variable to enter. Thus the J(j) mode identifiers should change as follows.

$$\text{If } 1 \leqslant r \leqslant t \quad \text{set } J(k) = 1,$$

$$\text{if} \quad t < r \quad \text{set } J(k) = 0.$$

As with the other codes this code employs the simplex method to proceed to a primal dual continuous feasible tableau. The outline of the succeeding steps will be simplified somewhat by means of the following simple definitions. Let R denote the set of indices of the nonbasic integer variables. Let N_c be the subset of $j \in N$ not contained in R. Having selected a row r as generator row let N_c^+ be the subset of N_c with $a_{rj} > 0$ and let N_c^- be the subset of N_c with $a_{rj} < 0$. Then from a primal and dual continuous feasible tableau the code proceeds with the following steps.

(i) Select a generator row r such that

$$a_{ro} \not\equiv 0 \text{ modulo } 1 , \quad r \in K = \{1,\ldots,t\}.$$

(ii) Augment the current tableau by the row

$$s = -f_{ro} - \sum_{j \in N} d_{rj} (-t_j)$$

$$\text{where } d_{rj} = \begin{cases} a_{rj}\ , & j \in N_c^{+} \\ \dfrac{f_{ro}}{1 - f_{ro}} \left| a_{rj} \right|\ , & j \in N_c^{-} \\ f_{rj}\ , & j \in R \text{ and } f_{rj} \leqslant f_{ro} \\ \dfrac{f_{ro}}{1 - f_{ro}} (1 - f_{rj})\ , & j \in R \text{ and } f_{rj} > f_{ro}. \end{cases}$$

(iii) Apply the dual simplex algorithm until $a_{io} \geqslant 0$, all $i \in I$.

(iv) Test the appropriate rows for feasibility. If $a_{io} \equiv 0$ modulo 1, $1 \leqslant i \leqslant t$, then the computation terminates with the optimal solution in the first column of the tableau. Otherwise the preceding steps are repeated from (i).

As before, the code caters for four variants of step (i) similar to those defined in Section 7.1.2.

7.1.4 Code 4. The Accelerated Euclidean Algorithm

As in the case of code 2 (Method of Integer Forms), which forms the nucleus of the current code, the computation is carried out in fixed point arithmetic. We again suppose the integral tableau elements to be a_{ij} and the associated common denominator D. Since this method (Martin [29]) has not been described earlier we briefly outline the basic principles below and follow this by the development of the appropriate transformation formulae.

The method starts by finding a non-negative optimal solution to the original problem. Then, assuming this to be non-integral, a row r with $a_{ro} \not\equiv 0$ modulo D is used to generate a "consolidated" constraint by employing the principles of Gomory's cutting plane

technique. Successively using cuts of this type as pivot rows and carrying out the necessary pivot operations will produce a system which is integral but which may violate the optimality condition as well as the non-negativity condition. The whole process is then repeated from the beginning. The production of the consolidated constraint is of fundamental importance and is achieved by constructing, from an infeasible row r, a sequence of row vectors $p_0, p_1, \ldots, p_s$ where p_0 is the vector representing row r, each vector p_{i+1} is obtained from its predecessor p_i by selecting the pivot from the associated Gomory cut and applying a Gaussian transformation to p_i only, and the terminating vector p_s is one with an integer element in the pivot column. A reverse Gaussian transformation then determines the consolidated cut which would produce p_s from p_0.

In order to develop the appropriate transformation formulae in terms of our formulation we assume that $a_{oj} \geq 0$, $j \in N$, and $a_{io} \geq 0$, $i \neq 0$.

Assume that we have extracted from the tableau a generator row r where

$$x_r = a_{ro}/D - \sum_{j \in N} a_{rj}(-t_j)/D. \tag{7.2}$$

If the augmented row has row index s then the tableau entries in this row are

$$a_{sj} = D\left[\frac{a_{rj}}{D}\right] - a_{rj}, \quad j = 0,1,\ldots,n, \tag{7.3}$$

and the pivot column k is given by

$$\frac{a_{ok}}{|a_{sk}|} = \min_{j \in N} \frac{a_{oj}}{|a_{sj}|} \quad .$$

Performing the indicated pivot operation on the necessary row would replace (7.2) by

$$x'_r = a'_{ro}/D' - \sum_{j \in N}' a'_{rj}(-t_j)/D' \quad ,$$

where
$$D' = -a_{sk} = -(D\left[\frac{a_{rk}}{D}\right] - a_{rk}) \quad , \tag{7.4}$$

and
$$\frac{a'_{rk}}{D'} = \frac{-a_{rk}/D}{[a_{rk}/D] - a_{rk}/D} = \frac{-a_{rk}}{D[a_{rk}/D] - a_{rk}}$$

$$\Longrightarrow a'_{rk} = a_{rk} \text{ by (7.4)} \quad , \tag{7.5}$$

and
$$\frac{a'_{rj}}{D'} = \frac{a_{rj}}{D} + \frac{a'_{rk}}{D'}\left\{\left[\frac{a_{rj}}{D}\right] - \frac{a_{rj}}{D}\right\} , \; j \neq k,$$

$$\Longrightarrow a'_{rj} = \frac{a_{rj}(D' - a'_{rk})}{D} + a'_{rk}\left[\frac{a_{rk}}{D}\right]$$

$$= \frac{a_{rj}(-D[a_{rj}/D])}{D} + a'_{rk}\left[\frac{a_{rj}}{D}\right]$$

$$= -a_{rj}\left[\frac{a_{rk}}{D}\right] + a_{rk}\left[\frac{a_{rj}}{D}\right] \quad . \tag{7.6}$$

Examination of these transformation formulae indicates that the complete sequence of rows leading to the row with the required integer property can be defined. The elements defined by (7.5) are integers, by definition, and clearly the expression in (7.6) is integral.

To revise the situation, if we have the elements a_{rj} and common denominator D of an infeasible tableau row then new elements a'_{rj} with common denominator D' can be produced from the current ones by

$$\left.\begin{aligned} a'_{rk} &= a_{rk}\,, \\ a'_{rj} &= a_{rk}\left[\frac{a_{rj}}{D}\right] - a_{rj}\left[\frac{a_{rk}}{D}\right]\,, \quad j \neq k, \\ D' &= a_{rk} - D\left[\frac{a_{rk}}{D}\right], \end{aligned}\right\} \tag{7.7}$$

where k is defined such that

$$\frac{a_{ok}}{|a_{sk}|} = \min_{j \in N}\left\{ \frac{a_{oj}}{\left|D\left[a_{rj}/D\right] - a_{rj}\right|} \right\} .$$

This process of applying (7.7) to the current row is repeated until the appropriate column entry attains integrality. At this stage, it is necessary to determine the transformation relating the original row (7.2) with the current row having the desired property. Suppose a_{rj} represents the generator row then the cut that will produce the necessary row a'_{rj} from this is given by

$$\left.\begin{aligned} a_{sj} &= \frac{D.a'_{rj} - a_{rj}}{a'_{rk}}\,, \quad j \neq k, \\ a_{sk} &= -\frac{a_{rk}}{a'_{rk}} \,.\, D' \,. \end{aligned}\right\} \tag{7.8}$$

The computation starts in exactly the same manner as in code 2 and proceeds with the following steps.

(i) Apply the simplex method until $a_{oj} \geq 0$, $j \in N$, and $a_{io} \geq 0$, $i \in I$. If $a_{io} \equiv 0$ modulo D then the current solution is the optimum, otherwise go to step (ii).

(ii) Select a row r such that

$$a_{ro} \not\equiv 0 \text{ modulo } D.$$

(iii) Use this row to define the elements (7.3) of the Gomory cut.

(iv) Choose the pivot column k by the dual simplex method pivot selection rule and pivot on row r only, using the transformation formulae (7.7).

(v) If the new row has $a'_{rk} \equiv 0$ modulo D' proceed from step (vi), otherwise return to step (iii) with this row as the new r row.

(vi) Apply (7.8) to the original generator row r and the current row r to produce the consolidated cut and augment the tableau by this cut.

(vii) Apply the dual simplex method. If the solution is all integer go to step (i), otherwise go to step (ii).

Our experience with this code is extremely limited. In the few problems tried, the results were disappointing especially in view of the success claimed for this algorithm in solving large problems (Beale [4]). Experimentation with this code did not fit in with the eventual investigation into cutting plane methods. Consequently, no results are given.

7.1.5. Code 5. The Optimal Cut Routine

As in the previous modification of the Method of Integer Forms, code 2 again forms the basis of this routine. However, at the stage when code 2 defines a cut and proceeds with the dual simplex method this code attempts to improve the cut in the manner described in Chapter III.

Suppose that a_i, D and r are non-negative integers. Then the further problem which arises in this technique is to find the minimum

value of r and non-negative integers x_i such that

$$\sum_{i \in N} a_i x_i = a_o + r D . \qquad (7.9)$$

The method of solution adopted is elementary and successful. Any improvement in this area can only add to the attraction of the optimal cut method.

Briefly, we fix r at successive non-negative integer values and attempt to solve the corresponding diophantine equations (7.9) in non-negative variables. We have attempted to direct and limit the search for a solution by first arranging the coefficients a_i in order of increasing magnitude. One distinct advantage here is that, since (7.9) is derived from a Gomory cut, all the a_i are non-negative. For a fixed value of r the search is finite, being bounded by the total enumeration of x_i values. If no solution is found at one level the current value of r is increased by one and the process repeated. The process is quite easy to code since it is recursive, as will be seen from the following description.

Suppose the coefficients a_i to be ordered such that $a_i \leqslant a_{i+1}$, $i = 1,2,\ldots,n-1$, and moreover that $a_i \neq 0$ for all $i \in N$. We may also assume that $a_o \geqslant a_i$ for at least one i, otherwise it is clear that no solution exists when $r = 0$.

Let $\sum_{i=1}^{n} a_i x_i = a_o + rD = \phi_n$ define problem P_n, and set $x_n = \left[\frac{\phi_n}{a_n}\right]$ so that the remaining unknown variables x_i must satisfy

$$\sum_{i=1}^{n-1} a_i x_i = \phi_n - a_n \left[\frac{\phi_n}{a_n}\right] = \phi_{n-1} \quad , \text{ which defines problem } P_{n-1}.$$

Continuing in this way, setting $x_k = \left[\frac{\phi_k}{a_k}\right]$, defining P_{k-1}, and so on, we would arrive at

$$a_1 x_1 = \phi_2 - a_2 \left[\frac{\phi_2}{a_2}\right] = \phi_1 ,$$

$$\text{and then define } \phi_1 - a_1 \left[\frac{\phi_1}{a_1}\right] = \phi_o .$$

If at any stage ϕ_k is zero then (7.9) is satisfied by

$$X = \left(\{0\}_1^k , \left\{\left[\frac{\phi_i}{a_i}\right]\right\}_{k+1}^n \right) , \text{ where } \{a\}_1^n = (a_1, a_2, \ldots, a_n).$$

If the entire sequence $\{P_i\}_n^1$ is produced and $\phi_o \neq 0$ then a new subsequence must be defined, commencing at the last non-zero setting for x_i. Suppose this was $x_k \neq 0$ then replace x_k by $x_k - 1$ and ϕ_{k-1} by $\phi_{k-1} + a_k$ and proceed with the subproblems $\{P_i\}_{k-1}^1$. This process is repeated until either some $\phi_k = 0$, in which case the solution is as given above, or $x_2 = x_3 = \ldots = x_n = 0$ and $\phi_o = -a_1 \left[\frac{\phi_n}{a_1}\right] + \phi_n \neq 0$ which indicates that no solution exists for the current value of r.

Although this process was formally presented and used to demonstrate the optimal cut method in Chapter III, we consider its illustration by means of a small example to be helpful at this point.

Suppose we have

$$8x_1 + 17x_2 + 23x_3 = 4 + 43r. \qquad (7.10)$$

Since $a_i > a_o$, all i, no solution in non-negative integers exists when r=0.

Cycle 1 Set r = 1

Then $8x_1 + 17x_2 + 23x_3 = 47 = \phi_3$

$x_3 = \left[\frac{47}{23}\right] = 2$; $8x_1 + 17x_2 = 47 - 2.23 = 1 = \phi_2$

$x_2 = \left[\frac{1}{17}\right] = 0$; $8x_1 = 1 = \phi_1$

$x_1 = \left[\frac{1}{8}\right] = 0;$ $\qquad 1 = \phi_0$

$\phi_0 \neq 0$ and $x_3 \neq 0$ so set $x_3 = 1$ and $\phi_2 = 1 + 23 = 24$

i.e. $8x_1 + 17x_2 = 24 = \phi_2$

$x_2 = \left[\frac{24}{17}\right] = 1;$ $\qquad 8x_1 = 24 - 17.1 = 7$

$x_1 = \left[\frac{7}{8}\right] = 0;$ $\qquad 7 = \phi_0$

$\phi_0 \neq 0$ and $x_2 \neq 0$ so set $x_2 = 0$ and $\phi_1 = 24$

i.e. $8x_1 = 24 = \phi_1$

$x_1 = \left[\frac{24}{8}\right] = 3;$ $\qquad 24 - 3.8 = 0 = \phi_0$

The solution to the diophantine equation (7.10) is then

$$\underline{\underline{r = 1, \quad x_1 = 3, \quad x_2 = 0, \quad x_3 = 1}}$$

All the variants discussed in Section 7.1.2 are clearly applicable here but we have confined our implementation to the variants selected for the major experiments under Section 7.1.2. This enabled us to make a direct comparison between the two respective codes. That is, we used the generator variants

(ii) select a row r with $f_{ro} = \max_{i \in I'} (f_{io})$

and

(iv) select a row r with $f_{rk} = \min_{i \in L} (f_{ik})$, $L = \left\{ i \middle| a_{ij} \not\equiv 0 \text{ modulo } D \right\}$

where the column k satisfies $a_{ok} = \min_{j \in N} (a_{oj})$,

and we select at random from among tying pivot columns.

The computational steps of this method are similar to those of code 2 and their repetition here would serve no worthwhile purpose.

However, we do note that instead of defining the Gomory cut this code determines the optimal cut (which may of course be the Gomory cut) by the process described above.

This code, using the previously defined generator variants and with and without restriction on the use of the objective function, was applied to the complete set of 30 problems. The results are displayed in Tables 7 and 8, and the solutions produced by the different variants are all tabulated. Those in Table 7 were obtained by using the largest fractional part rule for choosing a generator, while Table 8 presents those obtained by constructing cuts from rows containing the smallest fractional part in the column with the smallest relative cost coefficient. As in all of our codes, the search for a generator is confined to the leading $(n + 1)$ rows of the tableau. For each generator selection rule, two computer runs through the entire set of 30 problems were performed. In one the objective function was included in the search for a generator while in the other it was excluded.

Summary performance figures for both the standard and optimal cutting plane methods were tabulated in Chapter III. Apart from having encountered some of the difficulties discussed in that chapter, this code gave a performance which was generally satisfactory; in fact, for the problems 15 to 24 taken from [34], the code was most successful, having reduced the iteration count for one problem by 99% and for the majority of remaining problems by amounts ranging from 2% to 96%. These figures have not taken into account problems

which, although unsolved by the standard method, were rapidly solved by the optimal cutting plane method. On the other hand, we should point out that the optimal cut code did experience difficulty with problem 18 when using the first generator variant. By including the objective function in the set of possible cut generators the iteration count of the standard method was increased by a factor 3 while exclusion of the objective function caused the numbers to become too large and no solution was produced. Relative to the variants used, the best performance figures are fairly evenly divided between both generator selection rules. Finally, in common with most of the codes, the performance is unpredictable.

7.2 Branch and Bound Methods

7.2.1 Code 6. Dual Branching Tree Search Code

Two major features of this code are the control of rounding errors and the adaptive storage arrangements. Rounding errors are dealt with by the technique outlined in Section 7.1.3. The storage arrangements are designed to avoid the use of magnetic tape backing store whenever possible.

Dakin [8] has coded a branch and bound technique exclusively for in-core application while Beale and Small [5] make use of magnetic tape for retaining all subproblems generated. We point out here, that in the following discussion, when we refer to core we mean the virtual core as far as the programmer is concerned.

At the time this code was developed the version of FORTRAN available on the London Atlas contained no facilities for dynamic

arrays. This led to some inconvenience in the form of data handling that had to be adopted when our main requirement was to saturate the available one level store before utilizing magnetic tape. For this reason we define a variable LOC to be the number of locations available in core at any instant during execution, and an array TAB to be an area of core in which subproblems may be stored. The dimension statement is initially set up to cater for a working tableau of $(m + n + 1 + m')$ rows and $(n + 1)$ columns, and a storage area of k rows and $(n + 1)$ columns where $k \equiv 0$ modulo $(m + n + 1 + m')$. (Note that the number of rows here includes the $(m + n + 1)$ rows of the original problem plus a number of rows equal to the number of integer variables, since we keep track of the bounds on the integer variables by defining new integer variables representing the amount by which the current lower bounds are exceeded). The value of k is dependent upon the amount of core available and after being arbitrarily defined for a compilation run, can be redefined to demand a feasible amount of core. Even quite small problems may generate many subproblems but to demand unnecessary core store on a multiprogramming machine such as Atlas is clearly inappropriate. Before continuing, we should point out a particular feature of the Atlas FORTRAN system which has had to be allowed for in this code. If during the use of magnetic tape in the binary mode it is necessary to backspace over a record, overwrite this record with a new one and then read this record at some later stage, it is necessary that the records shall each consist of at least 512 words.

From the user's point of view, the amount of one level store available on Atlas is considerable (132K) and it may safely be assumed that only one of two possible states can exist during the solution of some problem.

(i) The working tableau may exceed 512 words in size. This clearly is a simple situation which may or may not use one level storage.

(ii) The tableau contains fewer than 512 words. It can be assumed that one level store is available and that the number of available locations is at least 512.

The first of these two situations presents no problem. Storage can occur at the first level until all core modulo $(m + n + 1 + m')(n + 1)$ is used. Then, when a further problem is about to be stored, this complete block of information can be transferred to tape and storage in core recommenced. If one level store is not used then magnetic tape can be utilized storing tableaux of size $(m + n + 1 + m')(n + 1)$ words per record. The second situation is slightly more complex. Before tape store can be utilized a block size of at least 512 words must exist. At the start of the routine the transfer block size, ITRANS, is determined and must have the properties

$$\text{LOC} \geqslant \text{ITRANS} \geqslant 512,$$

$$\text{ITRANS} \equiv 0 \text{ modulo ITAB}$$

$$\text{where ITAB} = (m + n + 1 + m')(n + 1),$$

except in the case

$$2.\text{ITAB}-1 \geqslant \text{LOC} \geqslant 512.$$

In this case $\left[\frac{LOC}{ITAB}\right] = 1$ so that one block only may be stored in core and it is clearly not possible to satisfy the property ITRANS $\equiv$ 0 modulo ITAB. In this situation the block tableau dimension will allow for just one tableau but during transfer to tape the tableau will be augmented by spurious data to make up a 512 word block.

One further feature of the data handling remains to be mentioned. Whenever stored data is required it is extracted on the "last in first out" basis. If all the listed problems are stored entirely either in core or on magnetic tape, the recovery of individual problems for solution is easily handled. With magnetic tape, for example, the tape is always positioned with the read/write heads at the end of the last listed problem, which will be required first and is thus readily available. If both core and magnetic tape are used for storing subproblems, the core always contains the last listed problem which is immediately available. Newly generated subproblems which require to be listed are successively stored in core until all available core has been used, at which point the information contained in the transfer block TAB is written to tape. However, such a transfer only takes place when another subproblem is about to be listed. This prevents the subproblem area of core ever being empty as long as subproblems exist. Whenever a listed problem is required it is copied to the working area within core and an indicator set to show that further core store is now available for listing. If the problem extracted

from the list is the last one in core, then during its execution a transfer block of information is retrieved from magnetic tape and the indicator set to show that the core store is fully utilized.

Although only applied to pure integer problems in our experiments, the code has, in fact, been written to cater for the solution of mixed integer problems. The major steps in the computation are as follows.

(i) Apply the simplex method to the original problem until $a_{oj} \geq 0$, $j \in N$, and $a_{io} \geq 0$, $i \in I$. Initialize the subproblem list count.

(ii) Test the appropriate rows for feasibility. If $a_{io} \equiv 0$ modulo 1 for all $i \in K$, where K denotes the set of indices of the variables required to be integer, then the optimal solution of this problem is contained in the first column of the tableau; proceed from step (vii). If $a_{io} \not\equiv 0$ modulo 1 for $i \in K' \subset K$, proceed from step (iii).

(iii) For each $i \in K'$ compute

$$(a) \quad \begin{cases} u_i = \min\limits_{\substack{a_{ij}<0 \\ j \in N}} (a_{oj}/|a_{ij}|), \\ du_i = (1 - f_{io})\, u_i \ ; \end{cases}$$

$$(b) \quad \begin{cases} v_i = \min\limits_{\substack{a_{ij}>0 \\ j \in N}} (a_{oj}/a_{ij}) \ , \\ dv_i = f_{io}\, v_i \ . \end{cases}$$

Then determine $w_r = \max\limits_{i \in K'} (du_i, dv_i)$

(iv) If $w_r = du_r$ then replace the lower bound on x_r by $\left[a_{ro}\right] + 1$ and list the current modified tableau. Update the list count. Modify the current tableau by replacing the upper bound on x_r with $\left[a_{ro}\right]$ and proceed from step (v).

If $w_r = dv_r$ then list the second problem defined above, modify the current tableau by replacing the lower bound on x_r with $\left[a_{ro}\right] + 1$ and proceed.

(v) Apply the dual simplex method. If $a_{io} < 0$ for some $i \in I$ and $a_{ij} \geq 0$, all j, then go to step (vi). Otherwise compare the functional value of this solution with the stored best value. If the present value is larger return to step (ii) if not then continue from step (vi).

(vi) If the list count is zero then no subproblems remain and the current best solution is the optimum. If the list count is not zero, recover the last stored subproblem and proceed from step (v).

(vii) Compare the functional value of this solution with the stored best value and retain the better. Go to step (vi).

A feature of major importance in this method is the process of selecting a variable on which to branch. Dakin [8] mentions some selection criteria and indicates his experience with them. We use the technique outlined in Chapter V and have found it most satisfactory. Having fixed our branching rule, the only variants that we have tried have been in connection with selecting a unique column from among tying columns when seeking a variable to replace a variable

which is about to become nonbasic. The code, using two column selection variants ((i) selecting the first and (ii) selecting randomly from the tying columns), has been applied to the set of 30 problems and the results are shown in Tables 9 and 10. In general, it appears that selecting randomly from among tying columns, although sometimes improving the situation in the continuous phase, worsens the situation in the integer phase. Of major significance is the fact that, except for two problems, the first integer solution reached by using the first variant was later proved to be the optimal integer solution. The second variant did not achieve such success since six problems attained first integer solutions which were non-optimal. In a number of problems, although the first integer solution reached proved to be the optimal solution many iterations were required. This arose due to the solution process having to retreat from many infeasible problems. This, however, did not make the process less efficient, because even if progress to the optimal solution had been direct, these infeasible problems would still have required generation due to them emanating from nodes of higher objective function values than the current best.

The performance figures of this code were compared with those of the code for the standard cutting plane method and discussed in Chapter IV.

7.2.2 Code 7. Convex Isolation and Bound Contraction

As discussed in Chapter V, the idea of this approach was

developed after a critical examination of the papers by Beale and Small [5] and Dakin [8]. The theory pointed to certain computational difficulties but as the investigation progressed these were overcome.

Once the first feasible solution is reached a convex region is defined by the hyperplanes emanating from the continuous optimum and the objective function so positioned as to exclude the current feasible solution from this new convex. Of the possible methods discussed in Chapter V for producing the vertices of this convex we decided to code the first because, although probably less sophisticated than the others, we felt it to be more efficient.

Apart from some control constants the input to this code is identical with that of the previous codes. The only restriction imposed is on the amount of space available for storing any hypotheses made about the bounded region over integer space when it can be reduced no further by the bound contraction process. A further point of interest in this code is the problem of measuring its efficiency. Previous methods have been assessed for efficiency purely on iteration count. In this technique, once a feasible solution is attained the pivotal operations cease and, instead, the contraction process takes place. A system of intersecting hyperplanes enclosing a subspace of the original solution space is investigated to determine whether or not it contains a feasible subspace. This process requires consideration of each such hyperplane in turn and we regard the sequential examination of the set of hyperplanes bounding this new convex

region as a cycle. This set of hyperplanes is a subset of the original constraint set but it is rather difficult to theoretically relate a cycle and an iteration. However, an analysis of the times taken in solving various problems appeared to indicate that a cycle required one tenth of the time taken by an iteration.

Before presenting the computational steps of the method we recall the two stages capable of modification.

The first concerns the logical flow of the method and occurs at the stage when a feasible lattice point has been produced by the bound contraction process. At this point, the computation may continue in one of two ways, namely,

(i) the new feasible lattice point can be used to construct a new isolated convex and the bound contraction process recommenced;

(ii) any remaining hypotheses (dichotomies) concerning the hyperrectangle defined by the isolated convex can be examined.

The second is associated with the construction of the dichotomies. When the bound contraction process has produced a hyperrectangle defining upper and lower bounds on the integer variables, at least one of which has unequal bounds, then this hyperrectangle must be partitioned into two non-intersecting hyperrectangles. This is achieved by dividing the bounded range on one of the integer variables into two. When a number of variables associated with non-zero ranges exist we choose to dichotomize that one with

(i) the smallest range between its lower and upper bounds;

(ii) the largest range between its bounds;

(iii) the largest range between its bounds and continue with this variable until its range is reduced to zero.

The detailed computational flow of this method is given in Flow Chart 3 but an outline of the basic steps now follows.

(i) Apply the simplex method to the original problem until $a_{oj} \geq 0$, $j \in N$, and $a_{io} \geq 0$, $i \in I$. Note the nonbasic variables and any bounds on the integer variables. Initialize the list count.

(ii) Test the integer variables for feasibility. If $a_{io} \equiv 0$ modulo 1 for all $i \in K$, where K denotes the set of indices of the variables required to be integer valued, then the feasible optimum of this problem is contained in the first column of the tableau; proceed from step (vii). If $a_{io} \not\equiv 0$ modulo 1 for $i \in K' \subset K$, continue from step (iii).

(iii) For each $i \in K'$ compute

$$(a)\begin{cases} u_i = \min\limits_{\substack{a_{ij}<0 \\ j \in N}} (a_{oj} / |a_{ij}|), \\ du_i = (1 - f_{io})\, u_i \ ; \end{cases}$$

$$(b)\begin{cases} v_i = \min\limits_{\substack{a_{ij}>0 \\ j \in N}} (a_{oj}/a_{ij}) \ , \\ dv_i = f_{io}\, v_i \end{cases}$$

Then determine $w_r = \max\limits_{i \in K'} (du_i, dv_i)$.

(iv) If du_r (dv_r) is infinite and the current list count is zero set the upper (lower) bound on x_r at $\left[a_{ro}\right]$ ($\left[a_{ro}\right] + 1$). If du_r (dv_r) is finite, update the list count.

If $w_r = du_r$ (dv_r) replace the upper (lower) bound on x_r with $\lfloor a_{ro} \rfloor$ ($\lfloor a_{ro} \rfloor + 1$) and proceed from step (v).

(v) Apply the dual simplex method. If $a_{io} < 0$ for some $i \in I$ and $a_{ij} \geq 0$, all j, then go to step (vi); otherwise return to step (ii).

(vi) If the list count is zero then the current best solution is the optimum. If the list count is not zero then recover the required nodal tableau and return to step (v).

(vii) At this point the current best feasible solution and possibly bounds on the integer variables have been determined.

At first entry to this step compute the greatest common divisor of a_{oj}, $j \in N$, of the original problem matrix. At subsequent entries this need not be computed.

Use the greatest common divisor of the objective function coefficients together with the current best feasible functional value to determine the functional constraint.

(viii) The nonbasic variables noted in step (i) give the constraints defining the continuous optimal vertex. Determine the points of intersection of these and the functional constraint. Then using these points and the continuous optimal vertex compute the bounds on the integer variables, compare these with any already produced and retain the best.

(ix) Apply the formulae of Section 5.2 to the constraints used in step (viii) and the current set of bounds. This will lead to one of the three following situations.

(a) The bounds have been contracted but at least one integer variable has non-coincident upper and lower bounds. In this case, use the method described in Section 5.4 to partition the hyperrectangle defined by the current set of bounds into two non-intersecting hyperrectangles. Store one of these and update the count associated with the storing of such hypotheses. Use the other and return to the start of step (ix).

(b) The isolated convex defined by these constraints may be empty, in which case the current feasible solution is optimal.

(c) A new lattice point is produced and must be tested for feasibility by application to the original problem matrix. (It need only be applied to a subset of the rows because those associated with nonbasic variables together with those defining the continuous optimal vertex are automatically satisfied). If it is feasible then it necessarily provides an improved solution which must be stored; return to step (vii). If it is infeasible and no hypotheses remain then the current solution is optimal; if hypotheses do remain then select one and recommence step (ix).

Part (c) of step (ix) has given the procedure for the first version of the code. That is, on finding a new feasible lattice point a new isolated convex is constructed and bound contraction recommenced. The

second version would, instead of returning to step (vii), indicate that the current solution is optimal if no hypotheses remain or return to the beginning of step (ix) with a new functional constraint if hypotheses do remain.

The code has been applied to the whole set of 30 problems in the data sample. It has behaved most satisfactorily, its performance comparing most favourably with those of other methods. It can be improved in certain areas of course. For example, if the solution process has to back away from many infeasible problems then this technique is inefficient since it has to recompute earlier nodal tableaux by inverse transformations.

We have applied the five variants previously mentioned and all were successful. The complete sets of results are given in Tables 11 and 12.

7.3 Concluding Remarks

We have presented and discussed in some detail the major techniques for solving integer programming problems. We have dealt most comprehensively with two classes of techniques, namely cutting plane methods and branch and bound methods, and, furthermore, have described some original extensions to methods within each of these classes. In addition, most of these methods have been coded in FORTRAN and the codes described in this chapter. Although our practical experience has been limited to the sample problems defined in Appendix I, many of which, though simple in dimension and formulation, proved difficult for standard cutting plane methods, it has nevertheless provided some quite distinct impressions.

Our first reaction was that only after computer implementation can one hope to appreciate both the advantages and difficulties of the techniques. What seem to be quite straightforward ideas in theory frequently prove to be complex in implementation. It would be most helpful to be able to decide which algorithm to apply to a given integer programming problem in order to obtain its solution most efficiently. In fact, a useful development in integer programming theory would be the determination of classes of problems which can and cannot be readily solved. As is often the case, to the disappointment of investigators and practitioners alike, we cannot say quite definitely that any one approach is uniformly better behaved than any other. However, we do feel a certain degree of confidence when using the branch and bound approach.

Based on our experience, and in spite of not having employed the probably more powerful lexicographical versions of our codes, we tend to agree with the opinion of Haldi [21] regarding the premature discarding of Gomory's Method of Integer Forms in favour of his All Integer Integer Method.

Our conclusions regarding Gomory's algorithms are as follows.

1. The All Integer Integer Method and Method of Integer Forms have proved useful in solving a number of pure integer problems but at the same time they have failed on some very small problems.
2. Both algorithms can be modified in detail in many ways and such modifications have been demonstrated to have a significant effect on performance.

3. Some of the problems attempted by these methods remained unsolved by any of the variants used after a very large number of iterations.
4. The modified version of the All Integer Integer Method was not successful, since any marginal improvements in iteration count were outweighed by the increased time required in determining a stronger cut.
5. The optimal cut method was successful and the results lend support to our above agreement with Haldi's statement concerning the replacement of the Method of Integer Forms by Gomory's All Integer Integer Algorithm.
6. Variations of both extensions of cutting plane methods again significantly affected performance.
7. The optimal cut method is open to further modification with regard to controlling the D number and may therefore be capable of producing even better results in certain circumstances.

The experiments carried out in this project would appear to indicate that branching methods have at least been consistent, in that all problems to which they have been applied have been solved within the limit of the iterative criterion. Our conclusions on such branching methods are

8. The branching heuristic employed in both methods has been very successful and has produced the optimum at an early stage in the computation for a major portion of the data sample.
9. All problems of the data sample were solved by both branching methods in all their variants.

10. The convex isolation and bound contraction method had generally similar success to the branch and bound method and on the problems tried appeared to be faster in most cases. Again a number of variants were applied with the familiar significant changes in performance. We conclude that this method merits further consideration and investigation.

Finally, we note one fairly general observation made during our computing experience. If the constant terms of the constraints defining a convex region are of similar magnitude to the coefficients of the constraints then an integer programming problem defined over such a convex appears to be readily solvable. On the other hand, if the constant terms are relatively large then the solution becomes difficult.

7.4 Detailed Results

To conclude this chapter we present a set of tables relating to the problem sample used in our experiments. The first table defines the structures of these problems and the remaining tables give details of the solutions and the associated performance figures obtained by the application of the appropriate codes, in their different variants, to the data sample.

The problems are defined in Appendix I. The first ten are fixed charge problems and problems 11 to 14 are IBM problems, all taken from [21]. The fixed charge problems are really mixed problems which have been formulated as pure integer problems. Problems 15 to 23 are taken from [34] in which they were presented as very difficult problems for

cutting plane methods. Problems 24 to 30 were constructed during the development of code 7. Problem 25, for example, with an objective function value of 4.2857 at the continuous optimum and a value of 1 at the feasible optimum, admits many infeasible lattice points during the stage when the functional constraint and the constraints determining the continuous optimal vertex are used to define the isolated convex region, and was constructed in order to examine how best to eliminate such infeasibilities.

In Table 1 it is assumed that the problem matrix is in the condensed form of the Tucker tableau presented so as to maximize an objective function over a set of inequality constraints of the less than or equal to type. The table columns are adequately headed but it is perhaps worth pointing out that the degree of dual degeneracy is the ratio between the number of variables not present in the objective function and the total number of variables.

PROBLEM	NUMBER OF VARIABLES	NUMBER OF CONSTRAINTS	DEGREE OF DUAL DEGENERACY	PERCENTAGE NUMBER OF ZERO CONSTRAINT COEFFICIENTS	PERCENTAGE NUMBER OF NEGATIVE CONSTRAINT COEFFICIENTS	RANGE OF MAGNITUDES OF CONSTRAINTS COEFFICIENTS	PERCENTAGE NUMBER OF NEGATIVE CONSTANT TERMS	RANGE OF MAGNITUDES OF CONSTANT TERMS	IS THE PROBLEM DUAL FEASIBLE?
1	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-7	0	0-18	No
2	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-9	0	0-18	No
3	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-9	0	0-21	No
4	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-8	0	0-19	No
5	5	6	2/5	$\frac{1400}{30} = 46.6$	$\frac{200}{30} = 6.6$	0-75	0	0-180	No
6	5	6	2/5	$\frac{1400}{30} = 46.6$	$\frac{200}{30} = 6.6$	0-90	0	0-210	No
7	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-75	0	0-180	No
8	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-90	0	0-210	No
9	6	6	3/6	$\frac{1800}{36} = 50$	$\frac{300}{36} = 8.3$	0-8	0	0-10	No
10	5	4	2/5	$\frac{600}{20} = 30$	$\frac{200}{20} = 10$	0-8	0	0-25	No
11	7	7	0	$\frac{2100}{49} = 42.9$	$\frac{2800}{49} = 57.1$	0-1	100	3-5	Yes
12	7	7	0	$\frac{2100}{49} = 42.9$	$\frac{2800}{49} = 57.1$	0-1	100	2-4	Yes
13	4	3	0	0	100	3-20	100	96-200	Yes
14	15	15	0	$\frac{10500}{225} = 46.6$	$\frac{12000}{225} = 53.3$	0-1	100	3-6	Yes
15	2	2	1/2	0	$\frac{300}{4} = 75$	14-82	100	1306-1430	Yes
16	3	3	1/3	0	$\frac{600}{9} = 66.6$	0-8	$\frac{200}{3} = 66.6$	11-89	Yes
17	2	2	1/2	0	$\frac{200}{4} = 50$	37-85	100	530-1216	Yes
18	3	3	1/3	0	$\frac{600}{9} = 66.6$	2-98	100	530-1668	Yes
19	3	3	1/3	0	$\frac{500}{9} = 55.5$	15-99	100	128-891	Yes
20	2	2	0	0	$\frac{200}{4} = 50$	100-101	50	50-50	Yes
21	2	2	0	0	$\frac{200}{4} = 50$	100-101	100	50-150	Yes
22	2	2	0	0	$\frac{200}{4} = 50$	100-101	100	150-250	Yes
23	8	8	0	$\frac{200}{64} = 3.1$	$\frac{5500}{64} = 85.9$	0-98	100	89-1315	Yes
24	2	2	0	0	0	2-11	0	15-55	No
25	2	3	0	0	$\frac{300}{6} = 50$	1-5	$\frac{100}{3} = 33.3$	3-10	No
26	2	5	0	0	$\frac{500}{10} = 50$	1-6	$\frac{100}{5} = 20$	3-17	No
27	3	3	0	$\frac{200}{9} = 22.2$	0	0-4	0	11-13	No
28	5	2	0	$\frac{100}{10} = 10$	$\frac{200}{10} = 20$	0-4	0	41-47	No
29	2	5	0	0	$\frac{500}{10} = 50$	1-5	$\frac{300}{5} = 60$	2-25	No
30	3	3	0	0	0	1-3	0	8-14	No

TABLE 1. STRUCTURES OF PROBLEMS.

PROBLEM	SOLUTION	NUMBER OF ITERATIONS							
		STANDARD CUT				MODIFIED CUT			
		VARIANT 0	VARIANT 1	VARIANT 2	VARIANT 3	VARIANT 0	VARIANT 1	VARIANT 2	VARIANT 3
11	8;(1,0,0,2,2,1,2,2,0,0,0,0,0,0)	9	38	9	10	9	38	9	10
12	7;(1,0,0,2,2,0,2,3,0,0,0,0,1,1)	15	24	15	13	15	24	15	13
13	187;(0,0,0,17,6,4,18)	8	5	9	5	7	7	8	7
14	10;(0,0,0,0,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,1,1,1,)	28	64	28	23	28	64	28	23
15	70;(7,70,6,40)	18	18	10	10	18	18	10	10
16	18;(3,3,18,2,4,1)	23	22	22	21	23	22	22	21
17	4015;(1754,4015,5,3)	13	13	12	13	13	13	12	13
18	4015;(1754,0,4015,5, 178254,3)	> 2000	27	> 2000	19	> 2000	27	> 2000	19
19	1459;(1233,1134,1459,0,15,13)	1560	1769	1560	1560	1560	1761	1560	1560
20	100;(50,50,0,100)	100	100	100	100	100	100	100	100
21	1650;(150,150,0,100)	234	234	235	234	234	234	235	234
22	2750;(250,250,0,100)	288	288	288	288	288	288	288	288
23	29;(19,1,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	> 2000	> 2000	> 2000	119	> 2000	> 2000	> 2000	120

VARIANT 0: Select the first row with $a_{io} < 0$.

VARIANT 1: Select cyclically.

VARIANT 2: Select the row generating the largest pivot column.

VARIANT 3: Select the row containing the least number of negative elements.

TABLE 2. RESULTS TABLE FOR FOUR GENERATOR VARIANTS OF ALL INTEGER INTEGEF CODE USING STANDARD AND MODIFIED CUTS.

VARIANT CODE		SOLUTION	ITERATIONS	CONSTRAINTS.
0	0	* * *	4 + 100 +	39 +
0	1	(7;1,1,1,6,0,0,2,5,1)	4 + 70	37
0	2	(7;0,1,0,1,6,1,0,0,6)	4 + 21	12
0	3	(7;0,1,0,7,0,1,6,0,0)	4 + 65	30
1	0	* * *	4 + 100 +	43 +
1	1	* * *	4 + 100 +	52 +
1	2	(7;0,1,0,1,6,1,0,0,6)	4 + 53	28
1	3	* * *	4 + 100 +	50 +
2	0	(7;0,1,0,7,0,1,6,0,0)	4 + 22	9
2	1	(7;0,0,0,0,7,4,1,0,0)	4 + 35	15
2	2	(7;0,0,0,0,7,4,1,0,0)	4 + 14	8
2	3	(7;0,0,0,0,7,4,1,0,0)	4 + 32	13
3	0	(7;0,1,0,1,6,1,0,0,6)	4 + 80	52
3	1	(7;0,0,0,0,7,4,1,0,0)	4 + 56	32
3	2	(7;0,1,0,7,0,1,6,0,0)	4 + 46	30
3	3	* * *	4 + 100 +	59 +

Asterisks indicate that the limit of 100 iterations in the integer phase was reached without a solution being obtained.

VARIANT CODE:

First digit defines generator row:-
- 0: Select the row with smallest f_{ik} where the column k is that with the smallest entry in the top row.
- 1: Select the first row with a_{io} non-integral.
- 2: Select the row with largest f_{io}.
- 3: Select the row with smallest f_{io}.

Second digit defines rule for breaking ties when two or more columns tie for pivot column selection:-
- 0: Select the first.
- 1: Select in cyclic order.
- 2: Select at random.
- 3: Select column representing a main variable. If none select randomly.

The iteration count is given as the sum of two numbers; the first gives the number of pivot operations in the continuous phase and the second gives the number in the integer phase. Numbers terminating with a plus sign indicate that a solution was not reached after the indicated number of pivot operations.

TABLE 3. COMBINATION OF VARIANTS OF PURE INTEGER CODE APPLIED TO FIRST FIXED CHARGE PROBLEM.

VARIANT CODE		SOLUTION	ITERATIONS	CONSTRAINTS.
0	0	* * *	4 + 100 +	67 +
0	1	(8;0,0,0,0,8,2,1,0,0)	4 + 21	11
0	2	(8;1,1,4,4,0,1,0,5,3)	4 + 14	12
0	3	(8;0,0,0,0,8,2,1,0,0)	4 + 60	29
1	0	* * *	4 + 100 +	71 +
1	1	(8;0,0,0,0,8,2,1,0,0)	4 + 45	28
1	2	(8;0,0,0,0,8,2,1,0,0)	4 + 22	11
1	3	(8;1,1,3,5,0,0,1,6,2)	4 + 89	62
2	0	(8;1,1,4,4,0,1,0,5,3)	4 + 12	8
2	1	(8;0,0,0,0,8,2,1,0,0)	4 + 10	5
2	2	(8;0,0,0,0,8,2,1,0,0)	4 + 10	5
2	3	(8;0,0,0,0,8,2,1,0,0)	4 + 13	7
3	0	(8;1,1,3,4,1,0,0,6,3)	4 + 91	65
3	1	* * *	4 + 100 +	56 +
3	2	* * *	4 + 100 +	61 +
3	3	(8;0,0,0,0,8,2,1,0,0)	4 + 95	71

The footnotes of Table 3 also apply to this table.

TABLE 4. COMBINATION OF VARIANTS OF PURE INTEGER CODE APPLIED TO SECOND FIXED CHARGE PROBLEM.

PROBLEM	DIMENSIONS CONSTRAINTS	DIMENSIONS VARIABLES	SOLUTIONS CONTINUOUS	SOLUTIONS INTEGER	CONTINUOUS PHASE ITERATIONS	INTEGER PHASE OBJECTIVE AS GENERATOR ADMISSIBLE ITERATIONS	INTEGER PHASE OBJECTIVE AS GENERATOR ADMISSIBLE CONSTRAINTS	INTEGER PHASE OBJECTIVE AS GENERATOR INADMISSIBLE ITERATIONS	INTEGER PHASE OBJECTIVE AS GENERATOR INADMISSIBLE CONSTRAINTS
1	4	5	1603;(93,150,558,1050,0,0,0,0,0)/183	7;(0,1,0,1,6,1,0,0,6) 7;(0,1,0,1,6,1,0,0,6)	4	21	12	17	13
2	4	5	248;(127,191,1143,1337,0,0,0,0,0,)/258	8;(1,1,4,4,0,1,0,5,3) 8;(1,1,4,4,0,1,0,5,3)	4	14	12	21	16
3	4	5	3780;(210,210,1890,1890,0,0,0,0,0)/320	10;(0,0,0,0,10,1,1,0,0) 10;(0,0,0,0,10,1,1,0,0)	4	39	23	39	25
4	4	5	1890;(95,165,570,1320,0,0,0,0,0)/205	8;(0,1,0,3,5,0,0,0,5) 8;(0,1,0,3,5,0,0,0,5)	4	9	5	10	8
5	6	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,22,52,2,0,0,38,23,0,0) 76;(1,1,24,52,0,2,0,36,23,0,0)	4	1427	847	69	64
6	6	5	3780;(21,21,1890,1890,0,0,0,0,0,11,11)/32	106;(1,1,52,52,2,0,0,38,38,0,0) 106;(1,1,54,52,0,2,0,36,38,0,0)	4	76	74	112	109
7	4	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	* * * 76;(1,1,24,52,0,2,0,36,23)	4	>2000	>1212	639	409
8	4	5	3780;(21,21,1890,1890,0,0,0,0,0)/32	106;(1,1,54,52,0,2,0,36,38) 106;(1,1,52,54,0,0,2,38,36)	4	85	84	510	500
9	6	6	24;(1,1,1,3,3,3,8,0,0,0,0,0,0)/2	9;(1,1,1,3,3,3,0,0,0,5,5,5) 9;(1,1,1,3,3,3,0,0,0,5,5,5)	6	552	340	813	497
10	4	5	2390;(285,85,1710,680,0,0,0,0,0)/205	11;(1,0,6,0,5,1,0,0,0) 11;(1,0,6,0,5,1,0,0,0)	4	4	2	10	8
11	7	7	240;(16,16,16,48,48,48,48,0,0,0,0,0,0,0)/32	8;(0,0,1,1,2,2,2,0,0,2,0,0,0,0) 8;(1,1,0,1,2,2,1,0,0,0,2,0,0,0)	8	1	1	6	3
12	7	7	184;(8,8,8,40,40,40,40,0,0,0,0,0,0,0)/32	7;(1,1,0,1,1,2,1,0,1,0,2,1,0,1) 7;(1,0,0,2,2,0,2,3,0,0,0,0,1,1)	8	17	6	22	10
13	3	4	12944;(48;0,0,1120,0,0,1096)/72	187;(0,0,0,17,6,4,18) 187;(0,0,0,17,6,4,18)	2	15	9	19	13
14	15	15	9472;(0,0,0,0,763,768,768,768,768,768,1024,1024,1024,1024,763,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1024)/1024	10;(0,0,0,0,1,1,1,1,1,0,1,1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,1,1,1) 10;(0,0,0,0,1,1,1,1,1,1,1,1,0,1,1,0,1,0,0,0,1,1,0,0,1,1,1,0,1,0)	16	25	12	34	13
15	2	2	50864;(5439,50864,0,0)/749	70;(7,70,6,40) 70;(7,70,6,40)	2	35	34	46	49
16	3	3	1321;(121,79,1321,0,0,0)/90	18;(3,3,18,2,4,1) 18;(3,3,18,2,4,1)	4	26	20	32	25
17	2	2	147880;(64602,147880,0,0)/37	4015;(1754,4015,5,3) 4015;(1754,4015,5,3)	2	11	8	23	19
18	3	3	147880;(64602,0,147880,0,6565040,0)/37	4015;(1754,0,4015,5,178254,3) 4015;(1754,0,4015,5,178254,3)	4	11	8	23	19
19	3	3	7845183;(6629388,6096691,7845183,0,0,0)/5478	1459;(1233,1134,1459,0,15,13) 1459;(1233,1134,1459,0,15,13)	3	52	41	451	394
20	2	2	50;(0,50,0,50)/101	100;(50,50,0,100) 100;(50,50,0,100)	1	199	100	1594	1541
21	2	2	220650;(20050,20150,0,0)/201	1650;(150,150,0,100) 1650;(150,150,0,100)	2	303	205	1595	1542
22	2	2	442650;(40150,40250,0,0)/201	2750;(250,250,0,100) 2750;(250,250,0,100)	2	296	200	1671	1619
23	8	8	187942;(126816,0,0,0,0,0,61126,0,7466348,5163386,0,12211496,163712,1307012,0,324938)/6522	29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49) 29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	9	37	23	25	24
24	2	2	490;(110,40,0,0)/28	16;(4,1,2,4) 16;(4,1,2,4)	2	8	6	7	6
25	3	2	30;(13,9,0,31,0)/7	1;(1,2,4,3,1) 1;(1,2,4,3,1)	2	3	2	5	4
26	5	2	137;(31,13,0,0,5,23,5)/7	13;(3,1,6,2,1,1,3) 13;(3,1,6,2,1,1,3)	3	5	3	5	3
27	3	3	194;(18,23,7,0,0,0)/10	19;(2,2,1,0,1,0) 19;(2,2,1,0,1,0)	3	1	1	1	1
28	2	5	639;(0,258,0,123,0,0,0)/6	106;(0,42,0,19,3,0,0) 106;(0,42,0,19,3,0,0)	3	1	1	2	2
29	5	2	142;(51,42,21,0,63,0,18)/12	7;(2,3,2,6,2,3,1) 7;(2,3,2,6,2,3,1)	3	9	5	9	5
30	3	3	79;(2,25,0,69,0,0)/7	10;(0,3,1,8,1,0) 10;(0,3,1,8,1,0)	2	7	5	8	6

Variant (0,2) indicates that (i) of the rows $i = i_1,\ldots,n$, the code selects as generator that row with $\min f_{ik}$, where column k has the smallest entry in the top row of the tableau;

(ii) the code uses the random choice rule for breaking ties when two or more columns qualify as pivot column.

(1) The DIMENSIONS columns define the number of inequality constraints and the number of variables, excluding slack variables, in the original problem.

(2) In the SOLUTIONS columns the first number is associated with the objective function value.

The continuous solutions are given in terms of numerators and a common denominator, which appears last.

(3) The feasible solutions produced by the application of each defined generator set are given in the INTEGER column.

(4) Asterisks indicate that no solution was obtained within the restriction of 2000 integer phase iterations.

(5) The iteration count (number of pivot operations) has been divided into the number required to reach the continuous optimum together with the number further required to reach the feasible optimum from the continuous optimum.

(6) The CONSTRAINTS columns give the number of cutting planes added to the original constraint set in the integer phase.

(7) The search for a generator from which to construct a cutting plane is confined to the rows $i = i_1,\ldots,n$ where $i_1 = 0$ if the objective function is allowed as a generator and $i_1 = 1$ if the objective function is not allowed as a generator.

TABLE 5. RESULTS FROM PURE INTEGER CODE USING VARIANT (0,2).

PROBLEM	DIMENSIONS		SOLUTIONS		CONTINUOUS PHASE	INTEGER PHASE			
						OBJECTIVE AS GENERATOR			
						ADMISSIBLE		INADMISSIBLE	
	CONSTRAINTS	VARIABLES	CONTINUOUS	INTEGER	ITERATIONS	ITERATIONS	CONSTRAINTS	ITERATIONS	CONSTRAINTS
1	4	5	1608;(93,150,558,1050,0,0,0,0,0)/183	7;(0,0,0,0,7,4,1,0,0) 7;(0,1,0,7,0,1,6,0,0)	4	14	8	19	13
2	4	5	2480;(127,191,1143,1337,0,0,0,0,0)/258	8;(1,1,3,4,1,0,0,6,3) 8;(0,0,0,0,8,2,1,0,0)	4	15	9	15	11
3	4	5	3780;(210,210,1890,1890,0,0,0,0,0)/320	10;(0,0,0,0,10,1,1,0,0) 10;(0,0,0,0,10,1,1,0,0)	4	17	10	27	14
4	4	5	1890;(95,165,570,1320,0,0,0,0,0)/205	8;(0,1,0,3,5,0,0,0,5) 8;(0,1,0,3,5,0,0,0,5)	4	4	3	43	21
5	6	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,24,52,0,2,0,36,23),0,0) 76;(1,1,23,52,2,0,0,33,23),0,0)	4	851	441	451	238
6	6	5	3780;(21,21,1890,1890,0,0,0,0,0,11,11)/32	106;(1,1,54,52,0,2,0,36,38,0,0) 106;(1,1,52,52,2,0,0,38,38,0,0)	4	555	331	749	428
7	4	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,23,52,2,0,0,38,23) 76;(1,1,24,52,0,2,0,36,23)	4	1858	1007	1562	864
8	4	5	3780;(21,21,1890,1890,0,0,0,0,0)/32	" 106;(1,1,52,54,0,0,2,38,36)	4	>2000	>1264	873	565
9	6	6	24;(1,1,1,8,8,8,0,0,0,0,0,0)/7	9;(1,1,1,3,3,3,0,0,0,5,5,5) "	6	1916	770	>2000	>976
10	4	5	2390;(285,85,1710,680,0,0,0,0,0)/205	11;(1,0,6,0,5,1,0,0,0) 11;(1,0,6,0,5,1,0,0,0)	4	3	1	6	5
11	7	7	240;(16,16,16,48,48,48,48,0,0,0,0,0,0,0)/32	8;(0,0,1,1,2,2,2,0,0,2,0,0,0,0) 8;(0,1,1,2,2,1,1,0,0,0,0,0,2,0)	8	1	1	2	2
12	7	7	184;(8,8,8,40,40,40,40,0,0,0,0,0,0,0)/32	7;(0,0,0,2,2,1,2,2,1,1,0,0,1,0) 7;(0,0,1,1,1,2,2,0,1,2,0,1,0,1)	8	13	6	27	12
13	3	4	12944;(48,0,0,1120,0,0,1096)/72	187;(0,0,0,17,6,4,18) 187;(0,0,0,17,6,4,18)	2	28	17	91	62
14	15	15	9472;(0,0,0,0,768,768,768,768,768,768,1024,1024, 1024,1024,768,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1024) /1024	10;(0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,0, 1,0,1,0,1,0,1) 10;(0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,0,0,0,1,1,1,0,1, 0,0,0,1,1,1,0)	16	235	51	84	20
15	2	2	50864;(5439,50864,0,0)/749	" "	2	>2000	>1985	>2000	>1985
16	3	3	1321;(121,79,1321,0,0,0)/90	18;(3,3,18,2,4,1) 18;(3,3,18,2,4,1)	4	55	37	262	216
17	2	2	147880;(64602,147880,0,0)/37	4015;(1754,4015,5,3) 4015;(1754,4015,5,3)	2	5	4	15	12
18	3	3	147880;(64602,0,147880,0,6566040,0)/37	4015;(1754,0,4015,5,178254,3) 4015;(1754,0,4015,5,178254,3)	4	5	4	15	12
19	3	3	7845183;(6629388,6096691,7845183,0,0,0)/6478	" "	3	>2000	>1870	>2000	>1870
20	2	2	50;(0,50,0,50)/101	100;(50,50,0,100) 100;(50,50,0,100)	1	199	100	820	791
21	2	2	220650;(20060,20150,0,0)/201	1650;(150,150,0,100) 1650;(150,150,0,100)	2	322	245	760	733
22	2	2	442650;(40150,40250,0,0)/201	3750;(250,250,0,100) 3750;(250,250,0,100)	2	306	239	760	733
23	8	8	137942;(126816,0,0,0,0,0,61126,0,72663.8, 5163396,0,17211496,1637?,3307012,0,324938)/6472	29;(19,1,0,0,0,0,0,9,0,1109,793,1,1596,65,486,1,49) 29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	9	64	60	64	60
24	2	2	490;(110,40,0,0)/78	16;(4,1,2,4) 16;(4,1,2,4)	2	5	4	8	7
25	3	2	30;(13,9,0,3),0)/7	1;(1,2,4,3,1) 1;(1,2,4,3,1)	2	3	2	5	4
26	5	2	137;(31,13,0,0,5,23,5)/7	13;(3,1,6,2,1,1,3) 13;(3,1,6,2,1,1,3)	3	5	3	5	3
27	3	3	194;(18,23,7,0,0,0)/10	19;(2,2,1,0,1,0) 19;(2,2,1,0,1,0)	3	1	1	1	1
28	2	2	639;(0,258,0,123,0,0,0)/6	106;(0,42,0,19,1,0,0) 106;(0,42,0,19,1,0,0)	3	1	1	2	2
29	3	2	144;(61,40,21,0,63,0,18)/12	7;(2,1,1,6,2,1,1) 7;(2,1,2,6,2,1,1)	3	7	4	7	5
30	3	3	79;(2,25,0,69,0,0)/7	10;(0,1,1,8,1,0) 10;(0,1,1,8,1,0)	2	6	5	6	5

Variant (2,2) indicates that (i) of the rows with non-integer a_{io}, $i = i_1, \ldots, i_k$, the code selects as generator that row with max. f_{io}.

(ii) the code uses the random choice rule for breaking ties when two or more columns qualify as pivot column.

The footnotes (1) to (3) of Table 5 also apply to this table.

TABLE 6. RESULTS FROM FIVE INTEGER CODE USING VARIANT (2,2).

PROBLEM	DIMENSIONS		SOLUTIONS		CONTINUOUS PHASE	INTEGER PHASE			
						OBJECTIVE AS GENERATOR			
						ADMISSIBLE		INADMISSIBLE	
	CONSTRAINTS	VARIABLES	CONTINUOUS	INTEGER	ITERATIONS	ITERATIONS	CONSTRAINTS	ITERATIONS	CONSTRAINTS
1	4	5	1608;(93,150,558,1050,0,0,0,0,0)/183	7;(0,1,0,1,6,1,0,0,6) 7;(0,1,0,1,6,1,0,0,6)	4	4	2	10	8
2	4	5	2480;(127,191,1143,1337,0,0,0,0,0)/258	8;(0,0,0,0,8,2,1,0,0) 8;(0,0,0,0,8,2,1,0,0)	4	14	9	35	22
3	4	5	3780;(210,210,1890,1890,0,0,0,0,0)/320	10;(0,0,0,0,10,1,1,0) 10;(0,0,0,0,10,1,1,0)	4	15	10	21	16
4	4	5	1890;(95,165,570,1320,0,0,0,0,0)/205	8;(0,1,0,1,5,0,0,0,5) 8;(0,1,0,1,5,0,0,0,5)	4	3	2	10	8
5	6	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,22,52,0,2,0,38,23,0,0) 76;(1,1,22,52,2,0,0,38,23,0,0)	4	393	241	64	60
6	6	5	3780;(21,21,1890,1890,0,0,0,0,0,11,11)/32	106;(1,1,52,52,2,0,0,38,38,0,0) 106;(1,1,52,52,2,0,0,38,38,0,0)	4	76	74	78	76
7	4	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,22,52,2,0,0,38,23) 76;(1,1,22,52,2,0,0,38,23)	4	553	339	63	61
8	4	5	3780;(21,21,1890,1890,0,0,0,0,0)/32	106;(1,1,52,52,2,0,0,38,38) 106;(1,1,52,52,2,0,0,38,38)	4	77	77	80	80
9	6	6	24;(1,1,1,8,8,8,0,0,0,0,0,0)/2	9;(1,1,1,3,3,3,0,0,0,5,5,5) 9;(1,1,1,3,3,3,0,0,0,5,5,5)	6	1810	988	37	28
10	4	5	2390;(285,35,1710,680,0,0,0,0,0)/205	11;(1,0,6,0,5,1,0,0,0) 11;(1,0,6,0,5,1,0,0,0)	4	4	2	10	8
11	7	7	240;(16,16,16,48,48,48,48,0,0,0,0,0,0,0)/32	8;(0,0,1,1,2,2,2,0,0,2,0,0,0,0) 8;(1,1,0,1,2,2,1,0,0,0,2,0,0,0)	8	1	1	6	3
12	7	7	184;(8,8,8,40,40,40,40,0,0,0,0,0,0,0)/32	7;(1,1,0,1,1,2,1,0,1,0,2,1,0,1) 7;(1,0,0,2,2,0,2,1,0,0,0,0,1,1)	8	17	6	19	8
13	3	4	12944;(48,0,0,1120,0,0,1096)/72	187;(0,0,0,17,6,4,18) 187;(0,0,0,17,6,4,18)	2	17	9	21	12
14	15	15	9472;(0,0,0,0,768,768,768,768,768,768,768,1024,1024,1024,1024,768,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1024)/1024	10;(0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,1,1,1,0,0,1) 10;(0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,0,0,0,0,1,1,1,0,0,1)	16	13	7	26	11
15	2	2	50864;(5439,50864,0,0)/749	70;(7,70,6,40) 70;(7,70,6,40)	2	33	32	40	39
16	3	3	1321;(121,79,1321,0,0,0)/10	18;(3,3,18,2,2,1) 18;(3,3,18,2,2,1)	4	3	3	13	11
17	2	2	147880;(64602,147880,0,0)/37	4015;(1754,4015,5,3) 4015;(1754,4015,5,3)	2	2	2	2	2
18	3	3	147880;(64602,0,147880,0,6565040,0)/37	4015;(1754,0,4015,5,178254,3) 4015;(1754,0,4015,5,178254,3)	4	2	2	12	10
19	3	3	7845183;(6629388,6096691,7845183,0,0,0,0)/5478	1459;(1233,1134,1459,0,15,13) 1459;(1233,1134,1459,0,15,13)	3	30	27	19	12
20	2	2	50;(0,50,0,50)/101	100;(50,50,0,0,100) 100;(50,50,0,0,100)	1	199	100	54	53
21	2	2	220650;(20050,20150,0,0)/201	1650;(150,150,0,100) 1650;(150,150,0,100)	2	87	62	1	1
22	2	2	447650;(40150,40250,0,0)/201	2750;(250,250,0,100) 2750;(250,250,0,100)	2	84	60	101	101
23	8	8	187942;(126816,0,0,0,0,0,61126,0,7466348,5163386,0,12711496,163712,3307012,0,324938)/6527	23;(19,1,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49) 23;(19,1,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	9	37	22	24	23
24	2	2	490;(110,40,0,0)/28	16;(4,1,2,4) 16;(4,1,2,4)	2	4	3	2	2
25	3	2	30;(11,9,0,31,0)/7	1;(1,2,4,3,1) 1;(1,2,4,3,1)	2		2	3	2
26	5	2	137;(31,13,0,0,5,23,5)/7	13;(3,1,6,2,1,1,3) 13;(3,1,6,2,1,1,3)	3	5	3	5	3
27	3	3	194;(18,23,7,0,0,0)/10	19;(2,2,1,0,1,0) 19;(2,2,1,0,1,0)	3	1	1	1	1
28	2	5	639;(0,258,0,123,0,0,0)/6	106;(0,42,0,19,3,0,0) 106;(0,42,0,19,3,0,0)	3	1	1	1	1
29	5	2	144;(51,42,21,0,61,0,18)/12	7;(2,3,2,6,2,2,1) 7;(2,3,2,6,2,3,1)	3	9	5	9	5
30	3	3	79;(2,26,0,69,0,0)/7	10;(0,3,1,8,1,0) 10;(0,3,1,8,1,0)	2	3	3	3	3

47

'Variant' (C_r) indicates that (i) of the rows with non-integer a_{ic}, $i = 1,\ldots,n$, the code selects as generator that row with min f_{ik}, where column k has the smallest entry in the top row of the tableau;

(ii) the code uses the random choice rule for breaking ties when two or more columns qualify as pivot column.

Footnotes (1) to (7) of Table 5 also apply to this table.

TABLE 7. RESULTS FROM OPTIMUM CUT ROUTINE USING VARIANT (C_r).

PROBLEM	DIMENSIONS		SOLUTIONS		CONTINUOUS PHASE	INTEGER PHASE			
						OBJECTIVE AS GENERATOR			
						ADMISSIBLE		INADMISSIBLE	
	CONSTRAINTS	VARIABLES	CONTINUOUS	INTEGER	ITERATIONS	ITERATIONS	CONSTRAINTS	ITERATIONS	CONSTRAINTS
1	4	5	1608;(93,150,558,1050,0,0,0,0,0)/183	7;(0,1,0,1,6,1,0,0,6) 7;(0,1,0,1,6,1,0,0,6)	4	8	6	10	8
2	4	5	2408;(127,191,1143,1337,0,0,0,0,0)/258	8;(1,1,3,4,1,0,0,6,3) 8;(1,1,3,5,0,0,1,6,2)	4	15	9	24	15
3	4	5	3780;(210,210,1890,1890,0,0,0,0,0)/320	10;(0,0,0,0,10,1,1,0,0) 10;(0,0,0,0,10,1,1,0,0)	4	9	5	11	6
4	4	5	1890;(95,165,570,1200,0,0,0,0,0)/205	8;(0,1,0,3,5,0,0,0,5) 8;(0,1,0,3,5,0,0,0,5)	4	4	3	9	8
5	6	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,24,52,0,2,0,36,23,0,0) 76;(1,1,22,52,2,0,0,38,23,0,0)	4	837	433	395	194
6	6	5	3780;(21,21,1890,1890,0,0,0,0,0,11,11)/32	106;(1,1,54,52,0,2,0,36,38,0,0) 106;(1,1,52,54,0,0,2,38,36,0,0)	4	555	331	29	22
7	4	5	17190;(99,150,5940,11250,0,0,0,0,0)/194	76;(1,1,22,52,2,0,0,38,23) 76;(1,1,22,52,2,0,0,38,23)	4	1541	815	224	120
8	4	5	3780;(21,21,1890,1890,0,0,0,0,0)/32	106;(1,1,52,54,0,0,2,38,36) 106;(1,1,55,50,0,5,0,35,0)	4	858	512	60	52
9	6	6	24;(1,1,1,8,8,8,0,0,0,0,0,0)/2	9;(1,1,1,3,3,3,0,0,0,5,5,5) 9;(1,1,1,3,3,3,0,0,0,5,5,5)	6	723	260	40	20
10	4	5	2390;(285,85,1710,680,0,0,0,0,0)/205	11;(1,0,6,0,5,1,0,0,0) 11;(1,0,5,0,6,0,0,1,0)	4	3	1	6	5
11	7	7	240;(16,16,16,48,48,48,48,0,0,0,0,0,0,0)/32	8;(0,0,1,1,2,2,2,0,0,2,0,0,0,0) 8;(0,1,1,2,2,1,1,0,0,0,0,2,0)	8	1	1	2	2
12	7	7	184;(8,8,8,40,40,40,40,0,0,0,0,0,0,0)/32	7;(0,0,0,2,2,1,2,2,1,1,0,0,1,0) 7;(0,0,1,0,2,2,2,0,0,3,1,0,0,1)	8	13	6	16	8
13	3	4	12944;(48,0,0,1120,0,0,1096)/72	187;(0,0,0,17,6,4,18) 187;(0,0,0,17,6,4,18)	2	27	16	14	10
14	15	15	9472;(0,0,0,0,768,768,768,768,768,768,1024,1024,1024,1024,768,0,0,0,0,0,0,0,0,0,0,0,0,0,1024)/1024	10;(0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1) 10;(0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,0,0,0,1,1,1,0,1,0,0,0,0,1,1,1,0)	16	235	51	84	20
15	2	2	50864;(5439,50864,0,0)/749	70;(7,70,6,40) 70;(7,70,6,40)	2	2	2	2	2
16	3	3	1321;(121,79,1321,0,0,0)/90	18;(3,3,18,2,4,1) 18;(3,3,18,2,4,1)	4	10	8	10	8
17	2	2	147880;(64602,147880,0,0)/37	4015;(1754,4015,5,3) 4015;(1754,4015,5,3)	2	3	3	3	3
18	3	3	147880;(64602,0,147880,0,5565040,0)/37	4015;(1754,0,4015,5,178254,3) 4015;(1754,0,4015,5,178254,3)	4	24	19		
19	3	3	7845183,(6629388,6096691,7845183,0,0,0,0)/5478	1459;(1233,1134,1459,0,15,13) 1459;(1233,1134,1459,0,15,13)	3	23	19	12	9
20	2	2	50;(0,50,0,50)/101	100;(50,50,0,100) 100;(50,50,0,100)	1	199	100	39	38
21	2	2	220650;(20050,20150,0,0)/201	1650;(150,150,0,100) 1650;(150,150,0,100)	2	4	4	101	101
22	2	2	442650;(40150,40250,0,0)/201	2750;(250,250,0,100) 2750;(250,250,0,100)	2	47	40	101	101
23	8	8	187942;(126816,0,0,0,0,0,0,611126,0,7466348,5163386,0,1221149 96, 1037 12,3307012,0,3249381 /6522	29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49) 29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	9	58	56	58	56
24	2	2	490;(110,40,0,0)/28	16;(4,1,2,4) 16;(4,1,2,4)	2	4	3	2	2
25	3	2	30;(13,9,0,31,0)/7	1;(1,2,4,3,1) 1;(1,2,4,3,1)	2	3	2	3	2
26	5	2	137;(31,13,0,0,5,23,5)/7	13;(3,1,6,2,1,1,3) 13;(3,1,6,2,1,1,3)	3	5	3	5	3
27	3	3	194;(18,23,7,0,0,0)/10	19;(2,2,1,0,1,0) 19;(2,2,1,0,1,0)	3	1	1	1	1
28	2	5	639;(0,258,0,123,0,0,0)/6	106;(0,42,0,19,3,0,0) 106;(0,42,0,19,3,0,0)	3	1	1	2	2
29	5	2	144;(51,42,21,0,63,0,18)/12	7;(2,3,2,6,2,3,1) 7;(2,3,2,6,2,3,1)	3	7	4	7	5
30	3	3	79;(2,25,0,69,0,0,0)/7	10;(0,3,1,8,1,0) 10;(0,3,1,8,1,0)	2	2	1	2	1

Variant (2,2) indicates that (i) of the rows with non-integer a_{io}, $i = i_1,\ldots,n$, the code selects as generator that row with max. f_{ic};

(ii) the code uses the random choice rule for breaking ties when two or more columns qualify as pivot column.

Footnotes (1) to (7) of Table 5 also apply here.

TABLE 8. RESULTS FROM OPTIMUM CUT ROUTINE USING VARIANT (2,2).

Variant 0 / Problem	SOLUTIONS			NUMBER OF ITERATIONS			
				CONT.	INTEGER PHASE.		
	CONTINUOUS	FIRST INTEGER	OPTIMAL INTEGER	PHASE	FIRST INTGR.	OPTML. INTGR.	TOTAL.
1	8.7869;(.5081,.8196,3.0491,5.7377,C,C,C,C,C)	7:(1,1,1,6,C,C,2,5,1)	7:(1,1,1,6,C,C,2,5,1)	4	9	9	64
2	9.6124;(.4922,.7403,4.4302,5.1821,0,0,0,0,0)	8:(1,1,4,4,C,1,C,5,3)	8:(1,1,4,4,C,1,C,5,3)	4	3	3	15
3	11.8125;(.6562,.6562,5.9062,5.9062,0,0,0,0,0)	10:(1,1,5,4,1,1,C,4,5)	10:(1,1,5,4,1,1,C,4,5)	4	7	7	20
4	9.2195;(.4634,.8048,2.7804,6.4390,0,C,C,C,0)	7:(0,C,C,C,7,5,1,C,C)	8:(0,1,C,8,C,C,5,C,C)	4	8	9	12
5	88.6082;(.5103,.7731,30.6185,57.9896,0,C,C,C,C,0,.4896,.2268)	75:(0,0,C,C,0,75,30,C,C,C,1,1)	76:(1,1,23,52,1,1,C,37,23,C,C)	4	3	11	16
6	118.125;(.6562,.6562,59.0625,59.0625,C,C,C,C,C,.3437,.3437)	106:(1,1,52,54,C,C,2,38,36,0,C)	106:(1,1,52,54,C,C,2,38,36,C,C)	4	7	7	13
7	88.6082;(.5103,.7731,30.6185,57.9896,0,C,C,C,C,0)	75:(0,0,C,C,75,30,C,C,0)	76:(1,1,23,52,1,1,C,37,23)	4	3	11	16
8	118.125;(.6562,.6562,59.0625,59.0625,C,C,C,C,C,0)	106:(1,1,53,52,1,1,C,37,38)	106:(1,1,53,52,1, ,0,37,38)	4	7	7	13
9	12.0:(.5,.5,.5,4,4,4,C,C,C,C,C,C)	9:(1,1,1,3,3,3,C,C,C,5,5,5)	9:(1,1,1,3,3,3,C,C,C,5,5,5)	6	1	1	7
10	11.6585;(1.3902,.4146,8.3414,3.3170,C,C,C,C,0)	11:(1,C,6,C,5,1,C,C,C)	11:(1,C,6,C,5,1,C,C,C)	4	3	3	8
11	7.5:(.5,.5,.5,1.5,1.5,1.5,1.5,C,C,C,C,C,C,0)	8:(1,C,C,2,2,1,2,2,C,C,C,C,C,C)	8:(1,C,C,2,2,1,2,2,C,C,C,C,C,C)	8	1	1	2
12	5.75:(.25,.25,.25,1.25,1.25,1.25,1.25,1.25,0,0,C,C,C,0,C)	7:(0,C,C,2,2,1,2,2,1,1,C,C,1,C)	7:(C,C,C,2,2,1,2,2,1,1,C,C,1,C)	8	5	5	26
13	179.7778;(.6666,C,C,15.5555,C,C,15.2222)	194:(C,1,1,15,2,18,58)	187:(C,C,C,17,6,4,18)	2	10	31	43
14	9.25;(0,C,C,C,.75,.75,.75,.75,.75,.75,1,1,1,1,.75,0,C,C,0,C,C,C,C,C,C,C,1)	10:(0,C,C,C,1,1,1,1,1,1,1,1,1,1,0,1,1,0,C,C,C,C,C,C,1,1,1,1,1,1,1,C,C)	10:(0,C,C,C,1,1,1,1,1,1,1,1,1,1,0,1,1,0,C,C,C,C,C,C,1,1,1,1,1,1,1,C,C)	23	10	10	28
15	67.9092;(7.2616,67.9092,C,C)	70:(7,7C,6,4C)	70:(7,7C,6,4C)	2	4	4	5
16	14.6778;(1.3444,.8777,12.6777,C,C,C)	18:(3,3,18,2,4,1)	18:(3,3,18,2,4,1)	4	21	21	25
17	3996.7568;(1746,3996.7567,C,C)	4015:(1754,4015,5,3)	4015:(1754,4015,5,3)	2	26	26	27
18	3996.7568;(1746,C,3996.7567,C,17433.5132,C)	4024:(1758,C,4024,12,178664,C)	4015:(1754,C,4015,5,178254,3)	5	62	66	74
19	1432.1254;(1710.1840,1112.9410,1432.1254,C,C,C)	1459:(1233,1134,1459,0,16,13)	1459:(1233,1134,1459,0,16,13)	3	92	92	93
20	.4950;(0,.4950,C,.495C)	100:(50,50,C,100)	100:(50,50,C,100)	1	199	199	199
21	1097.7612;(99.7612,100.2487,C,C)	1650:(150,150,C,100)	1650:(150,150,C,100)	2	201	201	201
22	2002.2388;(199.7512,200.2487,C,C)	2750:(250,250,C,100)	2750:(250,250,C,100)	2	201	201	201
23	28.8166;(19.4443,0,C,C,C,C,9.3722,0,1144.7942,791.6875,0,1872.3544,25.1015,507.0648,0,49.8218)	29:(19,1,C,C,C,C,9,C,1109,793,1,1896,65,486,1,49)	29:(19,1,C,C,C,C,9,C,1109,793,1,1896,65,486,1,49)	9	4	4	9
24	17.5;(3.9285,1.4285,C,C)	15:(5,C,C,5)	16:(4,1,2,4)	2	5	6	7
25	4.2857;(1.8571,1.2857,0,4.4285,0)	1:(1,2,4,3,1)	1:(1,2,4,3,1)	2	3	3	3
26	19.5714;(4.4285,1.8571,C,C,.7142,3.2857,.7142)	13:(3,1,6,2,1,1,3)	13:(3,1,6,2,1,1,3)	3	5	5	5
27	19.4;(1.8,2.3,.7,C,C,C)	19:(2,2,1,C,1,C)	19:(2,2,1,C,1,C)	3	1	1	2
28	106.5;(0,43.0,C,2C.5,C,C,0)	106:(0,42,C,19,3 ,C,C)	106:(0,42,C,19,3 ,C,C)	3	2	2	2
29	15:(4.25,3.5,1.75,C,5.25,C,1.5)	7:(2,3,2,6,2,3,1)	7:(2,3,2,6,2,3,1)	3	6	6	6
30	11.2857;(.2857,3.5714,C,9.3571,C,C)	10:(C,1,1,8,1,C)	10:(C,3,1,8,1,C)	2	3	3	4

Variant 0 indicates that, in the event of two or more tableau columns qualifying for pivot column selection, the first of the qualifying columns is chosen.

The values of the elements of the continuous solutions are given to four decimal places, the first value, that of the objective function, being rounded.

The iteration count has been separated into the number required in the continuous phase, i.e. the number of pivot operations needed to attain the continuous optimum, and the number required in the integer phase, i.e. the number of pivot operations from the continuous optimum state to end of the solution process.

In the integer phase iteration count the FIRST INTGR. column gives the number of integer phase pivot operations to reach the first feasible solution, the OPTML. INTGR. column gives the number of integer phase pivot operations to reach the feasible optimum and the TOTAL column gives the complete number of integer phase pivot operations which is the number required to reach the optimum and prove it.

TABLE 9. RESULTS OBTAINED FROM APPLICATION OF FIRST VARIANT OF BRANCH AND BOUND CODE.

Variant 2 / Problem	SOLUTIONS			NUMBER OF ITERATIONS			
				CONT.	INTEGER PHASE.		
	CONTINUOUS	FIRST INTEGER	OPTIMAL INTEGER	PHASE	FIRST INTG'L.	OPTML. INTGR.	TOTAL
1	8.7869;(.5081,.8196,3.0401,5.7377,0,0,0,0,0)	9;(1,1,1,6,0,0,2,5,1)	7;(1,1,1,6,0,0,2,5,1)	4	8	8	65
2	9.6124;(.4929,.7403,4.4302,5.1821,0,0,0,0,0)	8;(1,1,4,4,0,1,0,5,3)	8;(1,1,4,4,0,1,0,5,3)	4	3	3	14
3	11.8125;(.6562,.6562,5.9062,5.9062,0,0,0,0,0)	10;(1,1,5,4,1,1,0,4,5)	10;(1,1,5,4,1,1,0,4,5)	4	6	6	21
4	9.2195;(.4634,.8048,2.7804,6.4390,0,0,0,0,0)	7;(0,0,0,0,7,5,1,0,0)	8;(0,1,0,8,0,0,5,0,0)	4	7	10	13
5	88.6082;(.5103,.7731,30.6185,57.9895,0,0,0,0,0,,4896,.2268)	75;(0,0,0,0,75,30,0,0,0,1,1)	76;(1,1,23,52,1,1,0,37,23,0,0)	4	4	12	21
6	118.125;(.6562,.6562,59.0625,59.0625,0,0,0,0,0,,3437,.3437)	106;(1,1,52,54,0,0,2,38,36,0,0)	106;(1,1,52,54,0,0,2,38,36,0,0)	4	6	6	15
7	88.6082,(.5103,.7731,30.6185,57.9896,0,0,0,0,0)	75;(0,0,0,0,75,30,0,0,0)	76;(1,1,23,52,1,1,0,37,23)	4	4	12	18
8	118.125;(.6562,.6562,59.0625,59.0625,0,0,0,0,0)	106;(1,1,52,54,0,0,2,38,36)	106;(1,1,52,54,0,0,2,38,36)	4	6	6	13
9	12.0;(.5,.5,.5,4,4,4,0,0,0,0,0)	9;(1,1,1,3,3,3,0,0,0,5,5,5)	9;(1,1,1,3,3,3,0,0,0,5,5,5)	6	3	3	7
10	11.6585;(1.3902,.4126,8.3814,3.3170,0,0,0,0,0)	11;(1,0,6,0,5,1,0,0,0)	11;(1,0,6,0,5,1,0,0,0)	4	3	3	9
11	7.5;(.5,.5,.5,1.5,1.5,1.5,1.5,0,0,0,0,0,0,0)	8;(1,0,0,2,2,1,2,2,0,0,0,0,0,0)	8;(1,0,0,2,2,1,2,2,0,0,0,0,0,0)	8	1	1	2
12	5.75;(.25,.25,.25,1.25,1.25,1.25,1.25,0,0,0,0,0,0)	7;(0,0,0,2,2,1,2,2,1,1,0,0,1,0)	7;(0,0,0,2,2,1,2,2,1,1,0,0,1,0)	8	9	9	28
13	179.7778;(.6666,0,0,15.5555,0,0,15.2222)	194;(0,0,1,15,2,18,28)	187;(0,0,0,17,6,4,18)	2	10	31	43
14	9.25;(0,0,0,0,.75,.75,.75,.75,.75,.75,1,1,1,1,.75,0,0,0,0,0,0,0,0,0,0,0,0,0,1)	11;(1,0,0,0,1,1,1,2,1,1,0,1,1,1,0,0,0,0,0,3,3,1,1,1,1,0,2,2,0,1)	10;(0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,0,0,0,0,1,1,1,0,1,0,0,0,1,1,1,0)	16	30	49	86
15	67.9092;(7.2616,67.9092,0,0)	71;(8,71,46,13)	70;(7,70,6,40)	2	4	6	6
16	14.6778;(1.3444,.8777,14.6777,0,0,0)	21;(4,5,21,2,11,0)	18;(3,3,18,2,4,1)	4	29	31	35
17	3996.7568;(1746,3996.7567,0,0)	4015;(1754,4015,5,3)	4015;(1754,4015,5,3)	2	29	29	29
18	3996.7568;(1746,0,3996.7567,0,17433.5132,0)	4015;(1754,0,4015,5,178254,3)	4015;(1754,0,4015,5,178254,3)	4	43	43	47
19	1432.1254;(1210.1840,1112.9410,1432.1254,0,0,0)	1459;(1233,1134,1459,0,15,13)	1459;(1233,1134,1459,0,15,13)	3	116	116	123
20	.4950;(0,.4950,0,.4950)	100;(50,50,0,100)	100;(50,50,0,100)	1	199	199	199
21	1097.7612;(99.7612,100.2487,0,0)	1650;(150,150,0,100)	1650;(150,150,0,100)	2	201	201	201
22	2202.2388;(199.7612,200.2487,0,0)	2750;(250,250,0,100)	2750;(250,250,0,100)	2	201	201	201
23	28.8166;(19.4443,0,0,0,0,0,9.3722,0,1164,7942,791.6475,0,1872.3544,25.1015,507.0548,0,47.8213)	29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	29;(19,1,0,0,0,0,0,9,0,1109,793,1,1896,65,486,1,49)	2	4	4	9
24	17.5;(3.9285,1.4285,0,0)	15;(5,0,5,5)	16;(4,1,2,4)	2	5	6	7
25	4.2857;(1.8571,1.2857,0,..285,0)	1;(1,2,4,3,1)	1;(1,2,4,3,.)	2	3	3	3
26	19.5714;(4.4285,1.8571,0,0,.7142,3.2857,.7142)	13;(3,1,6,2,1,1,3)	13;(3,1,6,2,1,1,3)	3	5	5	5
27	19.4;(1.8,2.3,.7,0,0,0)	19;(2,2,1,0,1,0)	19;(2,2,1,0,1,0)	3	1	1	2
28	106.5;(0,43.0,0,20.5,0,0,0)	106;(0,42,0,19,3,0,0)	106;(0,42,0,19,3,0,0)	3	2	2	2
29	12;(4.25,3.5,1.75,0,5.25,0,1.5)	7;(2,3,2,6,2,3,1)	7;(2,3,2,6,2,3,1)	3	6	6	6
30	11.2857;(.2857,3.5714,0,9.8571,0,0)	10;(0,3,1,8,1,0)	10;(0,3,1,8,1,0)	2	3	3	4

Variant 2 indicates that the random selection rule was used to break ties in the event of two or more columns qualifying as pivot columns.

The values of the elements of the continuous solutions are given to four decimal places, the first value being rounded.

The iteration count has been separated into the number required in the continuous phase, i.e. the number of pivot operations necessary to reach the continuous optimum, and those required in the integer phase, i.e. the number of pivot operations taken to reach the solution and to prove its optimality.

The TOTAL count in the integer phase indicates the complete number of integer phase iterations, i.e. the number to the optimum together with the number to prove optimality.

TABLE 10. RESULTS FROM SECOND FRANK AND WOLFE VARIANT.

Version 0 / Problem	SOLUTIONS			ITERATIONS		CONTRACTION CYCLES. VARIANT 0		VARIANT 1		VARIANT 2	
	CONTINUOUS	FIRST INTEGER	OPTIMAL INTEGER	CONT.	F.INT.	A1	B1	A2	B2	A3	B3
1	8.7869;(.5081,.8196,3.0491,5.7377, 0,0,0,0,0)	7;(1,1,1,6,0,0,2,5,1)	7;(1,1,1,6,0,0,2,5,1)	4	9	0	19	0	22	0	25
2	9.6124;(.4922,.7403,4.4302,5.1821, 0,0,0,0,0)	8;(1,1,4,4,0,1,0,5,3)	8;(1,1,4,4,0,1,0,5,3)	4	3	0	0	0	0	0	0
3	11.8125;(.6562,.6562,5.9062,5.9062, 0,0,0,0,0)	10;(1,1,5,4,1,1,0,4,5)	10;(1,1,5,4,1,1,0,4,5)	4	7	0	13	0	11	0	13
4	9.2195;(.4634,.8048,2.7804,6.4390, 0,0,0,0,0)	7;(0,0,0,0,7,5,1,0,0)	8;(0,1,0,3,5,0,0,0,5)	4	8	11	11	13	13	15	15
5	88.6082;(.5103,.7731,30.6815,57.9896, 0,0,0,0,0,.4896,.2268)	75;(0,0,0,0,75,30,0,0,0,1,1)	76;(1,1,22,52,2,0,0,38,23,0,0)	4	3	132	240	23	215	336	528
6	118.125;(.6562,.6562,59.0625,59.0625, 0,0,0,0,0,.3437,.3437)	106;(1,1,53,52,1,1,0,37,38,0,0)	106;(1,1,53,52,1,1,0,37,38,0,0)	4	7	0	90	0	77	0	94
7	88.6082;(.5103,.7731,30.6815,57.9896, 0,0,0,0,0)	75;(0,0,0,0,75,30,0,0,0)	76;(1,1,22,52,2,0,0,38,23)	4	3	132	240	23	215	336	528
8	118.125;(.6562,.6562,59.0625,59.0625, 0,0,0,0,0)	106;(1,1,53,52,1,1,0,37,38)	106;(1,1,53,52,1,1,0,37,38)	4	7	0	90	0	77	0	94
9	12;(.5,.5,.5,4,4,4,0,0,0,0,0)	9;(1,1,1,3,3,3,0,0,0,5,5,5)	9;(1,1,1,3,3,3,0,0,0,5,5,5)	6	3	0	5	0	5	0	5
10	11.6585;(1.3902,.4146,8.3414,3.3170, 0,0,0,0,0)	11;(1,0,6,0,5,1,0,0,0)	11;(1,0,6,0,5,1,0,0,0)	4	1	0	0	0	0	0	0
11	7.5;(.5,.5,.5,1.5,1.5,1.5,1.5,0,0,0, 0,0,0)	8;(1,0,0,2,2,1,2,2,0,0,0,0,0,0)	8;(1,0,0,2,2,1,2,2,0,0,0,0,0,0)	8	1	0	0	0	0	0	0
12	5.75;(.25,.25,.25,1.25,1.25,1.25, 1.25,0,0,0,0,0,0,0)	7;(0,0,0,2,2,1,2,2,1,1,0,0,1,0)	7;(0,0,0,2,2,1,2,2,1,1,0,0,1,0)	8	5	0	0	0	0	0	0
13	179.7778;(.6666,0,0,15.5555,0,0, 15.2222)	194;(0,1,1,15,2,18,28)	187;(0,0,0,17,6,4,18)	9	10	8	12	40	44	52	56
14	9.25;(0,0,0,0,.75,.75,.75,.75,.75, .75,1,1,1,1,1,.75,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,1)	10;(0,0,0,0,0,1,1,1,1,1,1,1,1,1,0, 1,1,0,0,0,0,0,0,0,1,1,1,1,1,1, 0,0)	10;(0,0,0,0,0,1,1,1,1,1,1,1,1,1,0, 1,1,0,0,0,0,0,0,0,1,1,1,1,1,1, 0,0)	23	10	0	0	0	0	0	0
15	67.9092;(7.2616,67.9092,0,0)	70;(7,70,6,40)	70;(7,70,6,40)	9	4	0	0	0	0	0	0
16	14.6778;(1.3444,.8777,14.6778,0,0,0)	18;(3,3,18,2,4,1)	18;(3,3,18,2,4,1)	4	10	0	9	0	9	0	9
17	3996.7568;(1746,3996.7567,0,0)	4015;(1754,4015,5,3)	4015;(1754,4015,5,3)	2	26	0	0	0	0	0	0
18	3996.7568;(1746,0,3996.7567,0, 177433.5132,0)	4024;(1758,0,4024,12,178664,0)	4015;(1754,0,4015,5,178754,3)	5	65	12	15	13	16	13	16
19	1432.1254;(1210.1840,1112.9410, 1432.1254,0,0,0)	1459;(1233,1134,1459,0,15,13)	1459;(1233,1134,1459,0,15,13)	3	55	0	61	0	58	0	59
20	.495;(0,.4950,0,.4950)	100;(50,50,0,100)	100;(50,50,0,100)	1	100	0	0	0	0	0	0
21	1097.7612;(99.7512,100.2487,0,0)	1650;(150,150,0,100)	1650;(150,150,0,100)	9	201	0	0	0	0	0	0
22	2202.2388;(199.7512,200.2487,0,0)	2750;(250,250,0,100)	2750;(250,250,0,100)	9	201	0	0	0	0	0	0
23	28.8166;(19.4443,0,0,0,0,0,0,9.2722,0, 11.44.7927,791.6875,0,1872.354 4,25.1015,509.0548,0,29.8218)	29;(19,1,0,0,0,0,0,9,0,1109,793, 1,1896,65,486,1,49)	29;(19,1,0,0,0,0,0,9,0,1109,793, 1,1896,65,486,1,49)	9	4	0	0	0	0	0	0
24	17.5;(3.9285,1.4285,0,0)	15;(5,0,5,5)	16;(4,1,5,4)	9	5	2	2	9	9	9	9
25	4.2857;(1.8571,1.2857,0,4.4285,0)	1;(1,2,4,3,1)	1;(1,2,4,3,1)	9	1	0	0	0	0	0	0
26	19.5714;(4.2857,1.8571,0,0,.7142, 3.2857,.7142)	13;(3,1,6,2,1,1,3)	13;(3,1,6,2,1,1,3)	9	5	0	0	0	0	0	0
27	19.4;(1.8,2.3,.7,0,0,0)	19;(2,2,1,0,1,0)	19;(2,2,1,0,1,0)	3	1	0	0	0	0	0	0
28	106.5;(0,43,0,20.5,0,0,0)	106;(0,42,0,19,2,0,0)	106;(0,42,0,19,2,0,0)	3	2	0	0	0	0	0	0
29	12;(4.25,3.5,1.75,0,5.25,0,1.5)	7;(2,2,2,6,2,3,1)	7;(2,3,2,6,2,3,1)	9	6	0	0	0	0	0	0
30	11.2857;(.2857,3.5714,0,0.8571,0,0)	10;(0,3,1,8,1,0)	10;(0,3,1,8,1,0)	9	3	0	0	0	0	0	0

(i) Version 0 indicates that we return to construct a new isolated convex each time a new lattice point is produced.

(ii) The continuous solution is given to four places of decimals, the value of the objective function being rounded to the fourth place.

(iii) The ITERATIONS columns give the number of pivot operations required to reach the continuous optimum together with the number required to reach the first feasible solution after attainment of the continuous optimum.

(iv) The CONTRACTION CYCLES columns associated with each VARIANT give the number of cycles required in progressing from the first feasible solution to the optimal solution and then proving that it is the optimum. The columns A1, A2 and A3 give the number of cycles required between the first feasible solution and the optimal solution. Columns B1, B2 and B3 give the number of cycles between the first feasible solution and the termination of the solution process, i.e. the total number of cycles required.

(v) At the stage when the hyperrectangle L, defined by bounds on the integer variables, cannot be contracted further, hypotheses are made which divide L into two non-intersecting hyperrectangles contained in L. This is effected by making a dichotomy on one of the variables whose upper and lower bounds are not coincident. Three variants defining hypotheses are used.

VARIANT 0: Select that variable whose range is smallest.

VARIANT 1: Select that variable whose range is largest.

VARIANT 2: Select that variable whose range is largest and continue to use this variable until its range becomes zero.

TABLE 11. RESULTS FROM THREE VARIANTS OF VERSION 0 OF CONVEX ISOLATION AND BOUND CONTRACTION CODE.

VERSION 1 / PROBLEM	SOLUTIONS			ITERATIONS		CONTRACTION CYCLES. VARIANT 0		VARIANT 1		VARIANT 2	
	CONTINUOUS	FIRST INTEGER	OPTIMAL INTEGER	CONTINUOUS	FIRST INTEGER	OPTIMAL INTEGER	TOTAL	OPTIMAL INTEGER	TOTAL	OPTIMAL INTEGER	TOTAL
4	9.2195;(.4634,.8048, 2.7804,6.4390,0,0,0, 0,0)	7;(0,0,0,0,7,5,1,0,0)	8;(0,1,0,3,5,0,0,0,5) 8;(1,1,2,6,0,2,3,4,2)	4	8	11	23	13	28	15	35
5	88.6082;(.5103,.7731, 30.6815,57.9896,0,0, 0,0,0,.4896,.2268)	75;(0,0,0,0,75,30,0, 0,0,1,1)	76;(1,1,22,52,2,0,0, 38,23,0,0) 76;(1,1,24,52,0,0,4, 36,23,0,0)	4	3	132	147	23	193	336	344
7	88.6082;(.5103,.7731, 30.6815,57.9896,0,0, 0,0,0)	75;(0,0,0,0,75,30,0, 0,0)	76;(1,1,22,52,2,0,0, 38,23)	4	3	132	147	23	193	336	344
13	179.7778;(.6666,0,0, 15.5555,0,0,15.2222)	194;(0,1,1,15,2,18, 28)	187;(0,0,0,17,6,4, 18)	2	10	8	13	22	22	20	20
18	3996.7568;(1746,0, 3996.7567,0,177433. 5132,0)	4024;(1758,0,4024, 12,178664,0)	4015;(1754,0,4015, 5,178254,3)	5	62	12	12	13	13	13	13
24	17.5;(3.9285,1.4285, 0,0)	15;(5,0,5,5)	16;(4,1,2,4)	2	5	2	2	2	2	2	2

(i) Version 1 indicates that when the bound contraction process produces a feasible lattice point a new functional constraint is defined and any remaining hypotheses examined.

The footnotes (ii) through (v) of Table 11 are also applicable here.

TABLE 12. RESULTS FROM THREE VARIANTS OF VERSION 1 OF CONVEX ISOLATION AND BOUND CONTRACTION CODE.

BIBLIOGRAPHY.

[1] Balas, E., "An Additive Algorithm for Solving Linear Programs With Zero-One Variables". Opns. Res., Vol. 13, 1965, 517-546.

[2] Balinski, M.L., "An Algorithm for Finding All Vertices of Convex Polyhedral Sets"; SIAM Journal, Vol.9, 1961, 72-88.

[3] Balinski, M.L., "Integer Programming: Methods, Uses, Computation", Management Science Vol.12, 1965, 253-313.

[4] Beale, E.M L., "Survey of Integer Programming", Opnl. Res.Qu., Vol.16, 1965, 219-228.

[5] Beale, E.M.L. and Small, R.E., "Mixed Integer Programming by a Branch and Bound Technique", Proceedings IFIP Congress 65, Vol.2.

[6] Cooper, L. and Drebes, C., "An Approximate Method For The Fixed Charge Problem". Nav. Res. Log. Qu., Vol.14, 1967, 101-113.

[7] Benders, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol.4, 1962, 238-252.

[8] Dakin, R.J., "A Tree-Search Algorithm for Mixed Integer Programming Problems", Computer Journal, Vol.8, 1965,250-255.

[9] Dantzig, G.B., "Linear Programming and Extensions" Princeton University Press, Princeton, N.J., 1963.

[10] Dantzig, G.B., "Discrete Variable Extremum Problems", Opns. Res., Vol.5, 1957, 266-277.

[11] Dantzig, G.B., "On the Significance of Solving Linear Programming Problems with some Integer Variables", Econometrica, Vol.28, 1960, 30-34.

[12] Dantzig, G.B., "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programs", Econometrica, Vol.23, 1955, 174-183.

[13] Driebeck, Norman J., "An Algorithm for the Solution of Mixed Integer Programming Problems", Man. Sc., Vol.12, 1966, 576-587.

[14] Glover, Fred, "Generalized Cuts in Diophantine Programming", Man. Sc., Vol.13, 1966, 254-268.

[15] Gomory, R.E., "An Algorithm for Integer Solutions to Linear Programs", pp.269-302 in [19].

[16] Gomory, R.E., "All Integer Integer Programming Algorithm", Research Report RC-189 Jan.29, 1960, I.B.M., Yorktown Heights, N.Y.

[17] Gomory, R.E., "An Algorithm for the Mixed Integer Problem" pp.180-184 in [32].

[18] Gomory, R.E., "On the Relation Between Integer and Non-Integer Solutions to Linear Programs", Proc. Nat. Acad.Sc., Vol.53, 1965, 260-265.

[19] Graves, Robert L. and Wolfe, Philip, Editors,"Recent Advances in Mathematical Programming", McGraw-Hill, New York, 1963.

[20] Hadley, G., "Linear Programming", Addison Wesley, Reading, Mass., 1961.

[21] Haldi, John, "25 Programming Test Problems", Working Paper No.43, Graduate School of Business, Stanford University, December 1964.

[22] Hardy, G.H. and Wright, E.M., "An Introduction to The Theory of Numbers", Oxford University Press, 1954.

[23] Harris, P.M.J., "An Algorithm for Solving Mixed Integer Linear Programs", Opnl. Res. Qu., Vol.15, 1964, 117-132.

[24] Healy, W.C. Jr., "Multiple Choice Programming", Opns. Res., Vol.12, 1964, 122-138.

[25] Klee, V., "A Class of Linear Programming Problems Requiring a Large Number of Iterations", Numerische Mathematik, Vol.7, 1965, 313-321.

[26] Land, A.H. and Doig, A.G., "An Automatic Method of Solving Discrete Programming Problems", Econometrica, Vol.28, 1960, 497-520.

[27] Little, John D.C., Murty, Katta G., Sweeney, Dura N. and Karel, Caroline, "An Algorithm for the Travelling Salesman Problem", Opns. Res., Vol.11, 1963, 972-989.

[28] Markowitz, Harry M. and Manne, Alan S., "On the Solution of Discrete Programming Problems", Econometrica, Vol.25, 1957, 84-110.

[29] Martin, Glenn T., "An Accelerated Euclidean Algorithm for Integer Linear Programming", pp.311-318 in [19].

[30] Moroz, A.I., "A Model for Discrete Variable Linear Programs", International Journal of Computer Mathematics, Vol.1, 1964, 1-19.

[31] Reiter, Stanley and Rice, Donald B., "Discrete Optimizing Solution Procedures for Linear and Nonlinear Integer Programming Problems", Man. Sc., Vol.12, 1966, 829-840.

[32] Simmonard, Michel, "Linear Programming", Prentice Hall, Inc., Englewood Cliffs, N.J., 1966.

[33] Srinivasan, A.V., "An Investigation of Some Computational Aspects of Integer Programming", ACM Journal, Vol.12, 1965, 525-535.

[34] Thompson, G.L., "The Stopped Simplex Method: I. Basic Theory for Mixed Integer Programming; Integer Programming", Revue Francaise de Recherche Opérationelle, Vol.8, 1964, 159-182.

[35] Tucker, A.W., "A Combinatorial Equivalence of Matrices", pp.129-140 in [19].

[36] Vajda, S., "Mathematical Programming", Addison Wesley, Reading, Mass., 1961.

[37] Van der Waerden, B.L., "Modern Algebra", Vol.II, Frederick Ungar Publishing Co., New York, 1950.

[38] Wiest, J.D., "The Scheduling of Large Projects With Limited Resources", O.N.R. Res. Memo., 113, May 1962, Graduate School of Indust. Admin. Carn. Inst. Tech., Pitts., Penn.

[39] Young, Richard D., "A Primal (All Integer) Integer Programming Algorithm", Journal of Research of the National Bureau of Standards, Vol.69B, 1965, 213-250.

APPENDIX I

As mentioned in the main body of the thesis the sample of problems was constructed primarily from [21] and [34]. For ease of reference, and in fact for completeness, we define the problems in this appendix.

In all problems variables are required to be non-negative integers.

Problem 1

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
2	3	1	2	2	18
3	2	2	1	2	15
-6	0	1	0	0	0
0	-7	0	1	0	0

Problem 2

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
2	3	1	2	2	18
3	2	2	1	2	17
-9	0	1	0	0	0
0	-7	0	1	0	0

Problem 3

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
2	3	1	2	2	21
3	2	2	1	2	21
-9	0	1	0	0	0
0	-9	0	1	0	0

Problem 4

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
2	3	1	2	2	19
3	2	2	1	2	15
-6	0	1	0	0	0
0	-8	0	1	0	0

Problem 5

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
20	30	1	2	2	180
30	20	2	1	2	150
-60	0	1	0	0	0
0	-75	0	1	0	0
1	0	0	0	0	1
0	1	0	0	0	1

Problem 6

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
20	30	1	2	2	210
30	20	2	1	2	210
-90	0	1	0	0	0
0	-90	0	1	0	0
1	0	0	0	0	1
0	1	0	0	0	1

Problem 7

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
20	30	1	2	2	180
30	20	2	1	2	150
-60	0	1	0	0	0
0	-75	0	1	0	0

Problem 8

Maximize $x_0 = x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leqslant$
20	30	1	2	2	210
30	20	2	1	2	210
-90	0	1	0	0	0
0	-90	0	1	0	0

Problem 9

Maximize $x_0 = x_4 + x_5 + x_6$

subject to

x_1	x_2	x_3	x_4	x_5	x_6	$\leqslant$
2	2	0	1	1	0	10
2	0	2	1	0	1	10
0	2	2	0	1	1	10
-8	0	0	1	0	0	0
0	-8	0	0	1	0	0
0	0	-8	0	0	1	0

Problem 10

Maximize $x_0 = x_3 + x_4 + x_5$

x_1	x_2	x_3	x_4	x_5	$\leqslant$
2	3	1	2	2	19
3	2	2	1	2	25
-6	0	1	0	0	0
0	-8	0	1	0	0

Problems 11 and 12

Minimize $x_0 = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$

subject to

x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\geqslant$
1	0	0	1	1	0	1	5
0	1	0	1	0	1	1	5
0	0	1	0	1	1	1	5
1	1	0	0	1	1	0	4
1	0	1	1	0	1	0	4
0	1	1	1	1	0	0	4
1	1	1	0	0	0	1	3

and

$\geqslant$
4
4
4
3
3
3
2

Problem 13

Minimize $13x_1 + 15x_2 + 14x_3 + 11x_4$

subject to

x_1	x_2	x_3	x_4	$\geqslant$
4	5	3	6	96
20	21	17	12	200
11	12	12	7	101

Problem 14

Minimize $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15}$

subject to

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	$\geqslant$
1	0	0	0	1	1	1	0	0	0	1	1	1	0	1	6
0	1	0	0	1	0	0	1	1	0	1	1	0	1	1	6
0	0	1	0	0	1	0	1	0	1	1	0	1	1	1	6
0	0	0	1	0	0	1	0	1	1	0	1	1	1	1	6
1	1	0	0	0	1	1	1	1	0	0	0	1	1	0	5
1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	5
1	0	0	1	1	1	0	0	1	1	1	0	0	1	0	5
0	1	1	0	1	1	0	0	1	1	0	1	1	0	0	5
0	1	0	1	1	0	1	1	0	1	1	0	1	0	0	5
0	0	1	1	0	1	1	1	1	0	1	1	0	0	0	5
1	1	1	0	0	0	1	0	1	1	1	0	0	0	1	4
1	1	0	1	0	1	0	1	0	1	0	1	0	0	1	4
1	0	1	1	1	0	0	1	1	0	0	0	1	0	1	4
0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	4
1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	3

Problem 15

Minimize x_2

subject to

x_1	x_2	$\geqslant$
66	14	1430
-82	28	1306

Problem 16

Minimize x_3

subject to

x_1	x_2	x_3	$\geqslant$
5	8	-7	-89
-6	5	1	11
3	-5	2	29

Problem 17

Minimize x_2

subject to

x_1	x_2	$\geqslant$
85	-37	530
-84	37	1216

Problem 18

Minimize x_3

subject to

x_1	x_2	x_3	$\geqslant$
85	-74	-37	530
98	13	2	1668
-84	31	37	1216

Problem 19

Minimize x_3

subject to

x_1	x_2	x_3	$\geqslant$
-64	-57	99	891
94	15	-91	128
-75	27	43	867

Problem 20

Minimize $x_1 + x_2$

subject to

x_1	x_2	$\geqslant$
-100	101	50
101	-100	-50

Problem 21

Minimize $10x_1 + x_2$

subject to

x_1	x_2	$\geqslant$
-100	101	150
101	-100	50

Problem 22

Minimize $x_1 + 10x_2$

subject to

x_1	x_2	$\geqslant$
-100	101	250
101	-100	150

Problem 23

Minimize $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8$

subject to

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	$\geq$
74	9	16	51	70	-90	32	51	594
64	64	33	23	31	49	92	22	1315
79	16	94	98	89	39	-54	82	1030
84	74	-82	40	79	51	35	33	89
49	68	89	2	0	97	17	74	1087
45	25	91	87	56	-43	70	14	1024
33	38	1	0	-97	28	60	-63	1204
28	28	15	70	36	-77	44	17	907

Problem 24

Maximize $3x_1 + 4x_2$

subject to

x_1	x_2	$\leq$
2	5	15
10	11	55

Problem 25

Maximize $3x_1 - x_2$

subject to

x_1	x_2	$\leq$
3	-2	3
-5	-4	-10
2	1	5

Problem 26

Maximize $4x_1 + x_2$

subject to

x_1	x_2	$\leq$
3	2	17
2	-1	7
2	-3	4
-1	-1	-3
-2	6	3

Problem 27

Maximize $4x_1 + 5x_2 + x_3$

subject to

x_1	x_2	x_3	$\leq$
3	2	0	10
1	4	0	11
3	3	1	13

Problem 28

Maximize $x_1 + 2x_2 + 3x_3 + x_4 + x_5$

subject to

x_1	x_2	x_3	x_4	x_5	$\leq$
1	0	4	2	1	41
4	3	1	-4	-1	47

Problem 29

Maximize $2x_1 + x_2$

subject to

x_1	x_2	$\leq$
-1	5	15
2	3	19
-1	-2	-6
2	-3	-2
2	-2	-25

Problem 30

Maximize $2x_1 + 3x_2 + x_3$

subject to

x_1	x_2	x_3	$\leq$
2	1	3	14
3	2	1	8
1	3	2	11

APPENDIX II

The flow charts for the four codes

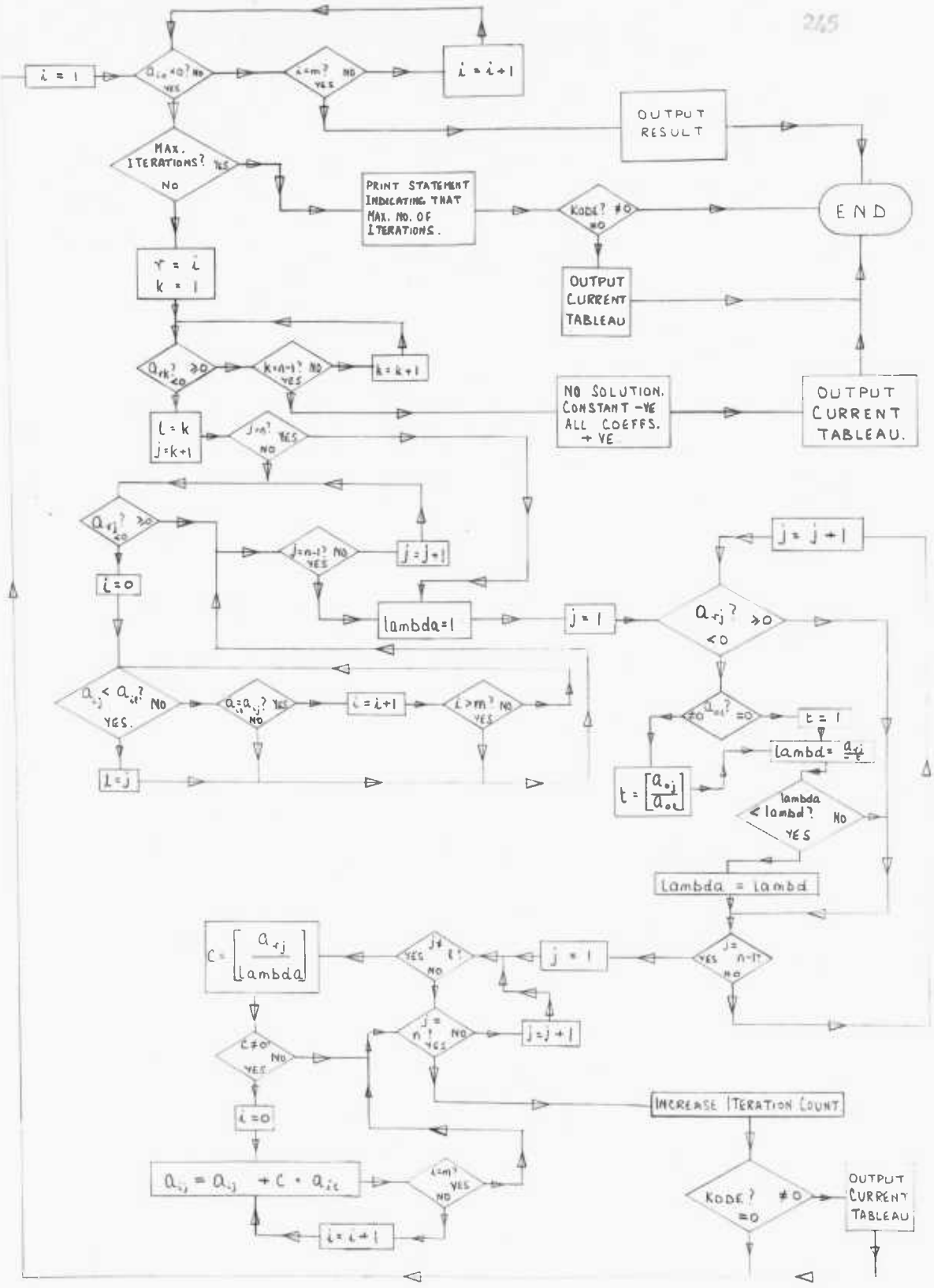
(i) The All Integer Integer Method,

(ii) A Modified Cutting Plane for the All Integer Integer Method,

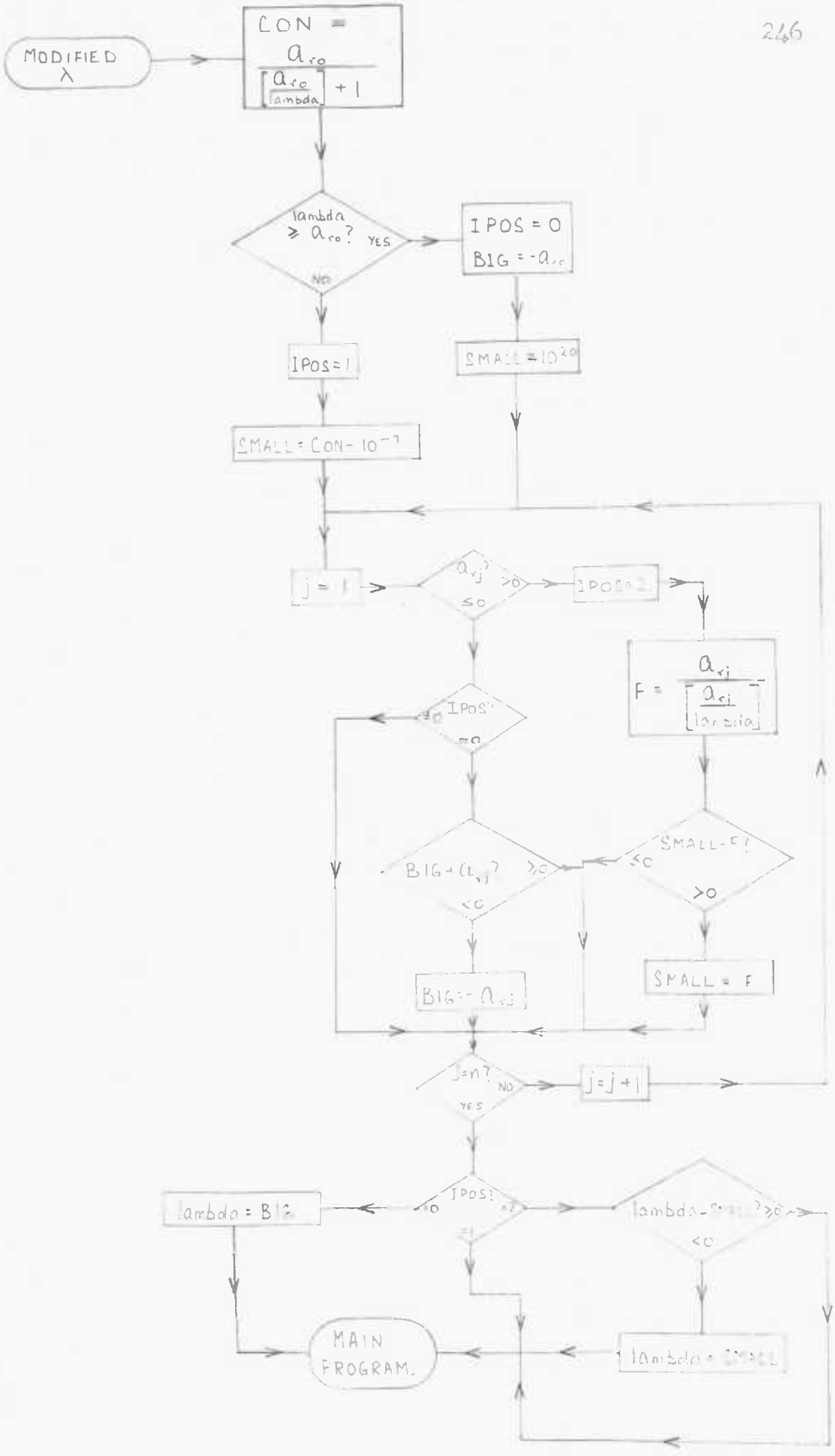
(iii) The Optimal Cut Routine,

and (iv) The Convex Isolation and Bound Contraction Method

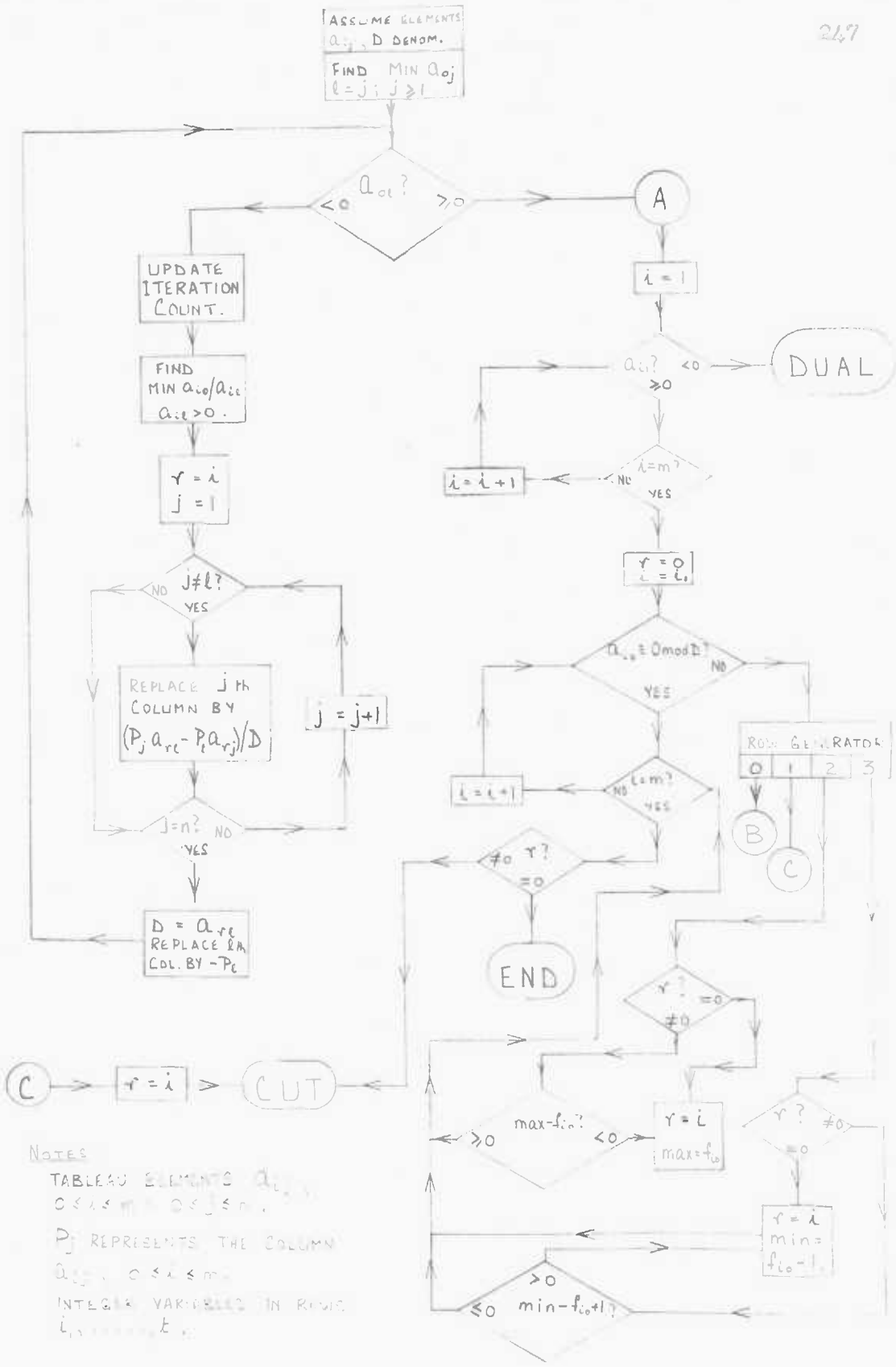
are given in this appendix.



FLOW CHART 1. THE ALL INTEGER INTEGER ALGORITHM.



FLOW CHART 2 — STRONGER CUT FOR THE ALL INTEGER INTEGER ALGORITHM.



NOTES

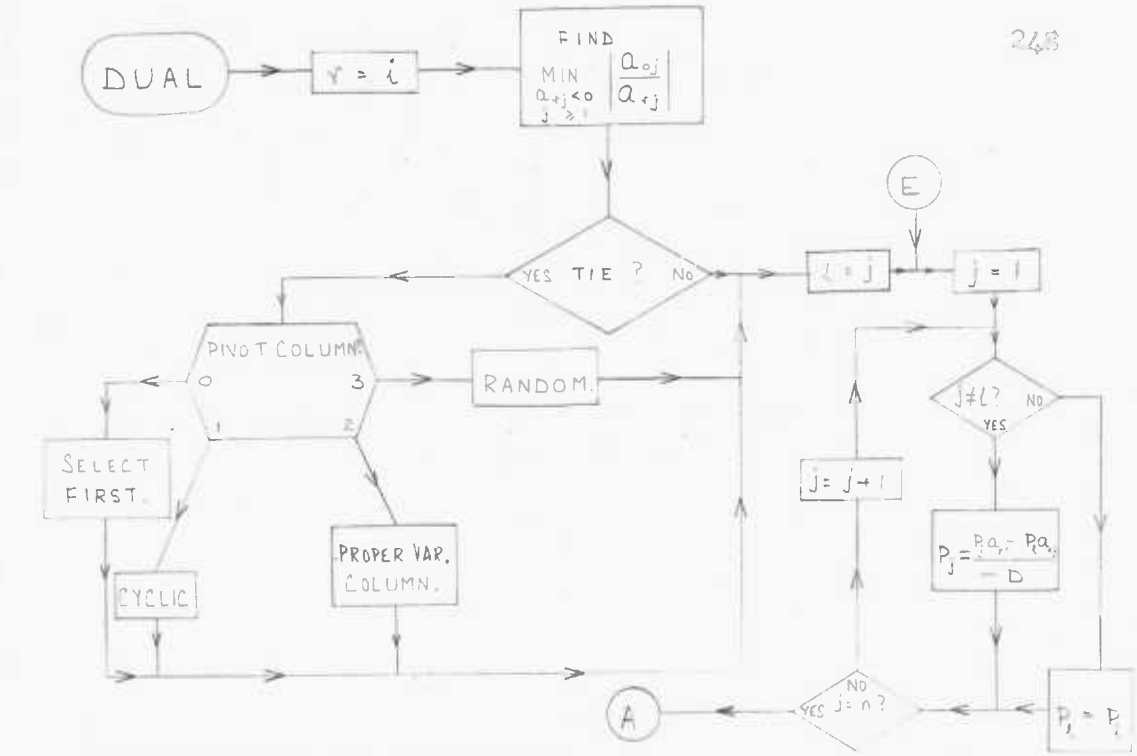
TABLEAU ELEMENTS a_{ij}, $0 \leq i \leq m$, $0 \leq j \leq n$.

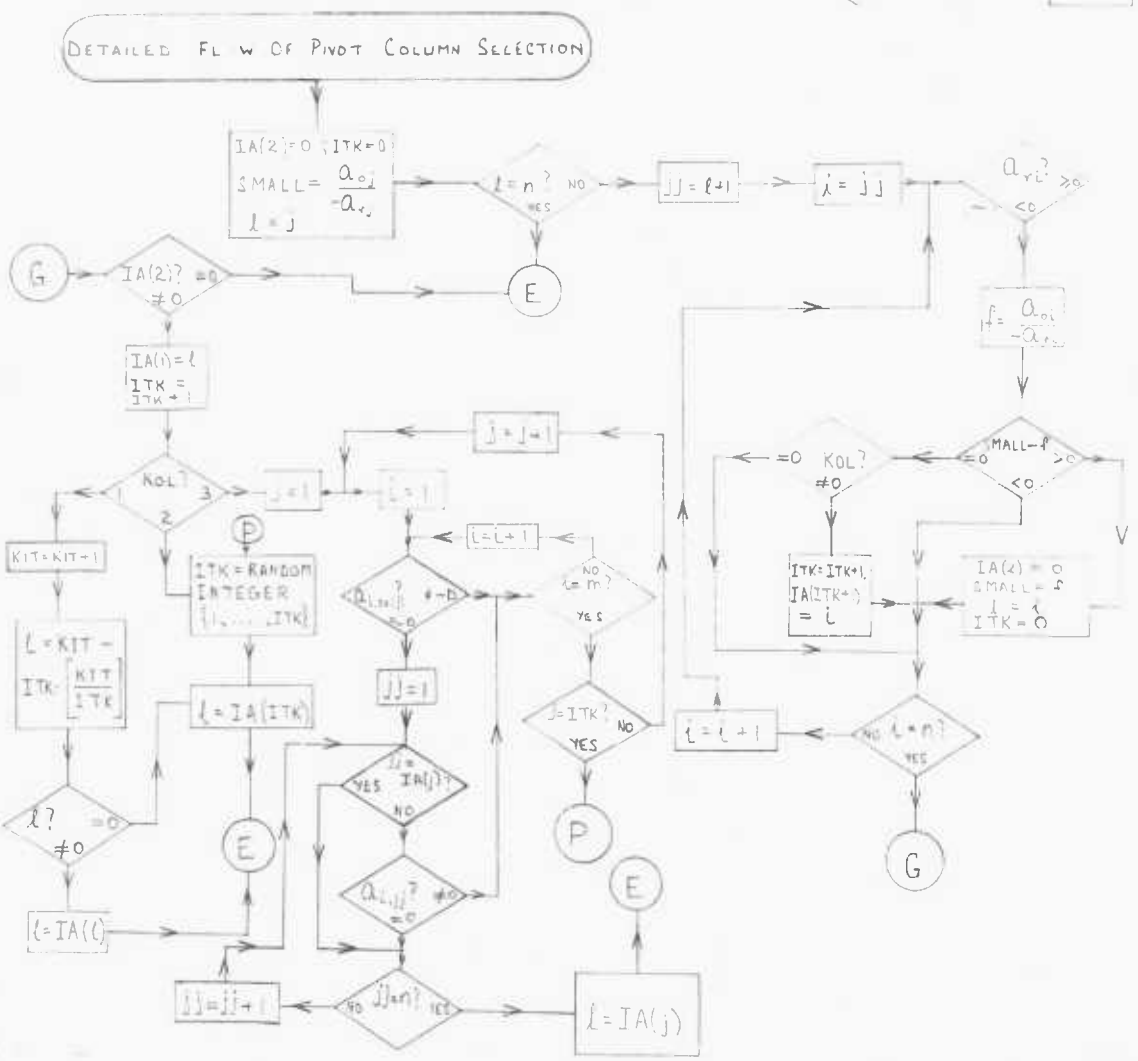
P_j REPRESENTS THE COLUMN

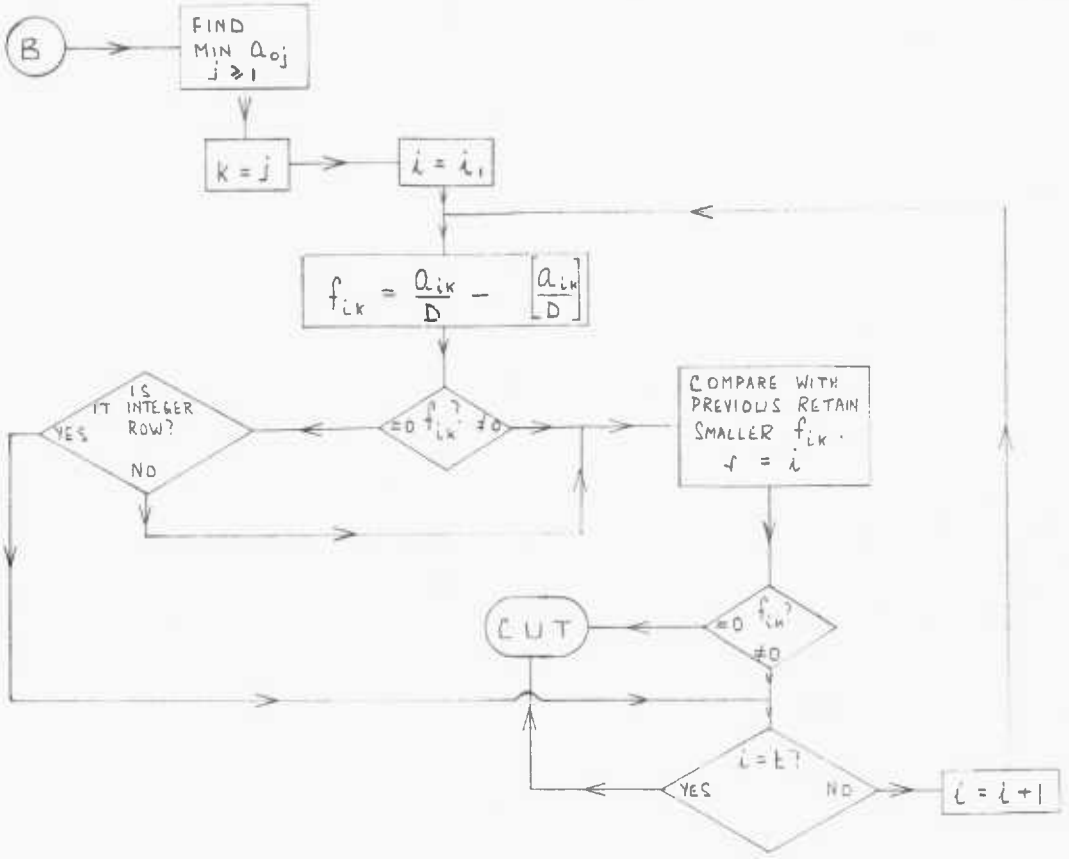
a_{ij}, $0 \leq i \leq m$

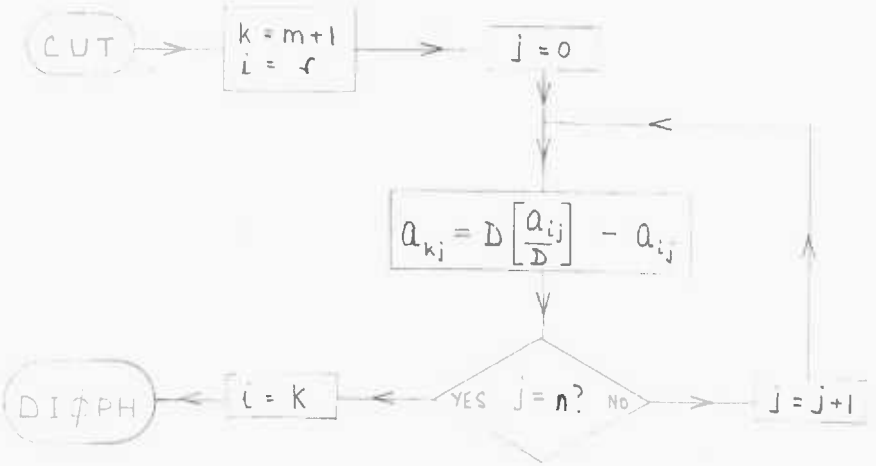
INTEGER VARIABLES IN ROWS i, r, t.

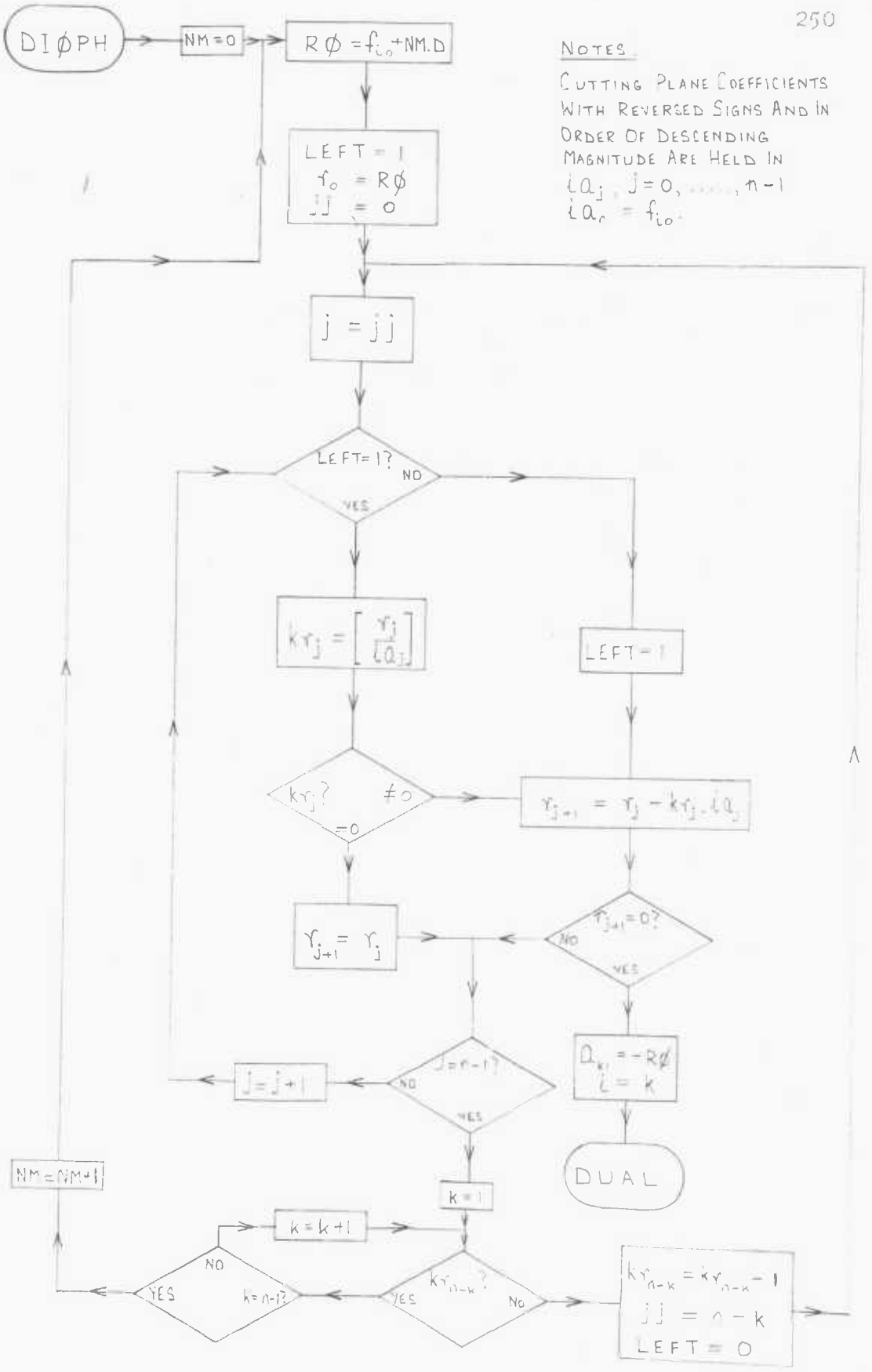
FLOW CHART 3. THE OPTIMAL CUT METHOD.


DUAL
r = i
FIND
MIN
$a_{+j}<0$
$j \geq 1$
$\frac{a_{0j}}{a_{rj}}$
E
YES TIE ? No
PIVOT COLUMN.
0
3
1
2
RANDOM.
SELECT FIRST.
CYCLIC
PROPER VAR. COLUMN.
l = j
j = 1
j≠l? NO
YES
j = j+1
$P_j = \frac{P_l a_r - P_l a_{rj}}{-D}$
NO YES j = n ?
A
$P_j = P_r$


DETAILED FLOW OF PIVOT COLUMN SELECTION
IA(2)=0 ITK=0
SMALL = $\frac{a_{0j}}{-a_{rj}}$
l = j
l = n ? NO
YES
JJ = l+1
j = JJ
a_{rJL} ? >0 <0
G
IA(2)? =0 ≠0
E
$f = \frac{a_{0i}}{-a_{ri}}$
IA(1)=l
ITK = ITK+1
j = j+1
KOL ? 3 2
j = 1
l = 1
P
KIT=KIT+1
ITK=RANDOM INTEGER 1,... ITK
l = KIT - ITK · [KIT/ITK]
$a_{L,JL}$? ≠0 =0
l = l+1
NO l = n ?
YES
J=ITK ? NO
YES
l = IA(ITK)
JJ=1
l ? =0 ≠0
E
JJ=JJ+1
JJ=n ? NO YES
l = IA(j)
$a_{L,JL}$? ≠0 =0
l = IA(l)
P
E
G
SMALL-f ? =0 <0
KOL ? =0 ≠0
ITK=ITK+1
IA(ITK+1) = i
IA(i) = 0
SMALL = f
l = 1
ITK = 0
V

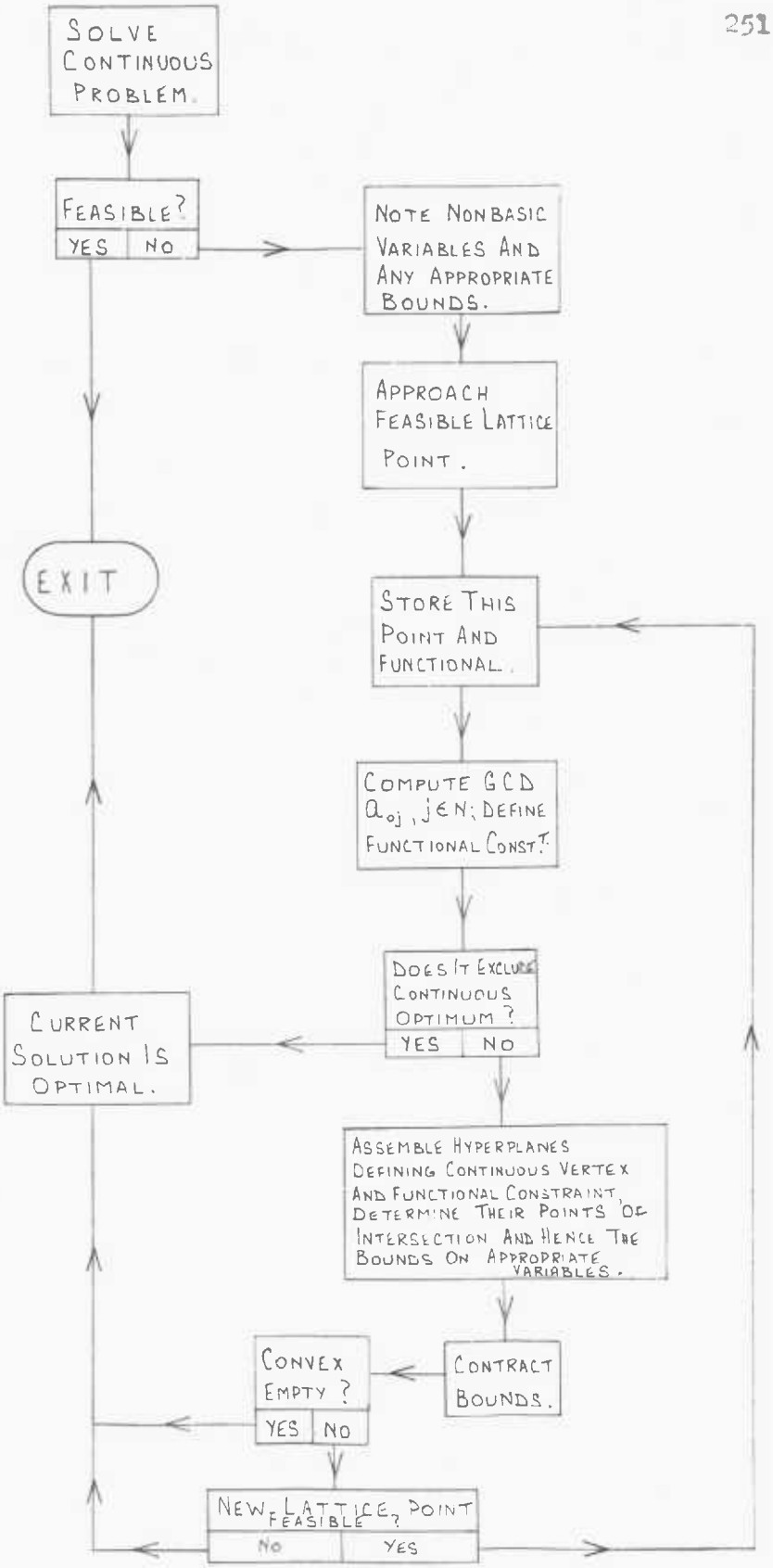

B
FIND MIN a_{oj} $j \geq 1$
$k = j$
$i = i_1$
$f_{ik} = \frac{a_{ik}}{D} - \left[\frac{a_{ik}}{D}\right]$
IS IT INTEGER ROW?
YES
NO
$=0$ f_{ik}? $\neq 0$
COMPARE WITH PREVIOUS RETAIN SMALLER f_{ik}. $r = i$
CUT
$=0$ f_{ik}? $\neq 0$
$i = t$?
YES
NO
$i = i+1$


CUT
$k = m+1$
$i = r$
$j = 0$
$a_{kj} = D\left[\frac{a_{ij}}{D}\right] - a_{ij}$
$j = n$?
YES
NO
$j = j+1$
$i = K$
DIOPH

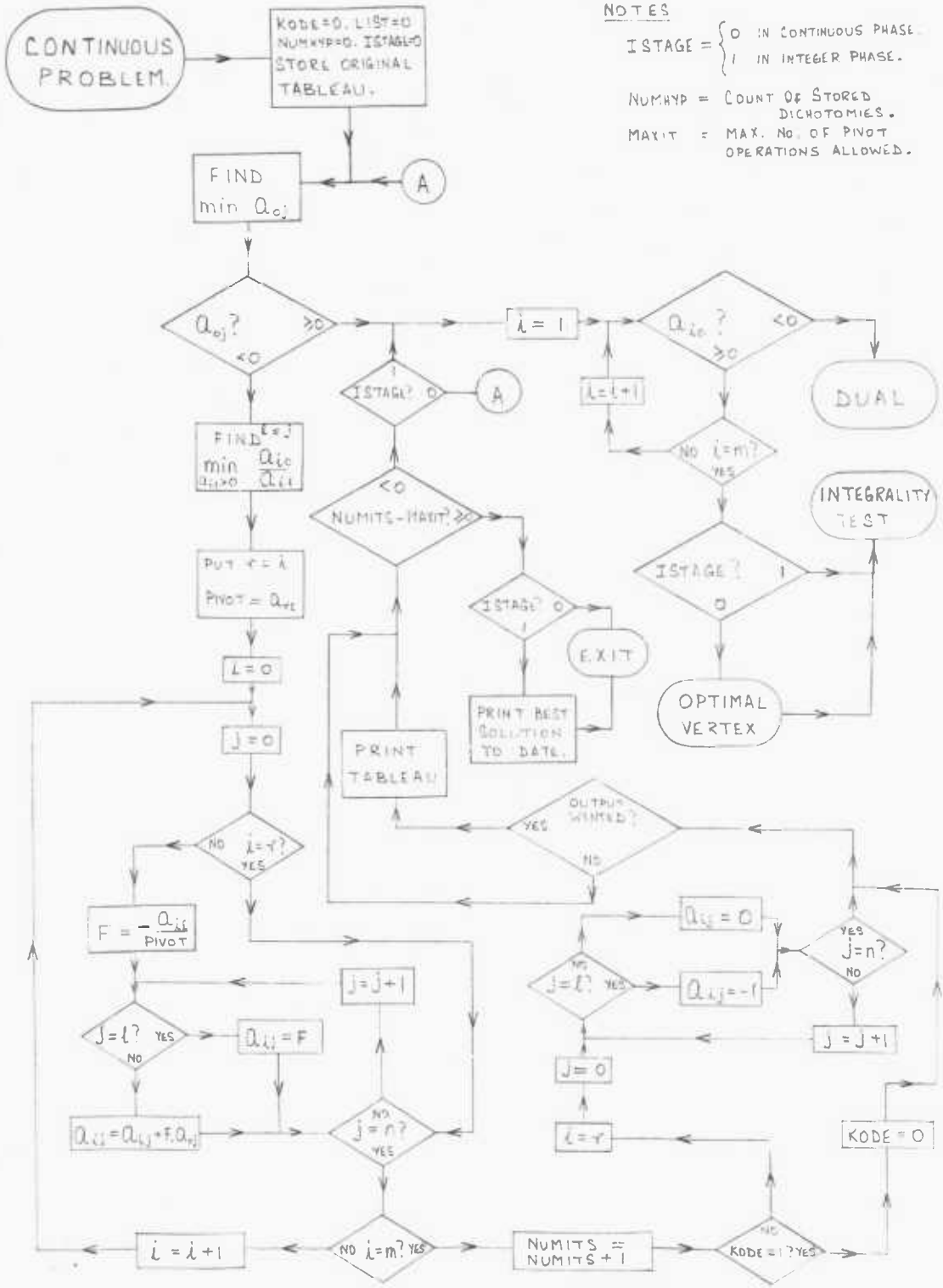


NOTES

CUTTING PLANE COEFFICIENTS WITH REVERSED SIGNS AND IN ORDER OF DESCENDING MAGNITUDE ARE HELD IN ia_j, $j=0,\ldots,n-1$
$ia_r = f_{i0}$.



FLOW CHART 4. CONVEX ISOLATION AND BOUND CONTRACTION METHOD.

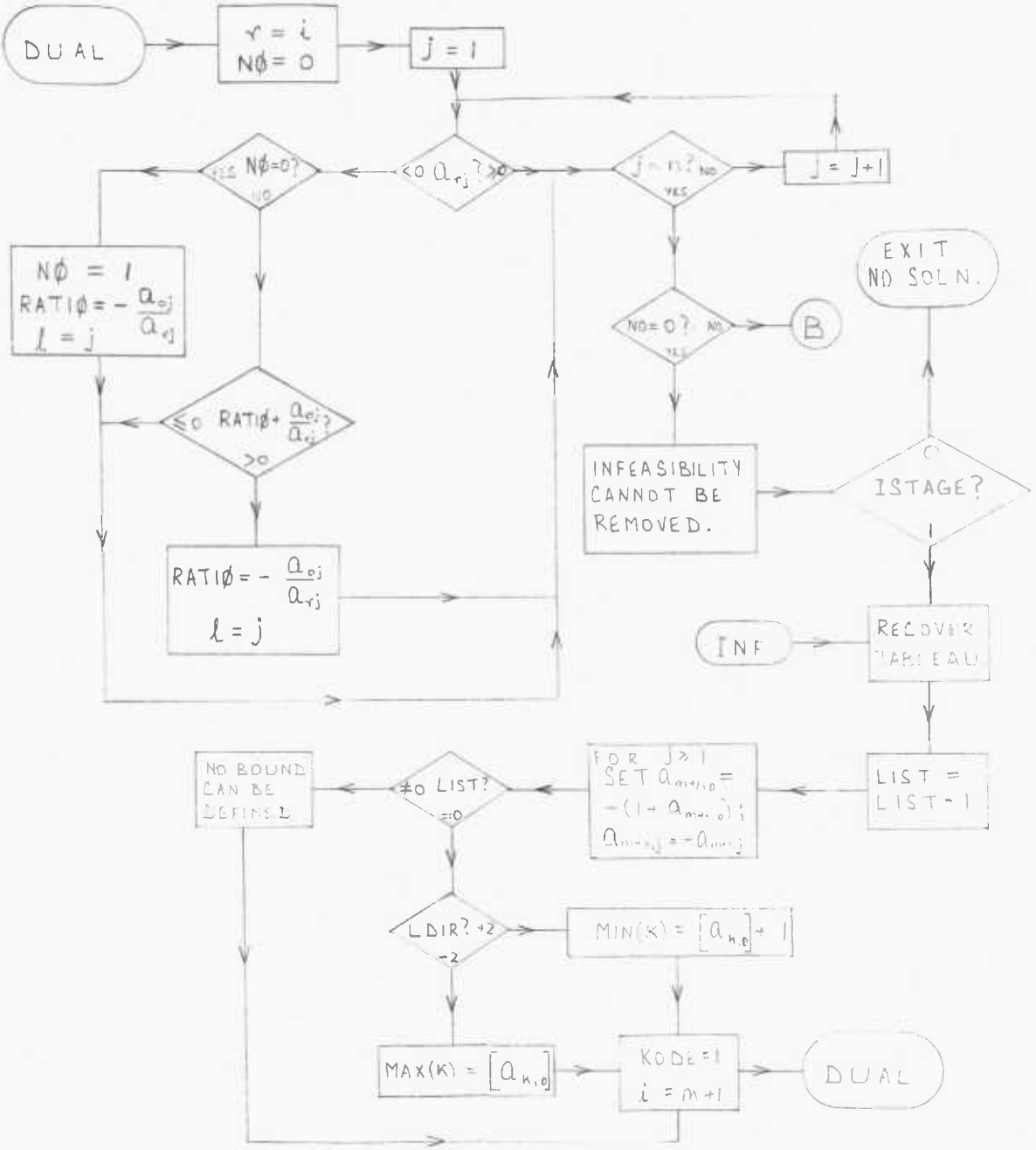


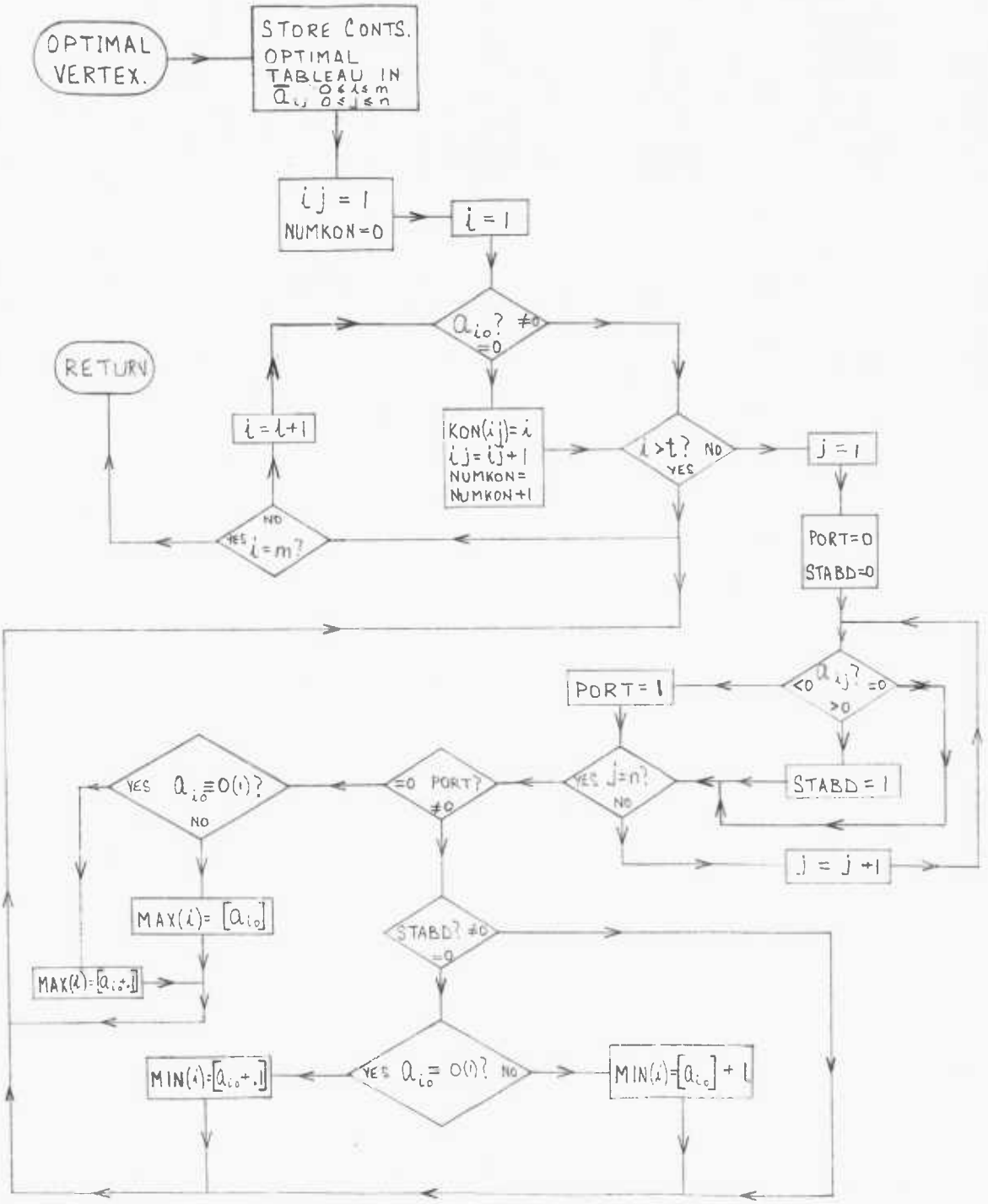
NOTES

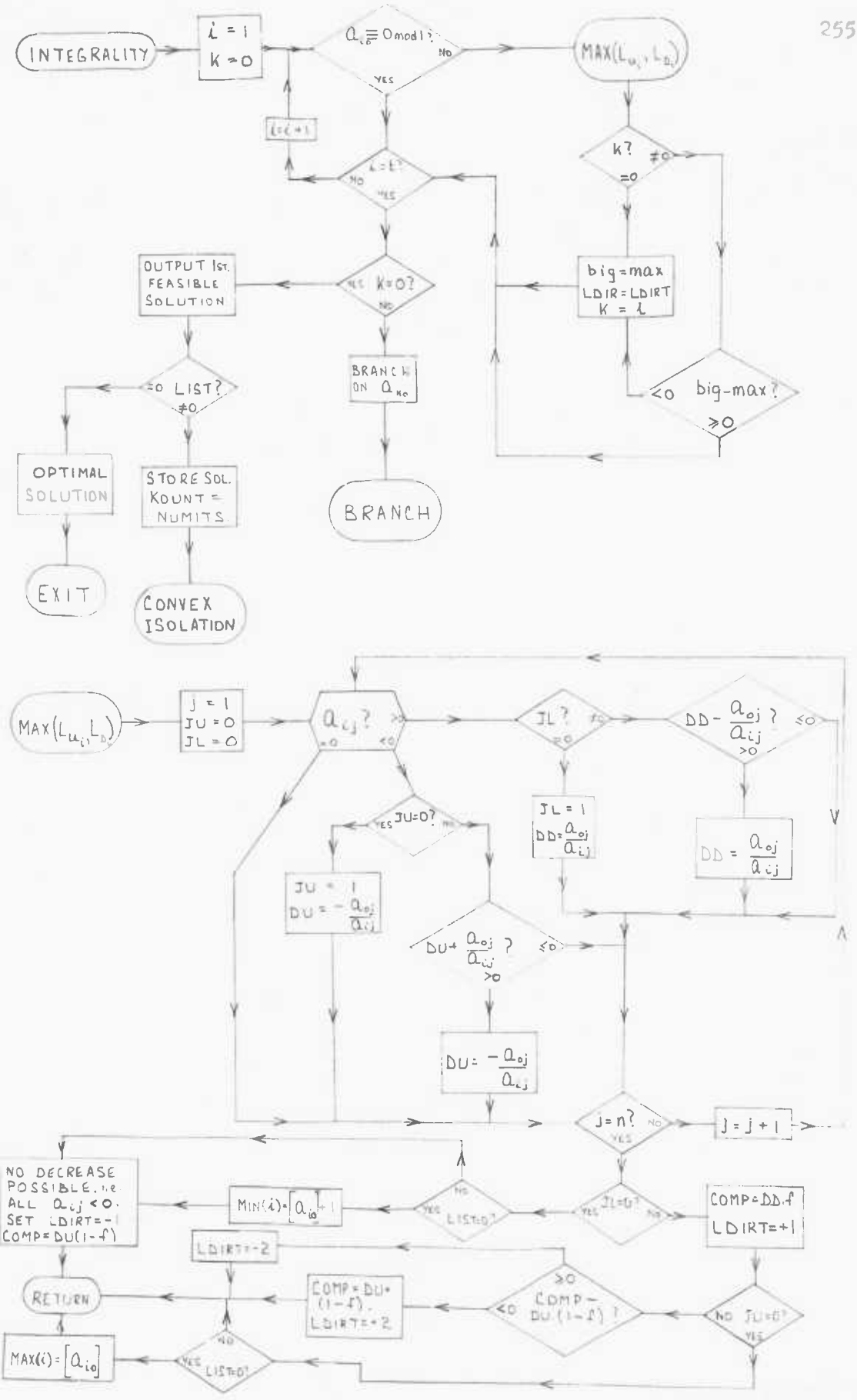
ISTAGE = { 0 IN CONTINUOUS PHASE. 1 IN INTEGER PHASE.

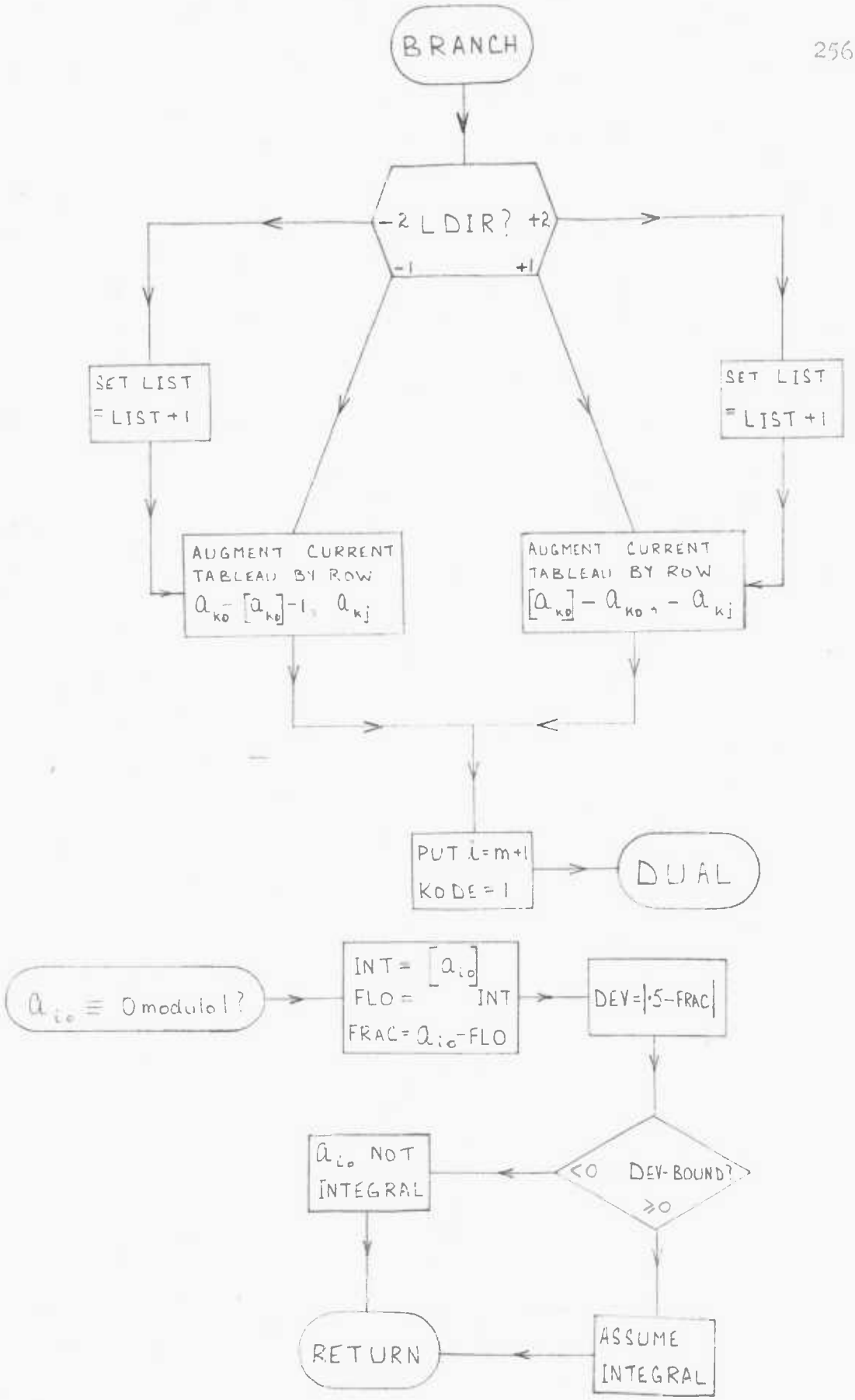
NUMHYP = COUNT OF STORED DICHOTOMIES.

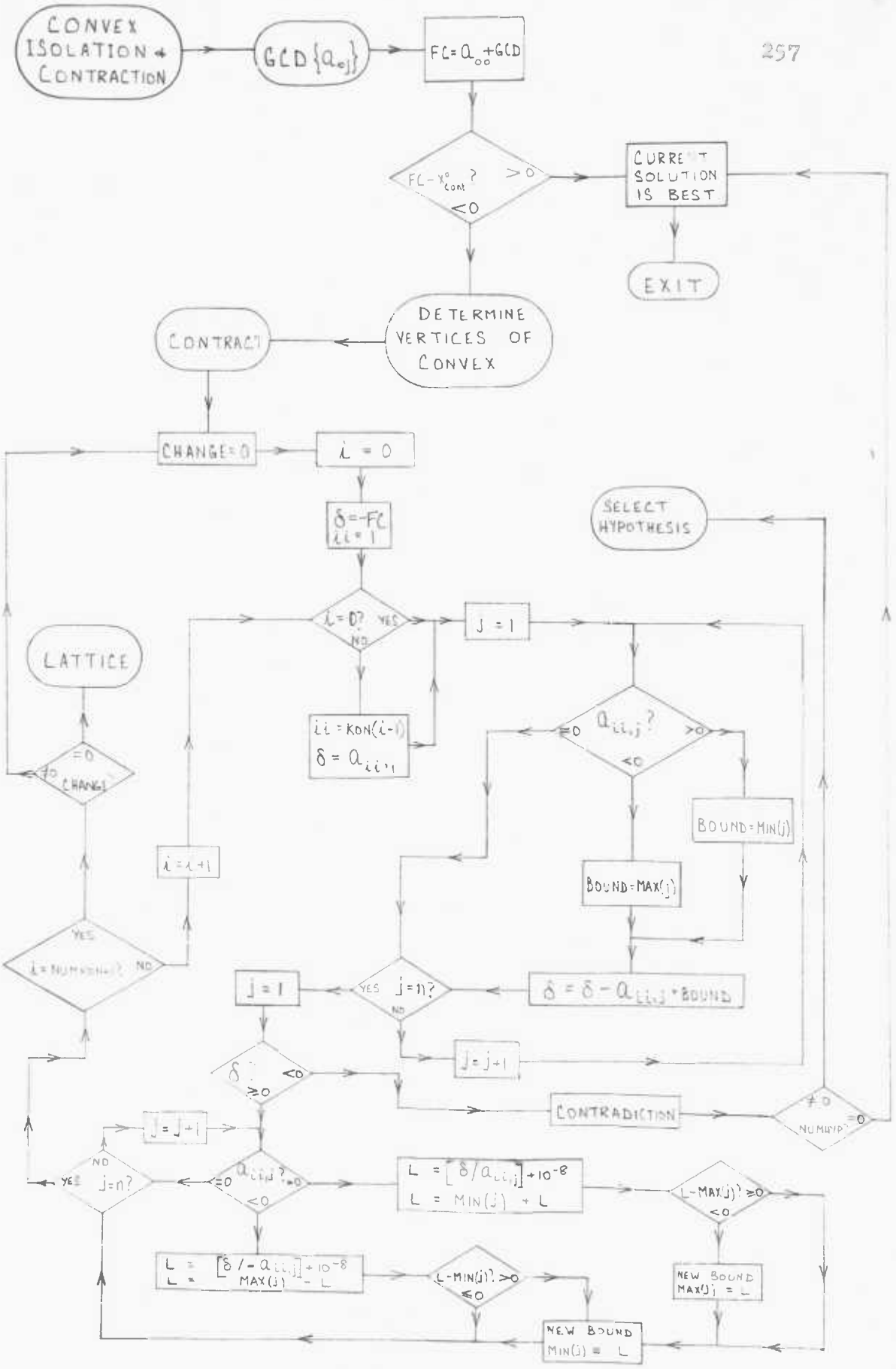
MAXIT = MAX. NO. OF PIVOT OPERATIONS ALLOWED.

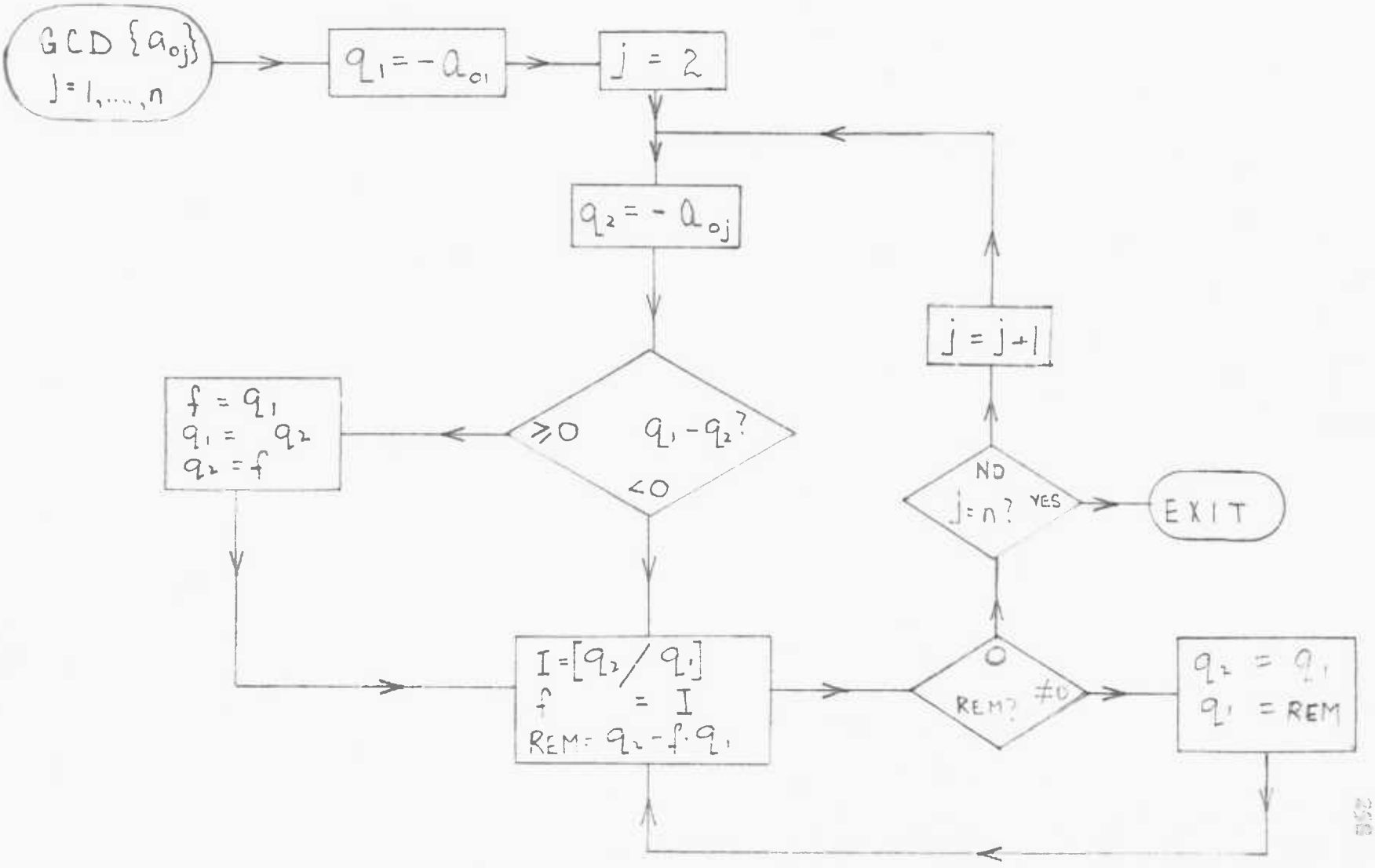

DUAL
r = i
NØ = 0
j = 1
<0 a_{rj}? >0
j = n? NO YES
J = J+1
YES NØ=0? NO
NØ = 1
RATIØ = $-\frac{a_{oj}}{a_{rj}}$
l = j
≤0 RATIØ + $\frac{a_{oj}}{a_{rj}}$? >0
RATIØ = $-\frac{a_{oj}}{a_{rj}}$
l = j
NØ=0? NO YES
B
INFEASIBILITY CANNOT BE REMOVED.
EXIT NO SOLN.
C ISTAGE?
INF
RECOVER TABLEAU
LIST = LIST - 1
FOR j ≥ 1 SET $a_{m+1,0}$ = $-(1+a_{max,0})$; $a_{m+1,j} = -a_{max,j}$
≠0 LIST? =0
NO BOUND CAN BE DEFINED
LDIR? +2 -2
MIN(K) = $[a_{k,0}]$ + 1
MAX(K) = $[a_{k,0}]$
KODE=1
i = m+1
DUAL

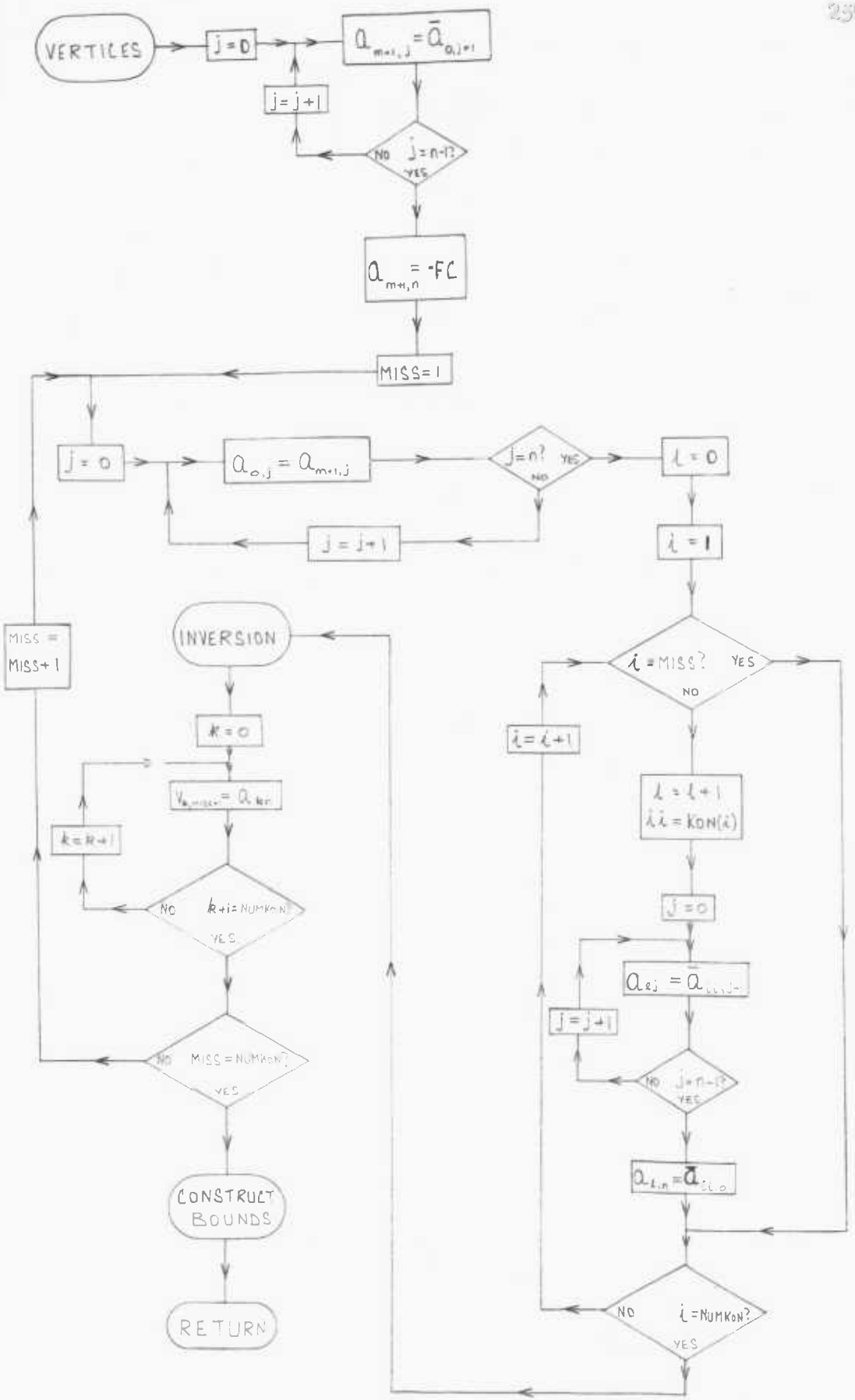

OPTIMAL VERTEX.
STORE CONTS. OPTIMAL TABLEAU IN $\bar{a}_{ij}$ $0 \le i \le m$ $0 \le j \le n$
$ij = 1$ NUMKON=0
$i = 1$
a_{i0}? $\neq 0$ $=0$
RETURN
$i = i+1$
KON(ij)=i $ij = ij+1$ NUMKON=NUMKON+1
$i > t$? NO YES
$j = 1$
NO YES $i = m$?
PORT=0 STABD=0
a_{ij}? <0 $=0$ >0
PORT=1
STABD=1
YES $j=n$? NO
$j = j+1$
$=0$ PORT? $\neq 0$
YES $a_{i0} \equiv 0(1)$? NO
MAX(i)=$[a_{i0}]$
MAX(i)=$[a_{i0}+.1]$
STABD? $\neq 0$ $=0$
YES $a_{i0} = 0(1)$? NO
MIN(i)=$[a_{i0}+.1]$
MIN(i)=$[a_{i0}]+1$

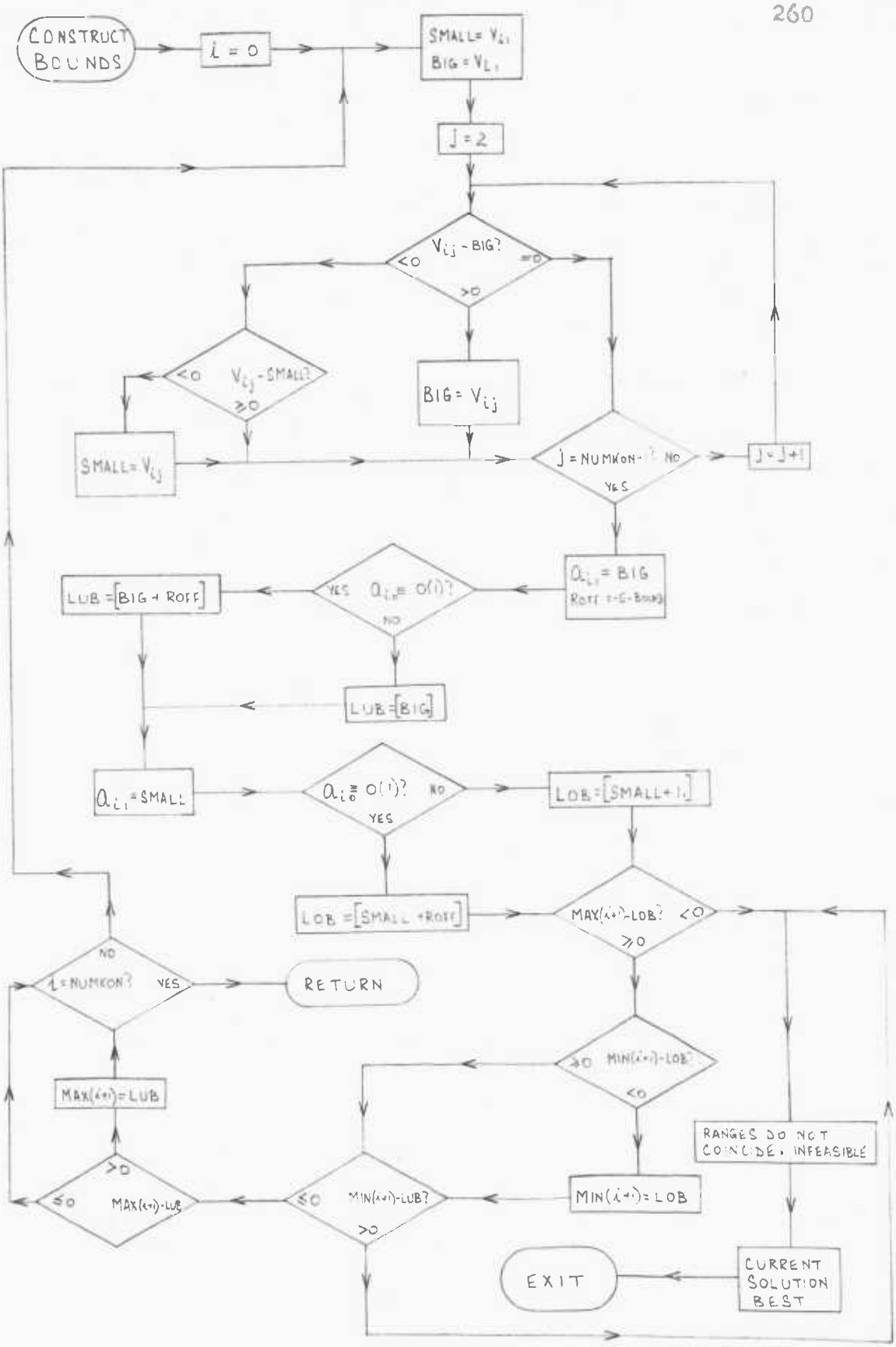

INTEGRALITY
i = 1
K = 0
$a_{i0} \equiv 0 \bmod 1$?
NO
YES
MAX(L_{u_i}, L_{D_i})
i = i + 1
i = L?
NO
YES
k? ≠0 =0
OUTPUT 1st. FEASIBLE SOLUTION
K = 0?
YES
NO
big = max
LDIR = LDIRT
K = i
<0 big-max? ≥0
=0 LIST? ≠0
BRANCH ON a_{ko}
OPTIMAL SOLUTION
STORE SOL. KOUNT = NUMITS
BRANCH
EXIT
CONVEX ISOLATION
MAX(L_{u_i}, L_{D_i})
j = 1
JU = 0
JL = 0
a_{ij}? >0 =0 <0
JL? ≠0 =0
DD - $\frac{a_{oj}}{a_{ij}}$? <0 >0
JL = 1
DD = $\frac{a_{oj}}{a_{ij}}$
DD = $\frac{a_{oj}}{a_{ij}}$
YES JU=0? NO
JU = 1
DU = $-\frac{a_{oj}}{a_{ij}}$
DU + $\frac{a_{oj}}{a_{ij}}$? <0 >0
DU = $-\frac{a_{oj}}{a_{ij}}$
j = n? NO YES
j = j + 1
NO DECREASE POSSIBLE. i.e. ALL a_{ij} < 0. SET LDIRT = -1 COMP = DU(1-f)
MIN(i) = $[a_{io}]$ + 1
NO YES LIST=0?
YES JL=0? NO
COMP = DD·f LDIRT = +1
LDIRT = -2
RETURN
COMP = DU· (1-f), LDIRT = 2
<0 COMP - DU·(1-f) ? >0
NO JL=0? YES
MAX(i) = $[a_{io}]$
NO YES LIST=0?

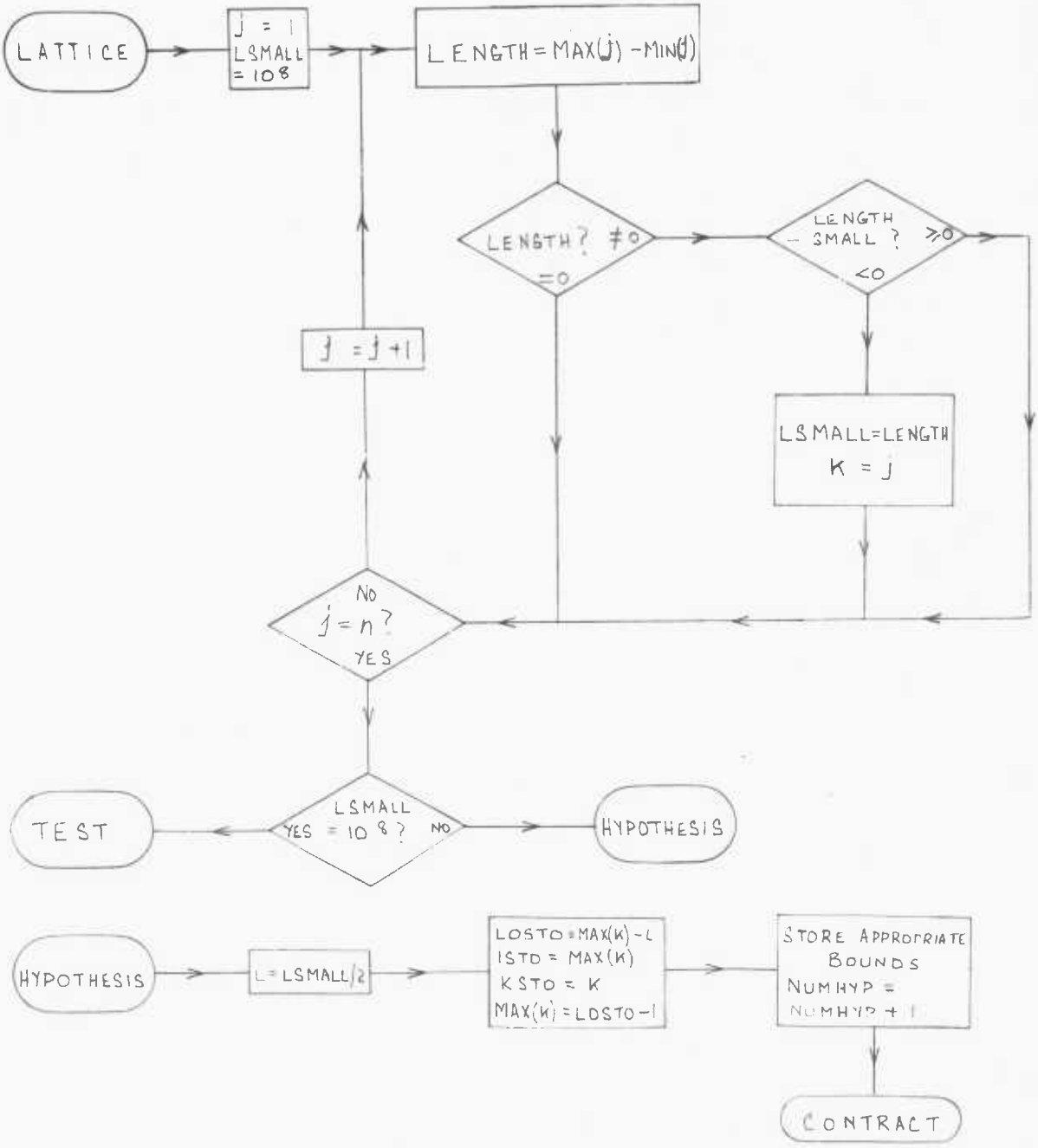

BRANCH
-2 LDIR? +2
-1
+1
SET LIST = LIST + 1
SET LIST = LIST + 1
AUGMENT CURRENT TABLEAU BY ROW
$a_{ko} - [a_{kj}] - 1 \cdot a_{kj}$
AUGMENT CURRENT TABLEAU BY ROW
$[a_{ko}] - a_{ko+1} - a_{kj}$
PUT i = m+1
KODE = 1
DUAL
$a_{io} \equiv$ 0 modulo 1?
INT = $[a_{io}]$
FLO = INT
FRAC = a_{io} - FLO
DEV = |.5 - FRAC|
a_{io} NOT INTEGRAL
<0 DEV - BOUND? ≥0
RETURN
ASSUME INTEGRAL

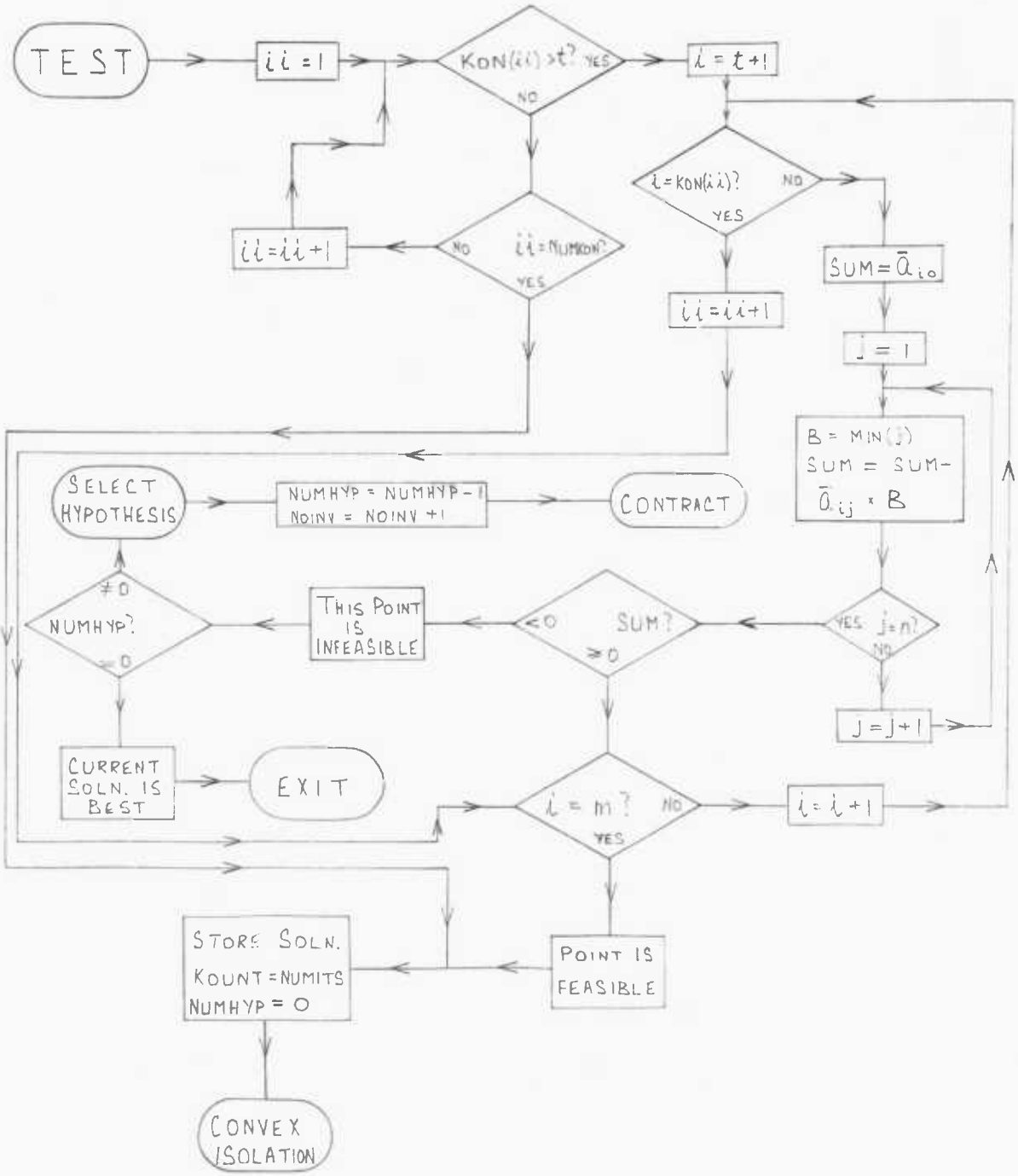

CONVEX ISOLATION + CONTRACTION
GCD $\{a_{oj}\}$
$FC = a_{oo} + GCD$
$FC - X^{o}_{cont}$?
>0
<0
CURRENT SOLUTION IS BEST
EXIT
DETERMINE VERTICES OF CONVEX
CONTRACT
CHANGE=0
$i = 0$
$\delta = -FC$
$ii = 1$
SELECT HYPOTHESIS
$i = 0$? YES NO
$j = 1$
LATTICE
$ii = KON(i-1)$
$\delta = a_{ii,1}$
$a_{ii,j}$?
=0
>0
<0
BOUND = MIN(j)
BOUND = MAX(j)
$i = i+1$
=0 CHANGE
$i = NUMBER(i)$? YES NO
$j = 1$
$j = 1$? YES NO
$\delta = \delta - a_{ii,j} * BOUND$
$j = j+1$
δ ?
<0
≥0
CONTRADICTION
≠0 NUMBER = 0
$j = j+1$
$j = n$? YES NO
$a_{ii,j}$?
=0
<0
$L = [\delta / a_{ii,j}] + 10^{-8}$
$L = MIN(j) + L$
$L - MAX(j)$? ≥0 <0
$L = [\delta / a_{ii,j}] + 10^{-8}$
$L = MAX(j) - L$
$L - MIN(j)$? ≥0 <0
NEW BOUND MAX(j) = L
NEW BOUND MIN(j) = L


GCD $\{a_{oj}\}$
$j = 1, \ldots, n$
$q_1 = -a_{o1}$
$j = 2$
$q_2 = -a_{oj}$
$q_1 - q_2$?
≥ 0
< 0
$f = q_1$
$q_1 = q_2$
$q_2 = f$
$I = [q_2 / q_1]$
$f = I$
$REM = q_2 - f \cdot q_1$
REM? 0
$\neq 0$
$q_2 = q_1$
$q_1 = REM$
$j = j + 1$
$j = n$? NO
YES
EXIT


VERTICES
j = 0
$a_{m+1,j} = \bar{a}_{0,j+1}$
j = j + 1
NO j = n-1? YES
$a_{m+1,n} = -FC$
MISS = 1
j = 0
$a_{0,j} = a_{m+1,j}$
j = n? YES NO
j = j + 1
i = 0
i = 1
i = MISS? YES NO
i = i + 1
l = l + 1
ii = KON(i)
j = 0
$a_{ij} = \bar{a}_{ii,j-1}$
j = j + 1
NO j = n-1? YES
$a_{i,n} = \bar{a}_{ii,0}$
NO i = NUMKON? YES
INVERSION
k = 0
$V_{k-MISS} = a_{kr}$
k = k + 1
NO k + i = NUMKON? YES
NO MISS = NUMKON? YES
MISS = MISS + 1
CONSTRUCT BOUNDS
RETURN


CONSTRUCT BOUNDS
i = 0
SMALL = V_i1
BIG = V_Li
J = 2
V_ij - BIG?
<0
=0
>0
V_ij - SMALL?
<0
≥0
BIG = V_ij
SMALL = V_ij
J = NUMKON-?
NO
YES
J = J+1
a_iJ = BIG
ROFF = C-BIG
a_iJ = 0(1)?
YES
NO
LUB = [BIG + ROFF]
LUB = [BIG]
a_i1 = SMALL
a_i1 ≥ 0(1)?
NO
YES
LOB = [SMALL + 1]
LOB = [SMALL + ROFF]
MAX(i+1) - LOB?
<0
≥0
I = NUMKON?
NO
YES
RETURN
≥0 MIN(i+1) - LOB
<0
MAX(i+1) = LUB
RANGES DO NOT COINCIDE, INFEASIBLE
>0
≤0 MAX(i+1) - LUB
≤0 MIN(i+1) - LUB?
>0
MIN(i+1) = LOB
EXIT
CURRENT SOLUTION BEST


LATTICE
J = 1
LSMALL = 10^8
LENGTH = MAX(J) - MIN(J)
LENGTH? ≠0 =0
LENGTH - SMALL ? ≥0 <0
J = J + 1
LSMALL = LENGTH
K = J
NO
J = n ?
YES
LSMALL = 10^8 ?
YES
NO
TEST
HYPOTHESIS
HYPOTHESIS
L = LSMALL/2
LOSTO = MAX(K) - L
ISTO = MAX(K)
KSTO = K
MAX(K) = LOSTO - 1
STORE APPROPRIATE BOUNDS
NUMHYP = NUMHYP + 1
CONTRACT


TEST
ii = 1
KON(ii) > t?
YES
NO
i = t+1
i = KON(ii)?
NO
YES
ii = ii+1
NO
ii = NUMKON?
YES
SUM = ā i0
ii = ii+1
j = 1
B = MIN( )
SUM = SUM - ā ij × B
SELECT HYPOTHESIS
NUMHYP = NUMHYP - 1
NOINV = NOINV + 1
CONTRACT
≠ 0
NUMHYP?
= 0
THIS POINT IS INFEASIBLE
< 0
SUM?
≥ 0
YES
j = n?
NO
J = J+1
CURRENT SOLN. IS BEST
EXIT
i = m?
NO
YES
i = i+1
STORE SOLN.
KOUNT = NUMITS
NUMHYP = 0
POINT IS FEASIBLE
CONVEX ISOLATION

APPENDIX III

In this final appendix we present the FORTRAN listings for the

(i) All Integer Integer Code (With Modified Cut),

(ii) Pure Integer Code (Standard Cut),

(iii) Pure Integer Code (Optimal Cut),

(iv) Convex Isolation and Bound Contraction Code,

(v) Branch and Bound Code,

(vi) Mixed Integer Code,

(vii) Accelerated Euclidean Code.

FORTRAN PROGRAM LISTING.

ALL INTEGER INTEGER CODE
(WITH MODIFIED CUT).

```
    DIMENSION MAT(63,38),VAR(38),IROW(60)
    READ 90,NOJOBS
 90 FORMAT(I3)
    NUM=0
 93 READ 100,M,N,KODE,MAX,METHOD,IS,MODIF,((MAT(I,J),J=1,N),I=1,M-N+1)
100 FORMAT(2I6,8X,I1,I6,1X,I1,2X,I2,I7/(10I8))
    NN=N-2
    MM=M-N+2
    DO 31 I=MM,M
    DO 31 J=1,N
 31 MAT(I,J)=0
    MM=M-N
    DO 32 J=2,N
 32 MAT(MM+J,J)=-1
    READ 107,(VAR(J),J=1,M)
107 FORMAT(20A4)
    PRINT 106,IS,METHOD
106 FORMAT(1H1//47H ALL INTEGER LINEAR PROGRAMME RESULT TO PROBLEM,I3,
   147H SELECTING THE GENERATOR DEFINED BY PARAMETER (,I1,2H).)
    IF(MODIF)251,250,251
250 PRINT 252
252 FORMAT(/14H STANDARD CUT.)
    GO TO 254
251 PRINT 253
253 FORMAT(/14H MODIFIED CUT.)
254 KOUNT=0
  1 NONSTO=N
    NOR=0
    MLXSTO=-1
    IROW(1)=0
```

```
      DO 10 I=2,M
      NEG=0
      IF(MAT(I,1))21,10,10
   21 IF(MAX-KOUNT)35,35,77
   77 IF(METHOD-1)11,78,79
   78 NOR=NOR+1
      IROW(NOR)=I
      GO TO 10
   79 NON=0
      SMALL=10.**20
      DO 1200 J=2,N
      IF(MAT(I,J))1201,1200,1200
 1201 NEG=1
      IF(METHOD-2)1202,1202,1203
 1202 F1=MAT(1,J)
      F2=-MAT(I,J)
      F1=F1/F2
      IF(F1-SMALL)1204,1200,1200
 1204 SMALL=F1
      MLX=MAT(1,J)
      GO TO 1200
 1203 NON=NON+1
 1200 CONTINUE
      IF(NEG)1205,1206,1205
 1205 IF(METHOD-2)1207,1207,1209
 1207 IF(MLX-MLXSTO)10,10,1208
 1208 MLXSTO=MLX
      IROW(1)=I
      GO TO 10
 1209 IF(NON-NONSTO)1210,10,10
 1210 NONSTO=NON
      IROW(1)=I
   10 CONTINUE
      IF(IROW(1))1211,25,1211
 1211 IF(METHOD-2)1212,1213,1213
 1212 JJ=KOUNT+1-(KOUNT/NOR)*NOR
```

```
      I=IROW(JJ)
      GO TO 11
 1213 I=IROW(1)
   11 IR=I
      DO 20 K=2,N
      IF(MAT(IR,K))12,20,20
   20 CONTINUE
 1206 PRINT 101
  101 FORMAT(//63H EXIT CALLED BECAUSE COEFFICIENTS OF NEGATIVE ROW ARE
     1POSITIVE./26H TABLEAU AT THIS STAGE WAS//)
      ASSIGN 2 TO KK
      GO TO 400
   12 LL=K
      JJ=K+1
      IF(JJ-N)46,46,45
   46 DO 30 J=JJ,N
      IF(MAT(IR,J))13,30,30
   13 DO 40 L=1,M
      IF(MAT(L,J)-MAT(L,LL))14,40,30
   40 CONTINUE
      GO TO 30
   14 LL=J
   30 CONTINUE
   45 AMBDA=1.0
      DO 50 J=2,N
      IF(MAT(IR,J))15,50,50
   15 IF(MAT(1,LL))16,17,16
   16 IT=MAT(1,J)/MAT(1,LL)
      GO TO 76
   17 IT=1
   76 FLT=IT
      FLM=-MAT(IR,J)
      AMBD=FLM/FLT
      IF(AMBDA-AMBD)18,50,50
   18 AMBDA=AMBD
   50 CONTINUE
```

```
      IF(MODIF)1227,1224,1227
1227  FLM=MAT(IR,1)
      KONST=INT(FLM/AMBDA)
      IF(KONST+1)1216,1214,1216
1214  IPOS=0
      MBIG=-MAT(IR,1)
      SMALL=10.**20
      GO TO 1240
1216  IPOS=1
1215  FLT=KONST
      SMALL=FLM/(FLT+1.)-.0000001
1240  DO 1217 J=2,N
      IF(MAT(IR,J))1218,1218,1221
1218  IF(IPOS)1217,1219,1217
1219  IF(MBIG+MAT(IR,J))1220,1217,1217
1220  MBIG=-MAT(IR,J)
      GO TO 1217
1221  IPOS=2
      FLM=MAT(IR,J)
      KONST=INT(FLM/AMBDA)
      IF(KONST)1230,1217,1230
1230  FLT=KONST
      FLM=FLM/FLT
      IF(SMALL-FLM)1217,1217,1222
1222  SMALL=FLM
1217  CONTINUE
      IF(IPOS-1)1223,1225,1225
1223  AMBDA=MBIG
      GO TO 1224
1225  IF(AMBDA-SMALL)1226,1224,1224
1226  AMBDA=SMALL
1224  DO 60 J=1,N
      IF(J-LL)19,60,19
   19 TOP=MAT(IR,J)
      KONST=INT(TOP/AMBDA)
      IF(KONST)29,60,29
```

```
 29 DO 70 I=1,M
 70 MAT(I,J)=MAT(I,J)+KONST*MAT(I,LL)
 60 CONTINUE
    KOUNT=KOUNT+1
    IF(KODE)5,1,5
  5 PRINT 142
    ASSIGN 1 TO KK
142 FORMAT(/19H CURRENT TABLEAU IS)
400 GO TO (201,202,203,204,205,206,207,208,209,210,211,212,213,214,215
   1,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,2
   132,233,234,235,236),NN
201 PRINT 301,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
301 FORMAT(20X,A4,3H = ,3I8)
    GO TO KK,(1,2)
202 PRINT 302,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
302 FORMAT(20X,A4,3H = ,4I8)
    GO TO KK,(1,2)
203 PRINT 303,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
303 FORMAT(20X,A4,3H = ,5I8)
    GO TO KK,(1,2)
204 PRINT 304,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
304 FORMAT(20X,A4,3H = ,6I8)
    GO TO KK,(1,2)
205 PRINT 305,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
305 FORMAT(20X,A4,3H = ,7I8)
    GO TO KK,(1,2)
206 PRINT 306,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
306 FORMAT(20X,A4,3H = ,8I8)
    GO TO KK,(1,2)
207 PRINT 307,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
307 FORMAT(20X,A4,3H = ,9I8)
    GO TO KK,(1,2)
208 PRINT 308,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
308 FORMAT(20X,A4,3H = ,10I8)
    GO TO KK,(1,2)
209 PRINT 309,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
```

```
309 FORMAT(A4,3H = ,11I10)
    GO TO KK,(1,2)
210 PRINT 310,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
310 FORMAT(A4,3H = ,11I10/107X,I10//)
    GO TO KK,(1,2)
211 PRINT 311,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
311 FORMAT(A4,3H = ,11I10/97X,2I10//)
    GO TO KK,(1,2)
212 PRINT 312,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
312 FORMAT(A4,3H = ,11I10/87X,3I10//)
    GO TO KK,(1,2)
213 PRINT 313,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
313 FORMAT(A4,3H = ,11I10/77X,4I10//)
    GO TO KK,(1,2)
214 PRINT 314,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
314 FORMAT(A4,3H = ,11I10/67X,5I10//)
    GO TO KK,(1,2)
215 PRINT 315,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
315 FORMAT(A4,3H = ,11I10/57X,6I10//)
    GO TO KK,(1,2)
216 PRINT 316,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
316 FORMAT(A4,3H = ,11I10/47X,7I10//)
    GO TO KK,(1,2)
217 PRINT 317,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
317 FORMAT(A4,3H = ,11I10/37X,8I10//)
    GO TO KK,(1,2)
218 PRINT 318,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
318 FORMAT(A4,3H = ,11I10/27X,9I10//)
    GO TO KK,(1,2)
219 PRINT 319,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
319 FORMAT(A4,3H = ,11I10/17X,10I10//)
    GO TO KK,(1,2)
220 PRINT 320,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
320 FORMAT(A4,3H = ,11I10/7X,11I10//)
    GO TO KK,(1,2)
221 PRINT 321,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
```

```
321 FORMAT(A4,3H = ,11I10/7X,11I10/107X,I10//)
    GO TO KK,(1,2)
222 PRINT 322,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
322 FORMAT(A4,3H = ,11I10/7X,11I10/97X,2I10//)
    GO TO KK,(1,2)
223 PRINT 323,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
323 FORMAT(A4,3H = ,11I10/7X,11I10/87X,3I10//)
    GO TO KK,(1,2)
224 PRINT 324,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
324 FORMAT(A4,3H = ,11I10/7X,11I10/77X,4I10//)
    GO TO KK,(1,2)
225 PRINT 325,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
325 FORMAT(A4,3H = ,11I10/7X,11I10/67X,5I10//)
    GO TO KK,(1,2)
226 PRINT 326,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
326 FORMAT(A4,3H = ,11I10/7X,11I10/57X,6I10//)
    GO TO KK,(1,2)
227 PRINT 327,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
327 FORMAT(A4,3H = ,11I10/7X,11I10/47X,7I10//)
    GO TO KK,(1,2)
228 PRINT 328,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
328 FORMAT(A4,3H = ,11I10/7X,11I10/37X,8I10//)
    GO TO KK,(1,2)
229 PRINT 329,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
329 FORMAT(A4,3H = ,11I10/7X,11I10/27X,9I10//)
    GO TO KK,(1,2)
230 PRINT 330,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
330 FORMAT(A4,3H = ,11I10/7X,11I10/17X,10I10//)
    GO TO KK,(1,2)
231 PRINT 331,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
331 FORMAT(A4,3H = ,11I10/7X,11I10/7X,11I10//)
    GO TO KK,(1,2)
232 PRINT 332,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
332 FORMAT(A4,3H = ,11I10/7X,11I10/7X,11I10/107X,I10//)
    GO TO KK,(1,2)
233 PRINT 333,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
```

```
333 FORMAT(A4,3H = ,11I10/7X,11I10/7X,11I10/97X,2I10//)
    GO TO KK,(1,2)
234 PRINT 334,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
334 FORMAT(A4,3H = ,11I10/7X,11I10/7X,11I10/87X,3I10//)
    GO TO KK,(1,2)
235 PRINT 335,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
335 FORMAT(A4,3H = ,11I10/7X,11I10/7X,11I10/77X,4I10//)
    GO TO KK,(1,2)
236 PRINT 336,(VAR(I),(MAT(I,J),J=1,N),I=1,M)
336 FORMAT(A4,3H = ,11I10/7X,11I10/7X,11I10/67X,5I10//)
    GO TO KK,(1,2)
 35 PRINT 3,MAX
  3 FORMAT(//96H ITERATIVE PROCESS HAS BEEN DISCONTINUED BECAUSE THE M
   1AXIMUM NUMBER OF ALLOWABLE ITERATIONS,VIZ.,I4/62H,HAVE BEEN PERFOR
   1MED AND A FEASIBLE SOLUTION NOT YET ATTAINED.)
    IF(KODE)75,8,75
 75 PRINT 110
110 FORMAT(/64H THE SITUATION AT THIS STAGE IS GIVEN BY THE LAST TABLE
   1AU ABOVE.)
    GO TO 2
  8 PRINT 112
112 FORMAT(/52H THE SITUATION AT THIS STAGE IS GIVEN BY THE TABLEAU//)
    ASSIGN 2 TO KK
    GO TO 400
 25 PRINT 102,(VAR(I),MAT(I,1),I=2,M)
102 FORMAT(///32H VALUES OF INTEGER VARIABLES ARE//7(A6,3H = ,I6,1H,))
    PRINT 103,VAR(1),MAT(1,1),KOUNT
103 FORMAT(//30H PRODUCING THE OPTIMAL VALUE  ,A4,3H = ,I6,6H AFTER,I4
   1,12H ITERATIONS.)
  2 NUM=NUM+1
    IF(NOJOBS-NUM)92,92,93
 92 CALL EXIT
    END
```

FORTRAN PROGRAM LISTING.

PURE INTEGER CODE
(STANDARD CUT).

```
      DIMENSION MAT(66,38),KEW(38)
      RA=34359738368.
      RB=68719476736.
      RC=137438953472.
      RE=1./RA
      READ 300,NOJOBS
  300 FORMAT(I3)
      NUM=0
  303 READ 1,M,N,LD,OUT,KODE,MAXIT,NOROS,METHOD,KOL,MZED,IS,(MAT(1,J),J=
     11,N),((MAT(I,J),J=1,N),I=N+1,M)
    1 FORMAT(3I6,F2.0,I1,I6,4I1,I2/(10I8))
      RD=13543288579.
      KIT=0
      KUTNO=0
      NN=N-2
      DO 301 I=2,N
      DO 301 J=1,N
  301 MAT(I,J)=0
      DO 302 I=2,N
  302 MAT(I,I)=-1
      IF(MZED)461,460,461
  460 I1=1
      GO TO 462
  461 I1=2
  462 IT=-1
      PRINT 80,IS,METHOD,KOL
   80 FORMAT(1H1//39H LP INTEGER ROUTINE SOLUTION TO PROBLEM,I3,47H USIN
     1G THE VARIANTS DEFINED BY PARAMETER CODE (,I1,1H,,I1,2H).)
      IF(MZED)581,580,581
  580 PRINT 582
```

```
582 FORMAT(64H THE OBJECTIVE FUNCTION IS ADMITTED AS A POSSIBLE CUT GE
   1NERATOR.)
    GO TO 583
581 PRINT 584
584 FORMAT(56H THE OBJECTIVE FUNCTION IS EXCLUDED FROM CUT GENERATION.
   1)
583 IF(NOROS)310,309,310
309 NINT=N
    GO TO 20
310 NINT=M
 20 MIN=MAT(1,2)
    L=2
    DO 2 J=3,N
    IF(MIN-MAT(1,J))2,2,3
  3 MIN=MAT(1,J)
    L=J
  2 CONTINUE
    IF(MIN)4,75,75
  4 R=10.**20
    IT=IT+1
    IF(OUT)901,902,901
901 ASSIGN 982 TO KK
    PRINT 904
904 FORMAT(//19H CURRENT TABLEAU IS)
    GO TO 903
982 PRINT 983,LD
983 FORMAT(16H WITH DIVISOR = ,I8,1H.)
902 DO 8 I=2,M
    IF(MAT(I,L))8,8,9
  9 TOP=MAT(I,1)
    BOT=MAT(I,L)
    RR=TOP/BOT
    IF(R-RR)8,8,10
 10 R=RR
    IR=I
  8 CONTINUE
```

```
      IF(R-10.**20)14,13,14
   13 PRINT 202
  202 FORMAT(///97H EXIT CALLED SINCE A CLASS OF FEASIBLE SOLUTIONS WITH
     1 AN UNBOUNDED FUNCTIONAL CAN BE CONSTRUCTED.)
      GO TO 60
   14 DO 15 J=1,N
      IF(J-L)16,15,16
   16 LAMBDA=MAT(IR,J)
      DO 17 I=1,M
      A1=MAT(I,J)
      A2=MAT(IR,L)
      A3=MAT(I,L)
      A4=LAMBDA
      A5=LD
      A1=(A1*A2-A3*A4)/A5
      MAT(I,J)=A1+.1
      IF(MAT(I,J))150,650,17
  650 IF(A1)150,17,17
  150 MAT(I,J)=MAT(I,J)-1
   17 CONTINUE
   15 CONTINUE
      LD=MAT(IR,L)
      DO 18 I=1,M
   18 MAT(I,L)=-MAT(I,L)
      GO TO 20
   75 NUMITS=0
      SMPX=1.
    5 IF(SMPX)54,33,54
   33 NUMITS=NUMITS+1
      IF(MAXIT-NUMITS)988,988,984
  984 IF(KODE)911,34,911
  986 FORMAT(/114H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF IT
     1ERATIONS HAVE BEEN PERFORMED.THE SITUATION AT THIS STAGE WAS)
      GO TO 987
  911 ASSIGN 985 TO KK
      PRINT 904
```

```
      GO TO 903
   54 IT=IT+1
  988 PRINT 986
      IF(OUT)912,34,912
  912 ASSIGN 985 TO KK
      PRINT 904
      GO TO 903
  985 PRINT 983,LD
   34 DO 21 I=2,M
      IF(MAT(I,1))22,21,21
   21 CONTINUE
      IF(SMPX-1.)84,82,82
   82 ASSIGN 81 TO KK
      PRINT 83
   83 FORMAT(//53H FIRST PRIMAL AND DUAL CONTINUOUS FEASIBLE TABLEAU IS)
      GO TO 903
   81 PRINT 983,LD
      SMPX=0
   84 IR=0
      DO 23 I=I1,N
      NLD=MAT(I,1)/LD
      IFRAC=MAT(I,1)-LD*NLD
      IF(IFRAC)224,23,124
  224 IFRAC=LD+IFRAC
  124 IF(METHOD-1)24,125,126
  125 IR=I
      GO TO 39
  126 IF(METHOD-3)127,130,130
  127 IF(IR)129,128,129
  128 IR=I
      MAX=IFRAC
      GO TO 23
  129 IF(MAX-IFRAC)128,23,23
  130 IF(IR)132,131,132
  131 IR=I
      MIN=IFRAC-1
```

```
      GO TO 23
  132 IF(MIN-IFRAC+1)23,23,131
   23 CONTINUE
      IF(IR)39,50,39
   24 MIN=MAT(1,2)
      K=2
      DO 25 J=3,N
      IF(MIN-MAT(1,J))25,25,26
   26 MIN=MAT(1,J)
      K=J
   25 CONTINUE
      IFRST=10**10
      DO 27 I=I1,NINT
      IFRAC=MAT(I,K)-LD*(MAT(I,K)/LD)
      IF(IFRAC)32,28,30
   28 DO 29 J=1,N
      IF(K-J)65,29,65
   65 IF(MAT(I,J)-LD*(MAT(I,J)/LD))30,29,30
   29 CONTINUE
      GO TO 27
   32 IFRAC=IFRAC+LD
   30 IF(IFRST-IFRAC)27,27,31
   31 IFRST=IFRAC
      IR=I
      IF(IFRAC)27,39,27
   27 CONTINUE
   39 K=M+1
      KUTNO=KUTNO+1
      DO 42 J=1,N
      MAT(K,J)=LD*(MAT(IR,J)/LD)-MAT(IR,J)
      IF(MAT(K,J))93,42,93
   93 IF(MAT(IR,J))94,42,42
   94 MAT(K,J)=MAT(K,J)-LD
   42 CONTINUE
      I=K
   22 IR=I
```

```
      DO 43 J=2,N
      IF(MAT(IR,J))44,43,43
   43 CONTINUE
      PRINT 53
   53 FORMAT(///78H EXIT CALLED BECAUSE HYPERPLANE UNDER CONSIDERATION H
     1AS POSITIVE COEFFICIENTS.)
      GO TO 60
   44 TOP=MAT(1,J)
      BOT=-MAT(IR,J)
      SMALL=TOP/BOT
      L=J
      KEW(2)=0
      ITK=0
      IF(L-N)57,56,56
   57 JJ=L+1
      DO 45 I=JJ,N
      IF(MAT(IR,I))46,45,45
   46 TOP=MAT(1,I)
      BOT=-MAT(IR,I)
      F1=TOP/BOT
      IF(SMALL-F1)45,95,47
   95 IF(KOL-1)45,360,360
  360 ITK=ITK+1
      KEW(ITK+1)=I
      GO TO 45
   47 SMALL=F1
      L=I
      ITK=0
      KEW(2)=0
   45 CONTINUE
      IF(KEW(2))37,56,37
   37 KEW(1)=L
      ITK=ITK+1
      IF(KOL-2)431,432,433
  431 KIT=KIT+1
      L=KIT-ITK*(KIT/ITK)
```

```
      IF(L)61,62,61
   61 L=KEW(L)
      GO TO 56
  433 DO 434 J=1,ITK
      DO 435 I=2,M
      IF(MAT(I,KEW(J))+LD)435,436,435
  436 DO 437 JJ=1,N
      IF(JJ-KEW(J))438,437,438
  438 IF(MAT(I,JJ))435,437,435
  437 CONTINUE
      L=KEW(J)
      GO TO 56
  435 CONTINUE
  434 CONTINUE
      KOL=2
  432 RD=5.*RD
      IF(RD-RC)451,451,450
  450 RD=RD-RC
  451 IF(RD-RB)453,453,452
  452 RD=RD-RB
  453 IF(RD-RA)455,455,454
  454 RD=RD-RA
  455 F1=RE*RD
      TOP=ITK
      ITK=1.+F1*(TOP-1.)+.5
   62 L=KEW(ITK)
   56 DO 48 J=1,N
      IF(J-L)49,48,49
   49 LAMBDA=MAT(IR,J)
      DO 59 I=1,M
      A1=MAT(I,L)
      A2=LAMBDA
      A3=MAT(I,J)
      A4=MAT(IR,L)
      A5=LD
      A1=(A1*A2-A3*A4)/A5
```

```
      MAT(I,J)=A1+.1
      IF(MAT(I,J))151,651,59
  651 IF(A1)151,59,59
  151 MAT(I,J)=MAT(I,J)-1
   59 CONTINUE
   48 CONTINUE
      LD=-MAT(IR,L)
      GO TO 5
   50 PRINT 51
   51 FORMAT(///64H THE OPTIMAL INTEGER SOLUTION IS GIVEN IN THE FOLLOWI
     1NG TABLEAU.//)
  987 ASSIGN 200 TO KK
  903 GO TO (103,104,105,106,107,108,109,110,111,112,113,114,115,116,117
     1,118,119,120,721,722,723,724,725.726,727,728,729,730,731,732,733,7
     134,735,736,737,738),NN
  103 PRINT 52,((MAT(I,J),J=1,N),I=1,M)
   52 FORMAT(20X,3I8)
      GO TO KK,(982,81,985,200)
  104 PRINT 204,((MAT(I,J),J=1,N),I=1,M)
  204 FORMAT(18X,4I8)
      GO TO KK,(982,81,985,200)
  105 PRINT 205 ,((MAT(I,J),J=1,N),I=1,M)
  205 FORMAT(16X,5I8)
      GO TO KK,(982,81,985,200)
  106 PRINT 206,((MAT(I,J),J=1,N),I=1,M)
  206 FORMAT(14X,6I8)
      GO TO KK,(982,81,985,200)
  107 PRINT 207,((MAT(I,J),J=1,N),I=1,M)
  207 FORMAT(12X,7I8)
      GO TO KK,(982,81,985,200)
  108 PRINT 208,((MAT(I,J),J=1,N),I=1,M)
  208 FORMAT(10X,8I8)
      GO TO KK,(982,81,985,200)
  109 PRINT 209,((MAT(I,J),J=1,N),I=1,M)
  209 FORMAT(8X,9I12)
      GO TO KK,(982,81,985,200)
```

```
110 PRINT 210,((MAT(I,J),J=1,N),I=1,M)
210 FORMAT(6X,10I8)
    GO TO KK,(982,81,985,200)
111 PRINT 211,((MAT(I,J),J=1,N),I=1,M)
211 FORMAT(4X,11I8)
    GO TO KK,(982,81,985,200)
112 PRINT 212,((MAT(I,J),J=1,N),I=1,M)
212 FORMAT(2X,12I8)
    GO TO KK,(982,81,985,200)
113 PRINT 213,((MAT(I,J),J=1,N),I=1,M)
213 FORMAT(1X,13I9)
    GO TO KK,(982,81,985,200)
114 PRINT 214,((MAT(I,J),J=1,N),I=1,M)
214 FORMAT(1X,14I8)
    GO TO KK,(982,81,985,200)
115 PRINT 215,((MAT(I,J),J=1,N),I=1,M)
215 FORMAT(1X,15I8)
    GO TO KK,(982,81,985,200)
116 PRINT 216,((MAT(I,J),J=1,N),I=1,M)
216 FORMAT(1X,16I7)
    GO TO KK,(982,81,985,200)
117 PRINT 217,((MAT(I,J),J=1,N),I=1,M)
217 FORMAT(1X,12I10/71X,5I10//)
    GO TO KK,(982,81,985,200)
118 PRINT 218,((MAT(I,J),J=1,N),I=1,M)
218 FORMAT(1X,12I10/61X,6I10//)
    GO TO KK,(982,81,985,200)
119 PRINT 219,((MAT(I,J),J=1,N),I=1,M)
219 FORMAT(1X,12I10/51X,7I10//)
    GO TO KK,(982,81,985,200)
120 PRINT 220,((MAT(I,J),J=1,N),I=1,M)
220 FORMAT(1X,12I10/41X,8I10//)
    GO TO KK,(982,81,985,200)
721 PRINT 821,((MAT(I,J),J=1,N),I=1,M)
821 FORMAT(1X,12I10/31X,9I10//)
    GO TO KK,(982,81,985,200)
```

```
722 PRINT 822,((MAT(I,J),J=1,N),I=1,M)
822 FORMAT(1X,12I10/21X,10I10//)
    GO TO KK,(982,81,985,200)
723 PRINT 823,((MAT(I,J),J=1,N),I=1,M)
823 FORMAT(1X,12I10/11X,11I10//)
    GO TO KK,(982,81,985,200)
724 PRINT 824,((MAT(I,J),J=1,N),I=1,M)
824 FORMAT(2(1X,12I10/)/)
    GO TO KK,(982,81,985,200)
725 PRINT 825,((MAT(I,J),J=1,N),I=1,M)
825 FORMAT(2(1X,12I10/),111X,I10//)
    GO TO KK,(982,81,985,200)
726 PRINT 826,((MAT(I,J),J=1,N),I=1,M)
826 FORMAT(2(1X,12I10/),101X,2I10//)
    GO TO KK,(982,81,985,200)
727 PRINT 827,((MAT(I,J),J=1,N),I=1,M)
827 FORMAT(2(1X,12I10/),91X,3I10//)
    GO TO KK,(982,81,985,200)
728 PRINT 828,((MAT(I,J),J=1,N),I=1,M)
828 FORMAT(2(1X,12I10/),81X,4I10//)
    GO TO KK,(982,81,985,200)
729 PRINT 829,((MAT(I,J),J=1,N),I=1,M)
829 FORMAT(2(1X,12I10/),71X,5I10//)
    GO TO KK,(982,81,985,200)
730 PRINT 830,((MAT(I,J),J=1,N),I=1,M)
830 FORMAT(2(1X,12I10/),61X,6I10//)
    GO TO KK,(982,81,985,200)
731 PRINT 831,((MAT(I,J),J=1,N),I=1,M)
831 FORMAT(2(1X,12I10/),51X,7I10//)
    GO TO KK,(982,81,985,200)
732 PRINT 832,((MAT(I,J),J=1,N),I=1,M)
832 FORMAT(2(1X,12I10/),41X,8I10//)
    GO TO KK,(982,81,985,200)
733 PRINT 833,((MAT(I,J),J=1,N),I=1,M)
833 FORMAT(2(1X,12I10/),31X,9I10//)
    GO TO KK,(982,81,985,200)
```

```
734 PRINT 834,((MAT(I,J),J=1,N),I=1,M)
834 FORMAT(2(1X,12I10/),21X,10I10//)
    GO TO KK,(982,81,985,200)
735 PRINT 835,((MAT(I,J),J=1,N),I=1,M)
835 FORMAT(2(1X,12I10/),11X,11I10//)
    GO TO KK,(982,81,985,200)
736 PRINT 836,((MAT(I,J),J=1,N),I=1,M)
836 FORMAT(3(1X,12I10/)//)
    GO TO KK,(982,81,985,200)
737 PRINT 837,((MAT(I,J),J=1,N),I=1,M)
837 FORMAT(3(1X,12I10/),1X,I10//)
    GO TO KK,(982,81,985,200)
738 PRINT 838,((MAT(I,J),J=1,N),I=1,M)
838 FORMAT(3(1X,12I10/),1X,2I10//)
    GO TO KK,(982,81,985,200)
200 PRINT 154,LD,IT,NUMITS,KUTNO
154 FORMAT(16H WITH DIVISOR = ,I4,1H,//25H WHICH WAS OBTAINED AFTER,I6
   1,35H ITERATIONS IN THE CONTINUOUS PHASE,I6,32H ITERATIONS IN THE I
   1NTEGER PHASE/30H AND THE ADDITION OF A FURTHER,I6,13H CONSTRAINTS.
   1)
    NUM=NUM+1
    IF(NOJOBS-NUM)60,60,303
 60 CALL EXIT
    END
```

FORTRAN PROGRAM LISTING.

PURE INTEGER CODE
(OPTIMUM CUT).

```
      DIMENSION MAT(66,38),IA(38),KR(37),IRES(38)
      RA=34359738368.
      RB=68719476736.
      RC=137438953472.
      RE=1./RA
      READ 300,NOJOBS,LIMIT
  300 FORMAT(2I3)
      NUM=0
  303 READ 1,M,N,LD,OUT,KODE,MAXIT,NOROS,METHOD,KOL,MZED,IS,(MAT(1,J),J=
     11,N),((MAT(I,J),J=1,N),I=N+1,M)
    1 FORMAT(3I6,F2.0,I1,I6,4I1,I2/(10I8))
      RD=13543288579.
      KIT=0
      KUTNO=0
      NN=N-2
      DO 301 I=2,N
      DO 301 J=1,N
  301 MAT(I,J)=0
      DO 302 I=2,N
  302 MAT(I,I)=-1
      IF(MZED)461,460,461
  460 I1=1
      GO TO 462
  461 I1=2
  462 IT=-1
      PRINT 80,IS,METHOD,KOL
   80 FORMAT(1H1//40H OPTIMUM CUT ROUTINE SOLUTION TO PROBLEM,I3,47H USI
     1NG THE VARIANTS DEFINED BY PARAMETER CODE (,I1,1H,,I1,2H).)
      IF(MZED)581,580,581
  580 PRINT 582
```

```
582 FORMAT(64H THE OBJECTIVE FUNCTION IS ADMITTED AS A POSSIBLE CUT GE
   1NERATOR.)
    GO TO 583
581 PRINT 584
584 FORMAT(56H THE OBJECTIVE FUNCTION IS EXCLUDED FROM CUT GENERATION.
   1)
583 IF(NOROS)310,309,310
309 NINT=N
    GO TO 20
310 NINT=M
 20 MIN=MAT(1,2)
    L=2
    DO 2 J=3,N
    IF(MIN-MAT(1,J))2,2,3
  3 MIN=MAT(1,J)
    L=J
  2 CONTINUE
    IF(MIN)4,75,75
  4 R=10.**20
    IT=IT+1
    IF(OUT)901,902,901
901 ASSIGN 982 TO KK
    PRINT 904
904 FORMAT(//19H CURRENT TABLEAU IS)
    GO TO 903
982 PRINT 983,LD
983 FORMAT(16H WITH DIVISOR = ,I8,1H.)
902  DO 8 I=2,M
    IF(MAT(I,L))8,8,9
  9 TOP=MAT(I,1)
    BOT=MAT(I,L)
    RR=TOP/BOT
    IF(R-RR)8,8,10
 10 R=RR
    IR=I
  8 CONTINUE
```

```
      IF(R-10.**20)14,13,14
   13 PRINT 202
  202 FORMAT(///97H EXIT CALLED SINCE A CLASS OF FEASIBLE SOLUTIONS WITH
     1 AN UNBOUNDED FUNCTIONAL CAN BE CONSTRUCTED.)
      GO TO 60
   14 DO 15 J=1,N
      IF(J-L)16,15,16
   16 LAMBDA=MAT(IR,J)
      DO 17 I=1,M
      A1=MAT(I,J)
      A2=MAT(IR,L)
      A3=MAT(I,L)
      A4=LAMBDA
      A5=LD
      A1=(A1*A2-A3*A4)/A5
      MAT(I,J)=A1+.1
      IF(MAT(I,J))150,650,17
  650 IF(A1)150,17,17
  150 MAT(I,J)=MAT(I,J)-1
   17 CONTINUE
   15 CONTINUE
      LD=MAT(IR,L)
      DO 18 I=1,M
   18 MAT(I,L)=-MAT(I,L)
      GO TO 20
   75 NUMITS=0
      SMPX=1.
    5 IF(SMPX)54,33,54
   33 NUMITS=NUMITS+1
      IF(MAXIT-NUMITS)988,988,984
  984 IF(KODE)911,34,911
  988 PRINT 986
  986 FORMAT(/113H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF IT
     1ERATIONS HAS BEEN PERFORMED.THE SITUATION AT THIS STAGE WAS)
      GO TO 987
  911 ASSIGN 985 TO KK
```

```
      PRINT 904
      GO TO 903
   54 IT=IT+1
      IF(OUT)912,34,912
  912 ASSIGN 985 TO KK
      PRINT 904
      GO TO 903
  985 PRINT 983,LD
   34 DO 21 I=2,M
      IF(MAT(I,1))22,21,21
   21 CONTINUE
      IF(SMPX-1.)84,82,82
   82 ASSIGN 81 TO KK
      PRINT 83
   83 FORMAT(//53H FIRST PRIMAL AND DUAL CONTINUOUS FEASIBLE TABLEAU IS)
      GO TO 903
   81 PRINT 983,LD
      SMPX=0
   84 IR=0
      DO 23 I=I1,N
      NLD=MAT(I,1)/LD
      IFRAC=MAT(I,1)-LD*NLD
      IF(IFRAC)224,23,124
  224 IFRAC=LD+IFRAC
  124 IF(METHOD-1)24,125,126
  125 IR=I
      GO TO 39
  126 IF(METHOD-3)127,130,130
  127 IF(IR)129,128,129
  128 IR=I
      MAX=IFRAC
      GO TO 23
  129 IF(MAX-IFRAC)128,23,23
  130 IF(IR)132,131,132
  131 IR=I
      MIN=IFRAC-1
```

```
      GO TO 23
  132 IF(MIN-IFRAC+1)23,23,131
   23 CONTINUE
      IF(IR)39,50,39
   24 MIN=MAT(1,2)
      K=2
      DO 25 J=3,N
      IF(MIN-MAT(1,J))25,25,26
   26 MIN=MAT(1,J)
      K=J
   25 CONTINUE
      IFRST=10**10
      DO 27 I=I1,NINT
      IFRAC=MAT(I,K)-LD*(MAT(I,K)/LD)
      IF(IFRAC)32,28,30
   28 DO 29 J=1,N
      IF(K-J)65,29,65
   65 IF(MAT(I,J)-LD*(MAT(I,J)/LD))30,19,30
   19 IF(J-1)29,27,29
   29 CONTINUE
      GO TO 27
   32 IFRAC=IFRAC+LD
   30 IF(IFRST-IFRAC)27,27,31
   31 IFRST=IFRAC
      IR=I
      IF(IFRAC)27,39,27
   27 CONTINUE
   39 K=M+1
      KUTNO=KUTNO+1
      DO 42 J=1,N
      MAT(K,J)=LD*(MAT(IR,J)/LD)-MAT(IR,J)
      IF(MAT(K,J))93,42,93
   93 IF(MAT(IR,J))94,42,42
   94 MAT(K,J)=MAT(K,J)-LD
   42 CONTINUE
C     SOLUTION OF DIOPHANTINE EQUATION IN NON NEGATIVE VARIABLES.
```

```
      IA(N)=-MAT(K,1)
      DO 790 I=1,N-1
  790 IA(I)=-MAT(K,I+1)
      DO 715 J=1,N-2
      DO 714 I=J+1,N-1
      IF(IA(J)-IA(I))716,714,714
  716 JF=IA(J)
      IA(J)=IA(I)
      IA(I)=JF
  714 CONTINUE
  715 CONTINUE
      NM=0
  701 IRO=IA(N)+NM*LD
      LEFT=1
      IRES(1)=IRO
      JJ=1
   70 DO 706 J=JJ,N-1
      IF(LEFT-1)704,702,704
  702 IF(IA(J))708,706,708
  708 KR(J)=IRES(J)/IA(J)
      IF(KR(J))703,705,703
  704 LEFT=1
  703 IRES(J+1)=IRES(J)-KR(J)*IA(J)
      IF(IRES(J+1))706,709,706
  705 IRES(J+1)=IRES(J)
  706 CONTINUE
      DO 710 I=2,N-1
      IF(KR(N-I))707,710,707
  710 CONTINUE
      NM=NM+1
      GO TO 701
  707 JJ=N-I
      KR(JJ)=KR(JJ)-1
      LEFT=0
      GO TO 70
  709 MAT(K,1)=-IRO
```

```
      I=K
   22 IR=I
      DO 43 J=2,N
      IF(MAT(IR,J))44,43,43
   43 CONTINUE
      PRINT 53
   53 FORMAT(///78H EXIT CALLED BECAUSE HYPERPLANE UNDER CONSIDERATION H
     1AS POSITIVE COEFFICIENTS.)
      GO TO 60
   44 TOP=MAT(1,J)
      BOT=-MAT(IR,J)
      SMALL=TOP/BOT
      L=J
      IA(2)=0
      ITK=0
      IF(L-N)57,56,56
   57 JJ=L+1
      DO 45 I=JJ,N
      IF(MAT(IR,I))46,45,45
   46 TOP=MAT(1,I)
      BOT=-MAT(IR,I)
      F1=TOP/BOT
      IF(SMALL-F1)45,95,47
   95 IF(KOL-1)45,360,360
  360 ITK=ITK+1
      IA(ITK+1)=I
      GO TO 45
   47 SMALL=F1
      L=I
      ITK=0
      IA(2)=0
   45 CONTINUE
      IF(IA(2))37,56,37
   37 IA(1)=L
      ITK=ITK+1
      IF(KOL-2)431,432,433
```

```
431 KIT=KIT+1
    L=KIT-ITK*(KIT/ITK)
    IF(L)61,62,61
 61 L=IA(L)
    GO TO 56
433 DO 434 J=1,ITK
    DO 435 I=2,M
    IF(MAT(I,IA(J))+LD)435,436,435
436 DO 437 JJ=1,N
    IF(JJ-IA(J))438,437,438
438 IF(MAT(I,JJ))435,437,435
437 CONTINUE
    L=IA(J)
    GO TO 56
435 CONTINUE
434 CONTINUE
    KOL=2
432 RD=5.*RD
    IF(RD-RC)451,451,450
450 RD=RD-RC
451 IF(RD-RB)453,453,452
452 RD=RD-RB
453 IF(RD-RA)455,455,454
454 RD=RD-RA
455 F1=RE*RD
    TOP=ITK
    ITK=1.+F1*(TOP-1.)+.5
 62 L=IA(ITK)
 56 DO 48 J=1,N
    IF(J-L)49,48,49
 49 LAMBDA=MAT(IR,J)
    DO 59 I=1,M
    A1=MAT(I,L)
    A2=LAMBDA
    A3=MAT(I,J)
    A4=MAT(IR,L)
```

```
      A5=LD
      A1=(A1*A2-A3*A4)/A5
      MAT(I,J)=A1+.1
      IF(MAT(I,J))151,651,59
  651 IF(A1)151,59,59
  151 MAT(I,J)=MAT(I,J)-1
   59 CONTINUE
   48 CONTINUE
      LD=-MAT(IR,L)
      GO TO 5
   50 PRINT 51
   51 FORMAT(///64H THE OPTIMAL INTEGER SOLUTION IS GIVEN IN THE FOLLOWI
     1NG TABLEAU.//)
  987 ASSIGN 200 TO KK
  903 GO TO(103,104,105,106,107,108,109,110,111,112,113,114,115,116),NN
  103 PRINT 52,((MAT(I,J),J=1,N),I=1,M)
   52 FORMAT(20X,3I8)
      GO TO KK,(982,81,985,200)
  104 PRINT 204,((MAT(I,J),J=1,N),I=1,M)
  204 FORMAT(18X,4I8)
      GO TO KK,(982,81,985,200)
  105 PRINT 205 ,((MAT(I,J),J=1,N),I=1,M)
  205 FORMAT(16X,5I8)
      GO TO KK,(982,81,985,200)
  106 PRINT 206,((MAT(I,J),J=1,N),I=1,M)
  206 FORMAT(14X,6I8)
      GO TO KK,(982,81,985,200)
  107 PRINT 207,((MAT(I,J),J=1,N),I=1,M)
  207 FORMAT(12X,7I8)
      GO TO KK,(982,81,985,200)
  108 PRINT 208,((MAT(I,J),J=1,N),I=1,M)
  208 FORMAT(10X,8I8)
      GO TO KK,(982,81,985,200)
  109 PRINT 209,((MAT(I,J),J=1,N),I=1,M)
  209 FORMAT(8X,9I8)
      GO TO KK,(982,81,985,200)
```

```
110 PRINT 210,((MAT(I,J),J=1,N),I=1,M)
210 FORMAT(6X,10I8)
    GO TO KK,(982,81,985,200)
111 PRINT 211,((MAT(I,J),J=1,N),I=1,M)
211 FORMAT(4X,11I8)
    GO TO KK,(982,81,985,200)
112 PRINT 212,((MAT(I,J),J=1,N),I=1,M)
212 FORMAT(2X,12I8)
    GO TO KK,(982,81,985,200)
113 PRINT 213,((MAT(I,J),J=1,N),I=1,M)
213 FORMAT(1X,13I9)
    GO TO KK,(982,81,985,200)
114 PRINT 214,((MAT(I,J),J=1,N),I=1,M)
214 FORMAT(1X,14I8)
    GO TO KK,(982,81,985,200)
115 PRINT 215,((MAT(I,J),J=1,N),I=1,M)
215 FORMAT(1X,15I8)
    GO TO KK,(982,81,985,200)
116 PRINT 216,((MAT(I,J),J=1,N),I=1,M)
216 FORMAT(1X,16I7)
    GO TO KK,(982,81,985,200)
200 PRINT 154,LD,IT,NUMITS,KUTNO
154 FORMAT(16H WITH DIVISOR = ,I4,1H,//25H WHICH WAS OBTAINED AFTER,I6
   1,35H ITERATIONS IN THE CONTINUOUS PHASE,I6,32H ITERATIONS IN THE I
   1NTEGER PHASE/30H AND THE ADDITION OF A FURTHER,I6,13H CONSTRAINTS.
   1)
    NUM=NUM+1
    IF(NOJOBS-NUM)60,60,303
 60 CALL EXIT
    END
```

FORTRAN PROGRAM LISTING.

CONVEX ISOLATION AND BOUND CONTRACTION CODE.

```
      DIMENSION A(70,20),SOLN(31),OC(31,20),MIN(19),MAX(19),MBTO(500,2),
     1V(19,19),KON(15),KEW(20),KEWROW(10),KEWCOL(10)
      RA=34359738368.
      RB=68719476736.
      RC=137438953472.
      RE=1./RA
      JOB=0
      READ 3,NOJORS,BOUND,LIMSTO
  503 READ 1,M,N,INTER,MAXIT,KOL,MZED,IS,NINT,GCD,LRANGE,IVERS
    1 FORMAT(2I6,8X,I1,I6,2X,2I1,I2,7X,I6,12X,F6.3,2I1)
      READ 2,(A(1,J),J=1,N),((A(I,J),J=1,N),I=N+1,M)
      RD=13543288579.
      DO 291 I=2,N
      DO 291 J=1,N
      IF(I-J)293,292,293
  293 A(I,J)=0
      GO TO 291
  292 A(I,J)=-1.
  291 CONTINUE
    2 FORMAT(10F8.0)
    3 FORMAT(I3,F9.9,I6)
      DO 960 I=1,M
      DO 960 J=1,N
  960 OC(I,J)=A(I,J)
      LIST=0
      NUMHYP=0
      IL=-1
      IFULL=0
      NUMITS=0
      KOUNT=0
      MRM=M
```

```
      BEST=-10.**20
      BDC=BEST
      NOCYC=0
      ISTAGE=0
      ROFF=.5-BOUND
      NN=N-2
      PRINT 262,IS,KOL,LRANGE,IVERS
  262 FORMAT(1H1//60H DISCRETE EXTREMA.SUBCONVEX ISOLATION AND BOUND CON
     1TRACTION./20H SOLUTION TO PROBLEM,I3,26H USING VARIANTS....COLUMN(
     1,I1,2H),13H HYPOTHESIS (,I1,15H) AND VERSION (,I1,2H).)
   25 SMALL=A(1,2)
      L=2
      DO 4 J=3,N
      IF(SMALL-A(1,J))4,4,5
    5 SMALL=A(1,J)
      L=J
    4 CONTINUE
      IF(SMALL)6,26,26
    6 RK=-BEST
      DO 8 I=2,M
      IF(A(I,L))8,8,9
    9 RR=A(I,1)/A(I,L)
      FLO=RK-RR
      IF(ABS(FLO)-ROFF)8,8,159
  159 IF(FLO)8,8,10
   10 RK=RR
      IR=I
    8 CONTINUE
      IF(RK+BEST)14,13,14
   13 PRINT 202
  202 FORMAT(//40H EXIT CALLED SINCE FUNCTIONAL UNBOUNDED.)
      GO TO 506
   14 PIVOT=A(IR,L)
   94 DO 15 I=1,MRM
      IF(I-IR)17,15,17
   17 F=-A(I,L)/PIVOT
```

```
      DO 16 J=1,N
      IF(J-L)18,19,18
   18 A(I,J)=A(I,J)+F*A(IR,J)
      IF(ABS(A(I,J))-ROFF)1500,1500,16
 1500 A(I,J)=0
      GO TO 16
   19 A(I,J)=F
   16 CONTINUE
   15 CONTINUE
      NUMITS=NUMITS+1
   20 DO 22 J=1,N
      IF(J-L)24,23,24
   23 A(IR,J)=-1.
      GO TO 22
   24 A(IR,J)=0
   22 CONTINUE
      IF(INTER)91,90,91
   91 PRINT 92
   92 FORMAT(//19H CURRENT TABLEAU IS)
   93 GO TO (81,82,83,84,85,86,87,88,88,88,88,88,88,599),NN
   81 PRINT 701,((A(I,J),J=1,N),I=1,M)
  701 FORMAT( 20X,3F11.6)
      GO TO 90
   82 PRINT 702,((A(I,J),J=1,N),I=1,M)
  702 FORMAT( 18X,4F11.6)
      GO TO 90
   83 PRINT 703,((A(I,J),J=1,N),I=1,M)
  703 FORMAT( 16X,5F11.6)
      GO TO 90
   84 PRINT 704,((A(I,J),J=1,N),I=1,M)
  704 FORMAT( 14X,6F11.6)
      GO TO 90
   85 PRINT 705,((A(I,J),J=1,N),I=1,M)
  705 FORMAT( 12X,7F11.6)
      GO TO 90
   86 PRINT 706,((A(I,J),J=1,N),I=1,M)
```

```
706 FORMAT( 10X,8F11.6)
    GO TO 90
 87 PRINT 707,((A(I,J),J=1,N),I=1,M)
707 FORMAT( 8X,9F11.6)
    GO TO 90
 88 PRINT 708,((A(I,J),J=1,N),I=1,M)
708 FORMAT( 6X,10F11.6)
    GO TO 90
599 PRINT 510,((A(I,J),J=1,N),I=1,M)
510 FORMAT(1X,16F7.3)
 90 IF(NUMITS-MAXIT)194,195,195
194 IF(ISTAGE)26,25,26
195 IF(ISTAGE)197,196,197
196 PRINT 198
198 FORMAT(//92H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF AL
   1LOWABLE ITERATIONS HAS BEEN PERFORMED./44H THE INTEGER PHASE HAS N
   1OT YET BEEN ENTERED.)
    GO TO 506
197 PRINT 199
199 FORMAT(//92H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF AL
   1LOWABLE ITERATIONS HAS BEEN PERFORMED.)
    IF(BEST+BDC)301,302,301
302 PRINT 303
303 FORMAT(89H THE INTEGER PHASE HAS BEEN ENTERED BUT NO INTEGER SOLUT
   1ION HAS BEEN FOUND AT THIS STAGE.)
    GO TO 506
301 PRINT 304,(SOLN(I),I=1,M-1),BEST,KOUNT
304 FORMAT(/25H BEST SOLUTION TO DATE IS//(12F10.4)//28H WITH FUNCTION
   1AL VALUE XO = ,F10.4/15H THIS REQUIRED ,I6,12H ITERATIONS.)
    GO TO 506
 26 DO 21 I=2,MRM
    IF(A(I,1))51,21,21
 21 CONTINUE
    IF(ISTAGE)28,27,28
 27 ISTAGE=1
    MRM=M+2*NINT
```

```
      MRM1=MRM+1
      DO 2119 I=M+1,MRM
      DO 2119 J=1,N
 2119 A(I,J)=0
      PRINT 200,NUMITS,(A(I,1),I=2,M)
  200 FORMAT(//58H PRIMAL AND DUAL CONTINUOUS FEASIBILITY WAS ATTAINED A
     1FTER,I4,37H ITERATIONS IN THE SIMPLEX ALGORITHM./34H THE CONTINUOU
     1S SOLUTION POINT WAS//6X,8F14.6/6X,8F14.6/6X,8F14.6/6X,8F14.6)
      PRINT 201,A(1,1)
  201 FORMAT(/31H PRODUCING THE FUNCTIONAL VALUE//30X,6H XO = ,F10.4//45
     1H PROGRAMME NOW PROCEEDS TO INTEGRALITY PHASE.)
      DO 222 I=2,NINT
      MAX(I)=10**20
  222 MIN(I)=0
C     OPTIMAL VERTEX
      OPCON=A(1,1)
      IJ=1
      NUMKON=0
      DO 400 I=2,M
      IF(ABS(A(I,1))-ROFF)402,402,403
  402 NONZER=0
      DO 7232 J=2,N
      IF(ABS(A(I,J))-ROFF)7232,7232,7231
 7231 NONZER=NONZER+1
 7232 CONTINUE
      IF(NONZER-1)403,7402,403
 7402 KON(IJ)=I
      IJ=IJ+1
      NUMKON=NUMKON+1
  403 IF(I-NINT)404,404,400
  404 PORT=0
      STABD=0
      DO 405 J=2,N
      IF(A(I,J))406,405,407
  406 PORT=1.
      GO TO 405
```

```
407 STABD=1.
405 CONTINUE
    IF(PORT)409,408,409
408 FRAC=DIFF(A(I,1))
    DEV=ABS(.5-FRAC)
    IF(DEV-BOUND)420,421,421
420 MAX(I)=INT(A(I,1))
    GO TO 400
421 MAX(I)=INT(A(I,1)+.1)
    GO TO 400
409 IF(STABD)400,410,400
410 FRAC=DIFF(A(I,1))
    DEV=ABS(.5-FRAC)
    IF(DEV-BOUND)411,412,412
411 MIN(I)=INT(A(I,1))
    MIN(I)=MIN(I)+1
    GO TO 400
412 MIN(I)=INT(A(I,1)+.1)
400 CONTINUE
    DO 755 I=1,NUMKON
755 V(I,1)=A(I+1,1)
 28 K=0
    DO 30 I=2,NINT
    FRAC=DIFF(A(I,1))
    DEV=ABS(.5-FRAC)
    IF(DEV-BOUND)31,30,30
 31 JU=0
    JL=0
    DO 60 J=2,N
    IF(A(I,J))61,60,62
 61 IF(JU)64,63,64
 63 JU=1
    DU=-A(1,J)/A(I,J)
    GO TO 60
 64 F=-A(1,J)/A(I,J)
    FLO=DU-F
```

```
      IF(ABS(FLO)-ROFF)60,60,2110
 2110 IF(FLO)60,60,65
   65 DU=F
      GO TO 60
   62 IF(JL)67,66,67
   66 JL=1
      DD=A(1,J)/A(I,J)
      GO TO 60
   67 F=A(1,J)/A(I,J)
      FLO=DD-F
      IF(ABS(FLO)-ROFF)60,60,2111
 2111 IF(FLO)60,60,68
   68 DD=F
   60 CONTINUE
      IF(JL)70,69,70
   69 IF(LIST)431,430,431
  430 MIN(I)=INT(A(I,1))
      MIN(I)=MIN(I)+1
  431 LDIRT=-1
 7400 K=I
      LDIR=LDIRT
      GO TO 35
   70 COMP=DD*FRAC
      LDIRT=1
      IF(JU)71,74,71
   74 IF(LIST)7400,433,7400
  433 MAX(I)=INT(A(I,1))
      GO TO 7400
   71 F=DU*(1.-FRAC)
      FLO=COMP-F
      IF(ABS(FLO)-ROFF)73,73,2112
 2112 IF(FLO)72,73,73
   72 COMP=F
      LDIRT=2
      GO TO 432
   73 LDIRT=-2
```

```
      GO TO 432
  432 IF(K)75,76,75
   75 FLO=BIG-COMP
      IF(ABS(FLO)-ROFF)30,30,2113
 2113 IF(FLO)76,30,30
   76 BIG=COMP
      K=I
      LDIR=LDIRT
   30 CONTINUE
      IF(K)35,54,35
   54 PRINT 907,(A(I,1),I=2,M)
  907 FORMAT(//27H THE FIRST INTEGER SOLUTION//6X,8F14.6/6X,8F14.6/6X,8F
     114.6/6X,8F14.6)
      PRINT 908,A(1,1),NUMITS
  908 FORMAT(/31H PRODUCING THE FUNCTIONAL VALUE//30X,6H XO = ,F10.4//19
     1H WAS ATTAINED AFTER,I6,12H ITERATIONS.)
      DO 1055 J=2,N
      KEW(J)=MIN(J)
 1055 KEW(10+J)=MAX(J)
      BEST=A(1,1)
      DO 34 I=2,M
      J=I-1
   34 SOLN(J)=A(I,1)
      KOUNT=NUMITS
      NUMITS=0
      IF(LIST)560,500,560
   35 INTA=INT(A(K,1))
      FLO=INTA
      II=M+K
      IF(LDIR)36,36,37
   36 II=II+NINT
      IF(LDIR+2)39,38,39
   38 LIST=LIST+1
   39 A(II,1)=A(K,1)-FLO-1.
      DO 40 J=2,N
   40 A(II,J)=A(K,J)
```

```
      GO TO 43
   37 IF(LDIR-2)32,531,32
  531 LIST=LIST+1
   32 A(II,1)=FLO-A(K,1)
      DO 41 J=2,N
   41 A(II,J)=-A(K,J)
   43 I=II
      NOINF=0
   51 IR=I
      NO=0
      KEW(2)=0
      DO 45 J=2,N
      IF(A(IR,J))46,45,45
   46 IF(NO)48,47,48
   47 NO=1
      RATIO=-A(1,J)/A(IR,J)
      L=J
      KEW(2)=0
      ITK=0
      GO TO 45
   48 F=-A(1,J)/A(IR,J)
      FLO=RATIO-F
      IF(ABS(FLO)-ROFF)945,945,2114
  945 IF(KOL-2)45,946,45
  946 ITK=ITK+1
      KEW(ITK+1)=J
      GO TO 45
 2114 IF(FLO)45,45,49
   49 RATIO=F
      L=J
      KEW(2)=0
      ITK=0
   45 CONTINUE
      IF(KEW(2))948,947,948
  947 IF(NO)89,50,89
  948 KEW(1)=L
```

```
      ITK=ITK+1
      RD=5.*RD
      IF(RD-RC)949,949,950
  950 RD=RD-RC
  949 IF(RD-RB)951,951,952
  952 RD=RD-RB
  951 IF(RD-RA)953,953,954
  954 RD=RD-RA
  953 F1=RE*RD
      TOP=ITK
      ITK=1.+F1*(TOP-1.)+.5
      L=KEW(ITK)
   89 IF(ISTAGE)555,14,555
  555 NOINF=NOINF+1
      KEWROW(NOINF)=IR
      KEWCOL(NOINF)=L
      JK=NOINF+MRM
      DO 556 J=1,N
  556 A(JK,J)=A(IR,J)
      GO TO 14
   50 IF(ISTAGE)52,53,52
   53 PRINT 900
  900 FORMAT(//48H CONTINUOUS PROBLEM HAS NO SOLUTION.EXIT CALLED.)
      GO TO 506
   52 LIST=LIST-1
C     RECOVER TABLEAU AND REVERSE CONSTRAINTS.
      DO 1605 IJ=1,NOINF
      IJK=NOINF-IJ+1
      IQ=KEWROW(IJK)
      LQ=KEWCOL(IJK)
      MQ=MRM+IJK
      DO 605 I=1,MRM
      DO 603 J=1,N
      IF(J-LQ)604,603,604
  604 A(I,J)=A(I,J)-A(I,LQ)*A(MQ,J)
  603 CONTINUE
```

```
      A(I,LQ)=-A(I,LQ)*A(MQ,LQ)
  605 CONTINUE
 1605 CONTINUE
      NOINF=0
      II=IQ
      DO 607 J=1,N
  607 A(II,J)=0
      IF(II-M-NINT)608,608,609
  608 II=II+NINT
      GO TO 661
  609 II=II-NINT
  661 A(II,1)=-1.-A(MRM1,1)
      DO 554 J=2,N
  554 A(II,J)=-A(MRM1,J)
      IF(LIST)43,57,43
   57 IF(LDIR+2)58,58,59
   58 MAX(K)=INT(A(K,1))
      GO TO 43
   59 MIN(K)=INT(A(K,1))
      MIN(K)=MIN(K)+1
      GO TO 43
C     CONVEX ISOLATION
  560 IF(GCD)234,177,234
  177 Q1=ABS(OC(1,2))
      DO 561 J=3,N
      Q2=ABS(OC(1,J))
      IF(Q1-Q2)563,562,562
  562 F=Q1
      Q1=Q2
      Q2=F
  563 I=INT(Q2/Q1)
      F=I
      REM=Q2-F*Q1
      IF(REM-.5+BOUND)561,561,564
  564 Q2=Q1
      Q1=REM
```

```
      GO TO 563
  561 CONTINUE
      GCD=Q1
  234 FC=A(1,1)+GCD
      IF(FC-OPCON)565,500,500
C     VERTICES
  565 DO 566 J=1,N-1
  566 A(M+1,J)=OC(1,J+1)
      A(M+1,N)=-FC
      MISS=1
      IRPT=0
      KK=2
   79 DO 567 J=1,N
  567 A(1,J)=A(M+1,J)
      L=1
      DO 568 I=1,NUMKON
      IF(I-MISS)569,568,569
  569 II=KON(I)
      L=L+1
      DO 570 J=1,N-1
  570 A(L,J)=OC(II,J+1)
      A(L,N)=OC(II,1)
  568 CONTINUE
      IF(IRPT)1078,1077,1078
 1078 DO 1079 J=1,N
 1079 A(IRPT,J)=A(M,J)
C     INVERSION
 1077 CALL MA02B (A,NUMKON,70)
      SMALL=0
      DO 77 K=1,NUMKON
      IF(IRPT)77,3058,77
 3058 IF(A(K,N))3055,77,77
 3055 IF(SMALL-A(K,N))77,77,3056
 3056 SMALL=A(K,N)
      K1=K
   77 V(K,KK)=A(K,N)
```

```
      IF(SMALL)2077,3057,2077
 3057 KK=KK+1
      IF(IRPT)2079,2087,2079
 2079 IF(IRPT-NUMKON)2077,2086,2077
 2077 IRPT=IRPT+1
      DO 2085 J=1,N
 2085 A(M,J)=0
      A(M,K1)=-1.
      GO TO 79
 2086 IRPT=0
 2087 IF(MISS-NUMKON)78,80,78
   78 MISS=MISS+1
      GO TO 79
C     CONSTRUCT BOUNDS
   80 DO 95 I=1,NUMKON
      SMALL=V(I,1)
      BIG=V(I,1)
      DO 97 J=2,KK-1
      IF(V(I,J)-BIG)96,97,99
   96 IF(V(I,J)-SMALL)98,97,97
   98 SMALL=V(I,J)
      GO TO 97
   99 BIG=V(I,J)
   97 CONTINUE
      IF(SMALL)320,321,321
  320 SMALL=0
      IF(BIG)322,321,321
  322 BIG=0
  321 A(I,1)=BIG
      FRAC=DIFF(A(I,1))
      DEV=ABS(.5-FRAC)
      IF(DEV-BOUND)104,105,105
  104 LUB=INT(BIG)
      GO TO 106
  105 LUB=INT(BIG+ROFF)
  106 A(I,1)=SMALL
```

```
      FRAC=DIFF(A(I,1))
      DEV=ABS(.5-FRAC)
      IF(DEV-BOUND)107,108,108
  107 LOB=INT(SMALL*1.)
      GO TO 109
  108 LOB=INT(SMALL*ROFF)
  109 IF(MAX(I+1)-LOB)500,111,111
  111 IF(MIN(I+1)-LOB)112,113,113
  112 MIN(I+1)=LOB
  113 IF(MIN(I+1)-LUB)114,114,500
  114 IF(MAX(I+1)-LUB)95,95,115
  115 MAX(I+1)=LUB
   95 CONTINUE
C     CONTRACT BOUNDS
  232 CHANGE=0
      DELTA=-FC
      II=1
  175 DO 150 I=1,NUMKON+1
      IF(I-1)151,152,151
  151 II=KON(I-1)
      DELTA=OC(II,1)
  152 DO 153 J=2,N
      IF(OC(II,J))154,153,155
  154 B=MAX(J)
      GO TO 156
  155 B=MIN(J)
  156 DELTA=DELTA-OC(II,J)*B
  153 CONTINUE
      IF(ABS(DELTA)-ROFF)8169,8169,8168
 8169 DELTA=0
      GO TO 270
 8168 IF(DELTA)161,270,162
  161 IF(NUMHYP)308,500,308
  162 DO 160 J=2,N
      IF(ABS(OC(II,J))-ROFF)160,160,363
  363 F=ABS(OC(II,J))
```

```
      L=INT(DELTA/F)
      IF(ABS(DELTA-F)-ROFF)8163,8163,8164
 8163 L=1
 8164 IF(OC(II,J))163,160,165
  165 L=MIN(J)+L
      IF(L-MAX(J))167,160,160
  167 MAX(J)=L
      GO TO 166
  163 L=MAX(J)-L
      IF(L-MIN(J))160,160,164
  164 MIN(J)=L
  166 CHANGE=1.
  160 CONTINUE
      IF(II-1)270,150,270
  270 A(II,1)=DELTA
  150 CONTINUE
      NOCYC=NOCYC+1
      IF(CHANGE)232,233,232
C     LATTICE
  233 LSMALL=10**8
      IF(LRANGE-2)9165,9166,9165
 9166 LSMALL=-LSMALL
 9165 DO 170 J=2,N
      IF(LRANGE-1)9125,9126,9125
 9126  KJ=N+2-J
      GO TO 9127
 9125 KJ=J
 9127 LENGTH=MAX(KJ)-MIN(KJ)
      IF(LENGTH)9171,170,9171
 9171 IF(LRANGE-1)171,9128,171
 9128 K=KJ
      LSMALL=LENGTH
      GO TO 174
  171 IF(LRANGE)9176,9175,9176
 9175 LR=1
      GO TO 9177
```

```
 9176 LR=-1
 9177 IF(LR*(LENGTH-LSMALL))172,170,170
  172 LSMALL=LENGTH
      K=J
  170 CONTINUE
      IF(ABS(LSMALL)-10**8)174,173,174
C     HYPOTHESES
  174 L=INT(LSMALL/2)
      LOSTO=MAX(K)-L
      ISTO=MAX(K)
      MAX(K)=LOSTO-1
      IF(IFULL)803,800,803
  800 DO 801 J=2,N
      IF(J-K)802,801,802
  802 MSTO(IL+J,1)=MIN(J)
      MSTO(IL+J,2)=MAX(J)
  801 CONTINUE
      MSTO(IL+K,1)=LOSTO
      MSTO(IL+K,2)=ISTO
      NUMHYP=NUMHYP+1
      IL=IL+N
      IF(IL-LIMSTO)805,804,805
  805 IL=IL-1
      GO TO 232
  804 IFULL=1
      IL=IL-1
      GO TO 232
  803 PRINT 806,(SOLN(J),J=1,M-1),BEST,KOUNT,NUMITS
  806 FORMAT(//113H HYPOTHESES HAVE BEEN MADE BUT ALLOCATED STORAGE IS F
     1ULL.JOB IS DISCONTINUED WITH THE BEST SOLUTION TO DATE BEING//(12F
     110.4)//36H WHICH PRODUCED THE FUNCTIONAL VALUE//30X,6H XO = ,F10.4
     1//6H AFTER,I6,15H ITERATIONS AND,I6,20H CONTRACTION CYCLES.)
      GO TO 506
C     TEST LATTICE POINT
  173 DO 870 II=1,NUMKON
      IF(KON(II)-NINT)870,870,871
```

```
  870 CONTINUE
      GO TO 872
  871 DO 300 I=NINT+1,M
      IF(I-KON(II))581,333,581
  333 IF(II-NUMKON)666,300,300
  666 II=II+1
      GO TO 300
  581 SUM=OC(I,1)
      DO 305 J=2,N
      B=MIN(J)
  305 SUM=SUM-OC(I,J)*B
      IF(ABS(SUM)-ROFF)3001,3001,395
 3001 SUM=0
  395 IF(SUM)306,396,396
  396 A(I,1)=SUM
  300 CONTINUE
  872 NUMITS=NOCYC
      BEST=0
      DO 588 I=2,N
      J=I-1
      SOLN(J)=MAX(I)
  588 BEST=BEST-OC(1,I)*SOLN(J)
      DO 285 I=N+1,M
  285 SOLN(I-1)=A(I,1)
      A(1,1)=BEST
      IF(IVERS)9236,9234,9236
 9234 DO 9235 J=2,N
      MIN(J)=KEW(J)
 9235 MAX(J)=KEW(10+J)
      NUMHYP=0
      GO TO 234
 9236 FC=A(1,1)+GCD
      IF(FC-OPCON)306,500,500
  306 IF(NUMHYP)308,500,308
  308 IL=IL-N+1
      DO 810 J=2,N
```

```
      MIN(J)=MSTO(IL+J,1)
  810 MAX(J)=MSTO(IL+J,2)
      NUMHYP=NUMHYP-1
      IFULL=0
      GO TO 232
  500 PRINT 501,(SOLN(I),I=1,M-1)
  501 FORMAT(//27H INVESTIGATION IS COMPLETE./30H THE OPTIMUM SOLUTION P
     1OINT IS//6X,10F11.4/6X,10F11.4/6X,10F11.4)
      PRINT 508,BEST,KOUNT,NUMITS,NOCYC
  508 FORMAT(//44H WHICH PRODUCED THE OPTIMUM FUNCTIONAL VALUE//30X,6H X
     10 = ,F10.4//6H AFTER,I6,15H ITERATIONS AND,I6,20H CONTRACTION CYCL
     1ES.//52H THE TOTAL NUMBER OF CONTRACTION CYCLES REQUIRED WAS,I6)
  506 JOB=JOB+1
      IF(NOJOBS-JOB)502,502,503
  502 CALL EXIT
      END


      FUNCTION DIFF(X)
      FLO=INT(X)
      DIFF=X-FLO
      RETURN
      END
```

FORTRAN PROGRAM LISTING.

BRANCH AND BOUND CODE.

```
      DIMENSION A(63,20),SOLN(30),TAB(600,20),KEW(20)
      RA=34359738368.
      RB=68719476736.
      RC=137438953472.
      RE=1./RA
      READ 150,NOJOBS,BOUND
      JOB=0
100   READ 1,M,N,INTER,MAXIT,KOL,MZED,IS,NINT,LOC,JTAPE
    1 FORMAT(2I6,8X,I1,I6,2X,2I1,I2,7X,3I6)
150   FORMAT(I3,F9.9)
      READ 2,(A(1,J),J=1,N),((A(I,J),J=1,N),I=N+1,M)
    2 FORMAT(10F8.0)
      RD=13543288579.
      ROFF=.5-BOUND
      NN=N-2
      LOCOR=LOC
      ISTAGE=0
      MRM=M
      LIST=0
      BEST=-10.**20
      BDC=BEST
      NUMITS=0
      IFULL=0
      IL=0
      DO 291 I=2,N
      DO 291 J=1,N
      IF(I-J)293,292,293
293   A(I,J)=0
      GO TO 291
292   A(I,J)=-1.
291   CONTINUE
```

```
      ITAB=(M+2*NINT)*N
      IF(LOC-ITAB)96,96,97
   97 REWIND JTAPE
   96 IF(ITAB-512)102,101,101
  102 IF(LOC-512)104,103,103
  104 PRINT 105
  105 FORMAT(118H SOLUTION TO THIS PROBLEM HAS NOT COMMENCED BECAUSE THE
     1POSSIBLE NECESSITY FOR MAGNETIC TAPE STORAGE CANNOT BE CATERED/
     182H FOR DUE TO THE IMPOSSIBILITY OF CREATING THE MINIMUM TRANSFER
     1BLOCK OF 512 WORDS.)
      GO TO 106
  103 ITRANS=LOC/ITAB
      ITRANS=ITAB*ITRANS
  101 PRINT 3,IS,KOL
    3 FORMAT(1H1//69H SOLUTION TO INTEGER LP BY BRANCH AND BOUND METHOD
     1APPLIED TO PROBLEM,I4,16H USING VARIANT (,I1,2H).)
   25 SMALL=A(1,2)
      L=2
      DO 4 J=3,N
      IF(SMALL-A(1,J))4,4,5
    5 SMALL=A(1,J)
      L=J
    4 CONTINUE
      IF(SMALL)6,26,26
    6 RK=-BEST
      DO 8 I=2,M
      IF(A(I,L))8,8,9
    9 RR=A(I,1)/A(I,L)
      FLO=RK-RR
      IF(ABS(FLO)-ROFF)8,8,333
  333 IF(FLO)8,8,10
   10 RK=RR
      IR=I
    8 CONTINUE
      IF(RK+BEST)14,13,14
   13 PRINT 202
```

```
  202 FORMAT(//40H EXIT CALLED SINCE FUNCTIONAL UNBOUNDED.)
      GO TO 106
   14 PIVOT=A(IR,L)
   94 DO 15 I=1,MRM
      IF(I-IR)17,15,17
   17 F=-A(I,L)/PIVOT
      DO 16 J=1,N
      IF(J-L)18,19,18
   18 A(I,J)=A(I,J)+F*A(IR,J)
      IF(ABS(A(I,J))-ROFF)1500,1500,16
 1500 A(I,J)=0
      GO TO 16
   19 A(I,J)=F
   16 CONTINUE
   15 CONTINUE
      NUMITS=NUMITS+1
   20 DO 22 J=1,N
      IF(J-L)24,23,24
   23 A(IR,J)=-1.
      GO TO 22
   24 A(IR,J)=0
   22 CONTINUE
      IF(INTER)91,90,91
   91 PRINT 92
   92 FORMAT(//19H CURRENT TABLEAU IS)
   93 GO TO (81,82,83,84,85,86,87,88,589,590,591,592,593,599),NN
   81 PRINT 701,((A(I,J),J=1,N),I=1,M)
  701 FORMAT(20X,3F11.6)
      GO TO 90
   82 PRINT 702,((A(I,J),J=1,N),I=1,M)
  702 FORMAT(18X,4F11.6)
      GO TO 90
   83 PRINT 703,((A(I,J),J=1,N),I=1,M)
  703 FORMAT(16X,5F11.6)
      GO TO 90
   84 PRINT 704,((A(I,J),J=1,N),I=1,M)
```

```
704 FORMAT(14X,6F11.6)
    GO TO 90
 85 PRINT 705,((A(I,J),J=1,N),I=1,M)
705 FORMAT(12X,7F11.6)
    GO TO 90
 86 PRINT 706,((A(I,J),J=1,N),I=1,M)
706 FORMAT(10X,8F11.6)
    GO TO 90
 87 PRINT 707,((A(I,J),J=1,N),I=1,M)
707 FORMAT(8X,9F11.6)
    GO TO 90
 88 PRINT 708,((A(I,J),J=1,N),I=1,M)
708 FORMAT(6X,10F11.6)
    GO TO 90
589 PRINT 594,((A(I,J),J=1,N),I=1,M)
594 FORMAT(6X,10F11.6/105X,F11.6)
    GO TO 90
590 PRINT 595,((A(I,J),J=1,N),I=1,M)
595 FORMAT(6X,10F11.6/94X,2F11.6)
    GO TO 90
591 PRINT 596,((A(I,J),J=1,N),I=1,M)
596 FORMAT(6X,10F11.6/83X,3F11.6)
    GO TO 90
592 PRINT 597,((A(I,J),J=1,N),I=1,M)
597 FORMAT(6X,10F11.6/72X,4F11.6)
    GO TO 90
593 PRINT 598,((A(I,J),J=1,N),I=1,M)
598 FORMAT(6X,10F11.6/61X,5F11.6)
    GO TO 90
599 PRINT 510,((A(I,J),J=1,N),I=1,M)
510 FORMAT(1X,16F7.3)
 90 IF(NUMITS-MAXIT)194,195,195
194 IF(ISTAGE)26,25,26
195 IF(ISTAGE)95,196,95
196 PRINT 198
198 FORMAT(//92H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF AL
```

```
     1LOWABLE ITERATIONS HAS BEEN PERFORMED./44H THE INTEGER PHASE HAS N
     1OT YET BEEN ENTERED.)
      GO TO 106
  95  PRINT 98,(SOLN(I),I=1,M-1)
  98  FORMAT(//111H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF A
     1LLOWABLE ITERATIONS HAS BEEN PERFORMED.HOWEVER THE LIST OF/34H SUB
     1PROBLEMS IS NOT YET EXHAUSTED./38H THE BEST SOLUTION ATTAINED TO D
     1ATE IS//6X,8F14.6/6X,8F14.6/6X,8F14.6/6X,8F14.6)
      PRINT 77,BEST,KOUNT
  77  FORMAT(/36H WHICH PRODUCED THE FUNCTIONAL VALUE//30X,6H XO = ,F10.
     14//6H AFTER,I6,12H ITERATIONS.)
      GO TO 106
  26  DO 21 I=2,MRM
      IF(A(I,1))51,21,21
  21  CONTINUE
      IF(ISTAGE)28,27,28
  27  ISTAGE=1
      MRM=M+2*NINT
      DO 150 I=M+1,MRM
      DO 150 J=1,N
 150  A(I,J)=0
      PRINT 200,NUMITS,(A(I,1),I=2,M)
 200  FORMAT(/58H PRIMAL AND DUAL CONTINUOUS FEASIBILITY WAS ATTAINED AF
     1TER,I4,37H ITERATIONS IN THE SIMPLEX ALGORITHM./34H THE CONTINUOUS
     1 SOLUTION POINT WAS//6X,8F14.6/6X,8F14.6/6X,8F14.6/6X,8F14.6)
      PRINT 201,A(1,1)
 201  FORMAT(/31H PRODUCING THE FUNCTIONAL VALUE//30X,6H XO = ,F10.4//45
     1H PROGRAMME NOW PROCEEDS TO INTEGRALITY PHASE.)
      GO TO 29
  28  FLO=A(1,1)-BEST
      IF(ABS(FLO)-ROFF)50,50,303
 303  IF(FLO)50,50,29
  29  K=0
      DO 30 I=2,NINT
      INTA=INT(A(I,1))
      FLO=INTA
```

```
      FRAC=A(I,1)-FLO
      DEV=ABS(.5-FRAC)
      IF(DEV-BOUND)31,30,30
   31 JU=0
      JL=0
      DO 60 J=2,N
      IF(A(I,J))61,60,62
   61 IF(JU)64,63,64
   63 JU=1
      DU=-A(1,J)/A(I,J)
      GO TO 60
   64 F=-A(1,J)/A(I,J)
      FLO=DU-F
      IF(ABS(FLO)-ROFF)60,60,2110
 2110 IF(FLO)60,60,65
   65 DU=F
      GO TO 60
   62 IF(JL)67,66,67
   66 JL=1
      DD=A(1,J)/A(I,J)
      GO TO 60
   67 F=A(1,J)/A(I,J)
      FLO=DD-F
      IF(ABS(FLO)-ROFF)60,60,2111
 2111 IF(FLO)60,60,68
   68 DD=F
   60 CONTINUE
      IF(JL)70,69,70
   69 LDIRT=-1
 7400 K=I
      LDIR=LDIRT
      GO TO 34
   70 COMP=DD*FRAC
      LDIRT=1
      IF(JU)71,7400,71
   71 F=DU*(1.-FRAC)
```

```
      FLO=COMP-F
      IF(ABS(FLO)-ROFF)73,73,2112
2112  IF(FLO)72,73,73
   72 COMP=F
      LDIRT=2
      GO TO 74
   73 LDIRT=-2
   74 IF(K)75,76,75
   75 FLO=BIG-COMP
      IF(ABS(FLO)-ROFF)30,30,2113
2113  IF(FLO)76,30,30
   76 BIG=COMP
      K=I
      LDIR=LDIRT
   30 CONTINUE
      IF(K)34,54,34
   54 IF(BEST-BDC)32,57,32
   57 PRINT 107,(A(I,1),I=2,M)
  107 FORMAT(//27H THE FIRST INTEGER SOLUTION//6X,8F14.6/6X,8F14.6/6X,8F
     114.6/6X,8F14.6)
      PRINT 108,A(1,1),NUMITS
  108 FORMAT(/31H PRODUCING THE FUNCTIONAL VALUE//30X,6H XO = ,F10.4//19
     1H WAS ATTAINED AFTER,I6,12H ITERATIONS.)
   32 BEST=A(1,1)
      DO 33 I=2,M
      J=I-1
   33 SOLN(J)=A(I,1)
      KOUNT=NUMITS
      GO TO 50
   34 INTA=INT(A(K,1))
      FLO=INTA
      II=M+K
      IF(LDIR)35,35,36
   35 II=II+NINT
      IF(LDIR+2)38,37,38
   37 A(II,1)=FLO-A(K,1)
```

```
      DO 180 J=2,N
  180 A(II,J)=-A(K,J)
      ASSIGN 38 TO KK
  600 IF(IFULL)404,400,404
  400 LOC=LOC-ITAB
      IF(LOC)403,401,401
  401 DO 402 I=1,MRM
      IL=IL+1
      DO 402 J=1,N
  402 TAB(IL,J)=A(I,J)
      GO TO 138
  403 IFULL=1
      LOC=LOC+ITAB
  404 IF(ITAB-512)121,120,120
  121 IF(ITRANS-512)123,122,122
  122 WRITE TAPE JTAPE,((TAB(I,J),J=1,N),I=1,IL)
      GO TO 124
  123 JUNK=512-ITAB
      MM=JUNK/N
      MN=JUNK-MM*N
      WRITE TAPE JTAPE,((TAB(I,J),J=1,N),I=1,MRM),((TAB(I,J),J=1,N),I=1,
     1MM),(TAB(1,J),J=1,MN)
  124 IFULL=0
      IL=0
      LOC=LOCOR
      GO TO 400
  120 WRITE TAPE JTAPE,((A(I,J),J=1,N),I=1,MRM)
  138 LIST=LIST+1
      GO TO KK,(38,41)
   38 A(II,1)=A(K,1)-FLO*1.
      DO 39 J=2,N
   39 A(II,J)=A(K,J)
      GO TO 43
   36 IF(LDIR-2)41,40,41
   40 A(II,1)=A(K,1)-FLO*1.
      DO 181 J=2,N
```

```
181 A(II,J)=A(K,J)
    ASSIGN 41 TO KK
    GO TO 600
 41 A(II,1)=FLO-A(K,1)
    DO 42 J=2,N
 42 A(II,J)=-A(K,J)
 43 I=II
 51 IR=I
    NO=0
    KEW(2)=0
    DO 45 J=2,N
    IF(A(IR,J))46,45,45
 46 IF(NO)48,47,48
 47 NO=1
    RATIO=-A(1,J)/A(IR,J)
    L=J
    KEW(2)=0
    ITK=0
    GO TO 45
 48 F=-A(1,J)/A(IR,J)
    FLO=RATIO-F
    IF(ABS(FLO)-ROFF)945,945,2114
945 IF(KOL-2)45,946,45
946 ITK=ITK+1
    KEW(ITK+1)=J
    GO TO 45
2114 IF(FLO)45,45,49
 49 RATIO=F
    L=J
    KEW(2)=0
    ITK=0
 45 CONTINUE
    IF(KEW(2))948,947,948
947 IF(NO)14,50,14
948 KEW(1)=L
    ITK=ITK+1
```

```
      RD=5.*RD
      IF(RD-RC)949,949,950
  950 RD=RD-RC
  949 IF(RD-RB)951,951,952
  952 RD=RD-RB
  951 IF(RD-RA)953,953,954
  954 RD=RD-RA
  953 F1=RE*RD
      TOP=ITK
      ITK=1.+F1*(TOP-1.)+.5
      L=KEW(ITK)
      GO TO 14
   50 IF(LIST)52,80,52
   52 IF(IL)11,7,11
    7 BACKSPACE JTAPE
      IF(ITAB-512)109,110,110
  109 IL=ITRANS/N
      LOC=0
      READ TAPE JTAPE,((TAB(I,J),J=1,N),I=1,IL)
      BACKSPACE JTAPE
   11 IL=IL-MRM
      DO 12 I=1,MRM
      IK=IL+I
      DO 12 J=1,N
   12 A(I,J)=TAB(IK,J)
      LOC=LOC+ITAB
      IFULL=0
      GO TO 53
  110 READ TAPE JTAPE,((A(I,J),J=1,N),I=1,MRM)
      BACKSPACE JTAPE
   53 LIST=LIST-1
      GO TO 26
   80 IF(BEST-BDC)99,58,99
   58 PRINT 59,NUMITS
   59 FORMAT(//36H NO OPTIMUM SOLUTION HAS BEEN FOUND./I6,27H ITERATIONS
     1 WERE PERFORMED.)
```

```
      GO TO 106
   99 PRINT 55,(SOLN(I),I=1,M-1)
   55 FORMAT(//79H ALL NECESSARY SUBPROBLEMS HAVE BEEN EXAMINED SO THE I
     1NVESTIGATION IS COMPLETE./30H THE OPTIMUM SOLUTION POINT IS//6X,10
     1F11.4/6X,10F11.4/6X,10F11.4)
      PRINT 111,BEST,KOUNT,NUMITS
  111 FORMAT(/44H WHICH PRODUCED THE OPTIMUM FUNCTIONAL VALUE//30X,6H XO
     1 = ,F10.4//6H AFTER,I6,12H ITERATIONS./47H THE COMPLETE INVESTIGAT
     1ION REQUIRED A TOTAL OF,I6,12H ITERATIONS.)
  106 JOB=JOB+1
      IF(NOJOBS-JOB)44,44,100
   44 CALL EXIT
      END
```

FORTRAN PROGRAM LISTING.

MIXED INTEGER CODE.

```
      DIMENSION A(66,38),R(38),IA(38)
      RA=34359738368.
      RB=68719476736.
      RC=137438953472.
      RE=1./RA
      READ 99,NOJOBS,BOUND
   99 FORMAT(I3,F9.9)
   98 READ 1,M,N,KODE,MAX,METHOD,KOL,MZED,IS,NINT,(R(I),I=2,N)
    1 FORMAT(2I6,8X,I1,I6,1X,3I1,I2,7X,I6/80F1.0)
      READ 1001,(A(1,J),J=1,N),((A(I,J),J=1,N),I=N+1,M)
 1001 FORMAT(10F8.0)
      RD=13543288579.
      NC=NINT
      ROFF=.5-BOUND
      IF(NINT-M)702,700,702
  700 NC=N
      NINT=NINT+1
  702 NUM=0
      KIT=0
      KUTNO=0
      NN=N-2
      DO 1301 I=2,N
      DO 1301 J=1,N
 1301 A(I,J)=0
      DO 1302 I=2,N
 1302 A(I,I)=-1.
      IF(MZED)63,62,63
   62 I1=1
      GO TO 25
   63 I1=2
   25 PRINT 800,IS,METHOD,KOL
```

```
800 FORMAT(1H1//42H MIXED INTEGER ROUTINE SOLUTION TO PROBLEM,I3,47H U
   1SING THE VARIANTS DEFINED BY PARAMETER CODE (,I1,1H,,I1,2H).)
    IF(MZED)926,26,926
 26 PRINT 582
582 FORMAT(64H THE OBJECTIVE FUNCTION IS ADMITTED AS A POSSIBLE CUT GE
   1NERATOR.)
    GO TO 583
926 PRINT 584
584 FORMAT(56H THE OBJECTIVE FUNCTION IS EXCLUDED FROM CUT GENERATION.
   1)
583 NUMITS=0
 50 SMALL=A(1,2)
    L=2
    DO 2 J=3,N
    IF(SMALL-A(1,J))2,2,3
  3 SMALL=A(1,J)
    L=J
  2 CONTINUE
    IF(SMALL)4,5,5
  4 RK=10.**20
    NOTSIM=1
    DO 8 I=2,M
    IF(A(I,L))8,8,9
  9 RR=A(I,1)/A(I,L)
    FLO=RK-RR
    IF(ABS(FLO)-ROFF)8,8,333
333 IF(FLO)8,8,10
 10 RK=RR
    IR=I
  8 CONTINUE
    IF(RK-10.**20)14,13,14
 13 PRINT 202
202 FORMAT(//40H EXIT CALLED SINCE FUNCTIONAL UNBOUNDED.)
    GO TO 169
 14 DO 15 J=1,N
    IF(J-L)16,15,16
```

```
  16 DO 17 I=1,M
     IF(I-IR)1502,17,1502
1502 A(I,J)=A(I,J)-(A(IR,J)*A(I,L))/A(IR,L)
     IF(ABS(A(I,J))-ROFF)1500,1500,17
1500 A(I,J)=0
  17 CONTINUE
     A(IR,J)=0
  15 CONTINUE
     CON=-A(IR,L)
     DO 18 I=1,M
     A(I,L)=A(I,L)/CON
     IF(ABS(A(I,L))-ROFF)1501,1501,18
1501 A(I,L)=0
  18 CONTINUE
     NUMITS=NUMITS+1
     IF(MAX-NUMITS)989,989,990
 989 PRINT 991
 991 FORMAT(//114H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF I
    1TERATIONS HAVE BEEN PERFORMED.THE SITUATION AT THIS STAGE WAS)
     GO TO 987
 990 IF(KODE)6,52,6
   6 ASSIGN 52 TO KK
     PRINT 7
   7 FORMAT(//19H CURRENT TABLEAU IS)
     GO TO 88
  52 IF(IR-NINT)19,19,20
  19 R(L)=1.
     GO TO 21
  20 R(L)=0
  21 IF(NOTSIM)50,55,50
   5 NOTSIM=0
     JPRIM=1
  55 DO 60 I=2,M
     IF(A(I,1))61,60,60
  60 CONTINUE
     IF(JPRIM-1)32,31,32
```

```
 31 JPRIM=0
    PRINT 33,NUMITS
 33 FORMAT(/47H PRIMAL AND DUAL FEASIBILITY WAS ATTAINED AFTER,I4,37H
   1ITERATIONS OF THE SIMPLEX ALGORITHM./30H CONTINUOUS OPTIMUM TABLEA
   1U IS)
    ASSIGN 32 TO KK
    GO TO 88
 32 IR=0
 64 DO 65 I=I1,NC
    INTA=INT(A(I,1))
    FLO=INTA
    FRAC=A(I,1)-FLO
    DEV=ABS(FRAC-.5)
    IF(BOUND-DEV)65,66,66
 66 IF(METHOD-1)70,67,800
800 IF(METHOD-2)67,801,68
801 FRAC=1.-FRAC
 68 IF(IR)41,40,41
 40 IR=I
    SMALL=FRAC
    GO TO 65
 41 FLO=SMALL-FRAC
    IF(ABS(FLO)-ROFF)65,65,935
935 IF(FLO)65,65,40
 65 CONTINUE
    IF(IR)90,165,90
165 PRINT 100
100 FORMAT(25H JOB COMPLETE.TABLEAU IS //)
987 ASSIGN 69 TO KK
 88 GO TO (301,302,303,304,305,306,307,308,309,310,311,312,313,314),NN
301 PRINT 401,((A(I,J),J=1,N),I=1,M)
401 FORMAT(40X,3F13.2)
    GO TO KK,(52,32,69)
302 PRINT 402,((A(I,J),J=1,N),I=1,M)
402 FORMAT(34X,4F13.2)
    GO TO KK,(52,32,69)
```

```
303 PRINT 403,((A(I,J),J=1,N),I=1,M)
403 FORMAT(28X,5F13.2)
    GO TO KK,(52,32,69)
304 PRINT 404,((A(I,J),J=1,N),I=1,M)
404 FORMAT(22X,6F13.2)
    GO TO KK,(52,32,69)
305 PRINT 405,((A(I,J),J=1,N),I=1,M)
405 FORMAT(16X,7F13.2)
    GO TO KK,(52,32,69)
306 PRINT 406,((A(I,J),J=1,N),I=1,M)
406 FORMAT(1X,8F13.2)
    GO TO KK,(52,32,69)
307 PRINT 407,((A(I,J),J=1,N),I=1,M)
407 FORMAT(1X,8F13.2/105X,F13.2//)
    GO TO KK,(52,32,69)
308 PRINT 408,((A(I,J),J=1,N),I=1,M)
408 FORMAT(1X,8F13.2/92X,2F13.2//)
    GO TO KK,(52,32,69)
309 PRINT 409,((A(I,J),J=1,N),I=1,M)
409 FORMAT(1X,8F13.2/79X,3F13.2//)
    GO TO KK,(52,32,69)
310 PRINT 410,((A(I,J),J=1,N),I=1,M)
410 FORMAT(1X,8F13.2/66X,4F13.2//)
    GO TO KK,(52,32,69)
311 PRINT 411,((A(I,J),J=1,N),I=1,M)
411 FORMAT(1X,8F13.2/53X,5F13.2//)
    GO TO KK,(52,32,69)
312 PRINT 412,((A(I,J),J=1,N),I=1,M)
412 FORMAT(1X,8F13.2/40X,6F13.2//)
    GO TO KK,(52,32,69)
313 PRINT 413,((A(I,J),J=1,N),I=1,M)
413 FORMAT(1X,8F13.2/27X,7F13.2//)
    GO TO KK,(52,32,69)
314 PRINT 414,((A(I,J),J=1,N),I=1,M)
414 FORMAT(2(1X,8F15.6//)/)
    GO TO KK,(52,32,69)
```

```
70 SMALL=A(1,2)
   K=2
   DO 45 J=3,N
   IF(SMALL-A(1,J))45,45,46
46 K=J
   SMALL=A(1,J)
45 CONTINUE
97 SMALL=2.0
27 DO 56 I=I1,NC
   INTA=INT(A(I,K))
   FLO=INTA
   FRAC=A(I,K)-FLO
   DEV=ABS(.5-FRAC)
   IF(BOUND-DEV)91,59,59
91 DO 57 J=1,N
   IF(J-K)95,57,95
95 INTA=INT(A(I,J))
   FLO=INTA
   DEV=ABS(.5-A(I,J)+FLO)
   IF(BOUND-DEV)57,67,67
57 CONTINUE
   GO TO 56
59 IF(SMALL-FRAC)56,56,96
96 SMALL=FRAC
   IR=I
56 CONTINUE
   GO TO 90
67 IR=I
90 I=M+1
   INTA=INT(A(IR,1))
   FLO=INTA
   A(I,1)=FLO-A(IR,1)
   DO 80 J=2,N
   IF(R(J)-1.)81,82,81
81 IF(A(IR,J))83,84,84
83 A(I,J)=A(I,1)*(-A(IR,J))/(A(I,1)+1.)
```

```
      GO TO 80
   84 A(I,J)=-A(IR,J)
      GO TO 80
   82 INTA=INT(A(IR,J))
      FLO=INTA
      FRAC=A(IR,J)-FLO
      IF(ABS(FRAC)-ROFF)930,930,931
  930 FRAC=0
  931 IF(NINT-M-1)985,85,985
  985 IF(FRAC+A(I,1))85,85,86
   85 A(I,J)=-FRAC
      GO TO 80
   86 A(I,J)=A(I,1)*(1.-FRAC)/(1.+A(I,1))
   80 CONTINUE
      KUTNO=KUTNO+1
   61 IR=I
      DO 73 J=2,N
      IF(A(IR,J))74,73,73
   73 CONTINUE
      PRINT 78
   78 FORMAT(//66H EXIT CALLED BECAUSE CURRENT HYPERPLANE HAS POSITIVE C
     1OEFFICIENTS.)
      GO TO 169
   74 SMALL=-A(1,J)/A(IR,J)
      IA(2)=0
      ITK=0
      L=J
      IF(L-N)77,14,14
   77 JJ=L+1
      DO 35 I=JJ,N
      IF(A(IR,I))36,35,35
   36 FRAC=-A(1,I)/A(IR,I)
      FLO=SMALL-FRAC
      IF(ABS(FLO)-ROFF)350,350,359
  359 IF(FLO)35,350,37
  350 IF(KOL-1)35,360,360
```

```
360 ITK=ITK+1
    IA(ITK+1)=I
    GO TO 35
 37 SMALL=FRAC
    L=I
    ITK=0
    IA(2)=0
 35 CONTINUE
    IF(IA(2))351,14,351
351 IA(1)=L
    ITK=ITK+1
    IF(KOL-2)431,432,433
431 KIT=KIT+1
    L=KIT-ITK*(KIT/ITK)
    IF(L)355,356,355
355 L=IA(L)
    GO TO 14
433 DO 434 J=1,ITK
    DO 435 I=2,M
    IF(ABS(A(I,IA(J))+1.)-ROFF)436,436,435
436 DO 437 JJ=1,N
    IF(JJ-IA(J))438,437,438
438 IF(A(I,JJ))435,437,435
437 CONTINUE
    L=IA(J)
    GO TO 14
435 CONTINUE
434 CONTINUE
    KOL=2
432 RD=5.*RD
    IF(RD-RC)451,451,450
450 RD=RD-RC
451 IF(RD-RB)453,453,452
452 RD=RD-RB
453 IF(RD-RA)455,455,454
454 RD=RD-RA
```

```
455 F1=RE*RD
    TOP=ITK
    ITK=1.+F1*(TOP-1.)+.5
356 L=IA(ITK)
    GO TO 14
 69 PRINT 115,NUMITS,KUTNO
115 FORMAT(//50H THIS TABLEAU WAS ATTAINED AFTER A TOTAL NUMBER OF,I6,
   141H ITERATIONS AND THE ADDITION OF A FURTHER,I6,13H CONSTRAINTS.)
    NUM=NUM+1
    IF(NOJOBS-NUM)169,169,98
169 CALL EXIT
    END
```

FORTRAN PROGRAM LISTING.

ACCELERATED EUCLIDEAN CODE.

```
    DIMENSION MAT(22,10),KEW(9)
    RA=34359738368.
    RB=68719476736.
    RC=137438953472.
    RE=1./RA
    READ 300,NOJOBS
300 FORMAT(I3)
    NUM=0
303 READ 1,M,N,LD,OUT,KODE,MAXIT,NOROS,METHOD,KOL,MZED,IS,(MAT(1,J),J=
   11,N),((MAT(I,J),J=1,N),I=N+1,M)
  1 FORMAT(3I6,F2.0,I1,I6,4I1,I2/(10I8))
    RD=13543288579.
    NN=N-2
    KIT=0
    KUTNO=0
    KUTPA=-1
    DO 301 I=2,N
    DO 301 J=1,N
301 MAT(I,J)=0
    DO 302 I=2,N
302 MAT(I,I)=-1
    IF(MZED)461,460,461
460 I1=1
    GO TO 462
461 I1=2
462 IT=-1
    PRINT 80,IS,METHOD,KOL
 80 FORMAT(1H1//50H ACCELERATED EUCLIDEAN ROUTINE SOLUTION TO PROBLEM,
   1,I3,47H USING THE VARIANTS DEFINED BY PARAMETER CODE (,I1,1H,,I1,2
   1H).)
    IF(MZED)581,580,581
```

```
 580 PRINT 582
 582 FORMAT(64H THE OBJECTIVE FUNCTION IS ADMITTED AS A POSSIBLE CUT GE
    1NERATOR.)
     GO TO 583
 581 PRINT 584
 584 FORMAT(56H THE OBJECTIVE FUNCTION IS EXCLUDED FROM CUT GENERATION.
    1)
 583 IF(NOROS)310,309,310
 309 NINT=N
     GO TO 20
 310 NINT=M
  20 MIN=MAT(1,2)
     L=2
     DO 2 J=3,N
     IF(MIN-MAT(1,J))2,2,3
   3 MIN=MAT(1,J)
     L=J
   2 CONTINUE
     IF(MIN)4,75,75
   4 R=10.**20
     IT=IT+1
     IF(MAXIT-IT)988,988,111
 111 IF(OUT)901,902,901
 901 ASSIGN 982 TO KK
     PRINT 904
 904 FORMAT(//19H CURRENT TABLEAU IS)
     GO TO 903
 982 PRINT 983,LD
 983 FORMAT(16H WITH DIVISOR = ,I4,1H.)
 902 DO 8 I=2,M
     IF(MAT(I,L))8,8,9
   9 TOP=MAT(I,1)
     BOT=MAT(I,L)
     RR=TOP/BOT
     IF(R-RR)8,8,10
  10 R=RR
```

```
      IR=I
    8 CONTINUE
      IF(R-10.**20)14,13,14
   13 PRINT 202
  202 FORMAT(///97H EXIT CALLED SINCE A CLASS OF FEASIBLE SOLUTIONS WITH
     1 AN UNBOUNDED FUNCTIONAL CAN BE CONSTRUCTED.)
      GO TO 60
   14 DO 15 J=1,N
      IF(J-L)16,15,16
   16 LAMBDA=MAT(IR,J)
      DO 17 I=1,M
   17 MAT(I,J)=(MAT(I,J)*MAT(IR,L)-MAT(I,L)*LAMBDA)/LD
   15 CONTINUE
      LD=MAT(IR,L)
      DO 18 I=1,M
   18 MAT(I,L)=-MAT(I,L)
      GO TO 20
   75 IT=IT+1
      IF(MAXIT-IT)988,988,984
  984 IF(KODE)911,34,911
  988 PRINT 986
  986 FORMAT(/114H EXIT HAS BEEN CALLED BECAUSE THE MAXIMUM NUMBER OF IT
     1ERATIONS HAVE BEEN PERFORMED.THE SITUATION AT THIS STAGE WAS)
      GO TO 987
  911 ASSIGN 985 TO KK
      PRINT 904
      GO TO 903
  985 PRINT 983,LD
   34 DO 21 I=2,M
      IF(MAT(I,1))22,21,21
   21 CONTINUE
      IR=0
      DO 23 I=I1,N
      NLD=MAT(I,1)/LD
      IFRAC=MAT(I,1)-LD*NLD
      IF(IFRAC)124,23,124
```

```
124 IF(METHOD-1)24,125,126
125 IR=I
    GO TO 39
126 IF(METHOD-3)127,130,130
127 IF(IR)129,128,129
128 IR=I
    MAX=IFRAC
    GO TO 23
129 IF(MAX-IFRAC)128,23,23
130 IF(IR)132,131,132
131 IR=I
    MIN=IFRAC-1
    GO TO 23
132 IF(MIN-IFRAC+1)23,23,131
 23 CONTINUE
    IF(IR)39,50,39
 24 MIN=MAT(1,2)
    K=2
    DO 25 J=3,N
    IF(MIN-MAT(1,J))25,25,26
 26 MIN=MAT(1,J)
    K=J
 25 CONTINUE
    IFRST=10**10
    DO 27 I=I1,NINT
    IFRAC=MAT(I,K)-LD*(MAT(I,K)/LD)
    IF(IFRAC)32,28,30
 28 DO 29 J=1,N
    IF(K-J)65,29,65
 65 IF(MAT(I,J)-LD*(MAT(I,J)/LD))30,29,30
 29 CONTINUE
    GO TO 27
 32 IFRAC=IFRAC+LD
 30 IF(IFRST-IFRAC)27,27,31
 31 IFRST=IFRAC
    IR=I
```

```
      IF(IFRAC)27,39,27
   27 CONTINUE
   39 K=M+1
      KM=M+2
      DO 500 J=1,N
  500 MAT(K,J)=MAT(IR,J)
      JKN=0
      NJD=LD
      ITK=0
      KEW(2)=0
  510 DO 501 J=1,N
      MAT(KM,J)=MAT(K,J)/NJD
      IF(MAT(K,J))502,503,503
  502 MAT(KM,J)=MAT(KM,J)-1
  503 IF(JKN)504,504,501
  504 IF(J-2)515,505,505
  505 BOT=MAT(K,J)-NJD*MAT(KM,J)
      IF(BOT)514,501,514
  514 TOP=MAT(1,J)
      F1=TOP/BOT
      IF(SMALL-10.**9)507,506,507
  506 SMALL=F1
      L=J
      GO TO 501
  507 IF(SMALL-F1)501,550,506
  550 IF(KOL-1)501,551,551
  551 ITK=ITK+1
      KEW(ITK+1)=J
      GO TO 501
  515 SMALL=10.**9
  501 CONTINUE
      IF(JKN)552,562,552
  562 ASSIGN 552 TO MN
      GO TO 553
  552 JKN=1
      NJD=MAT(K,L)-NJD*MAT(KM,L)
```

```
      DO 508 J=1,N
      IF(J-L)509,508,509
  509 MAT(K,J)=MAT(K,L)*MAT(KM,J)-MAT(K,J)*MAT(KM,L)
  508 CONTINUE
      NLD=MAT(K,L)/NJD
      IFRAC=MAT(K,L)-NJD*NLD
      IF(IFRAC)570,511,570
  570 KUTPA=KUTPA+1
      GO TO 510
  511 DO 512 J=1,N
      IF(J-L)513,512,513
  513 MAT(K,J)=(LD*MAT(K,J)-NJD*MAT(IR,J))/MAT(K,L)
  512 CONTINUE
      MAT(K,L)=-(NJD*MAT(IR,L)/MAT(K,L))
      KUTNO=KUTNO+1
      IR=K
      GO TO 56
   22 IR=I
      DO 43 J=2,N
      IF(MAT(IR,J))44,43,43
   43 CONTINUE
      PRINT 53
   53 FORMAT(///78H EXIT CALLED BECAUSE HYPERPLANE UNDER CONSIDERATION H
     1AS POSITIVE COEFFICIENTS.)
      GO TO 60
   44 TOP=MAT(1,J)
      BOT=-MAT(IR,J)
      SMALL=TOP/BOT
      L=J
      KEW(2)=0
      ITK=0
      IF(L-N)57,56,56
   57 JJ=L+1
      DO 45 I=JJ,N
      IF(MAT(IR,I))46,45,45
   46 TOP=MAT(1,I)
```

```
      BOT=-MAT(IR,I)
      F1=TOP/BOT
      IF(SMALL-F1)45,95,47
   95 IF(KOL-1)45,360,360
  360 ITK=ITK+1
      KEW(ITK+1)=I
      GO TO 45
   47 SMALL=F1
      L=I
      ITK=0
      KEW(2)=0
   45 CONTINUE
      ASSIGN 56 TO MN
  553 IF(KEW(2))37,556,37
  556 GO TO MN,(552,56)
   37 KEW(1)=L
      ITK=ITK+1
      IF(KOL-2)431,432,433
  431 KIT=KIT+1
      L=KIT-ITK*(KIT/ITK)
      IF(L)61,62,61
   61 L=KEW(L)
      GO TO MN,(552,56)
  433 DO 434 J=1,ITK
      DO 435 I=2,M
      IF(MAT(I,KEW(J))+LD)435,436,435
  436 DO 437 JJ=1,N
      IF(JJ-KEW(J))438,437,438
  438 IF(MAT(I,JJ))435,437,435
  437 CONTINUE
      L=KEW(J)
      GO TO MN,(552,56)
  435 CONTINUE
  434 CONTINUE
      KOL=2
  432 RD=5.*RD
```

```
      IF(RD-RC)451,451,450
  450 RD=RD-RC
  451 IF(RD-RB)453,453,452
  452 RD=RD-RB
  453 IF(RD-RA)455,455,454
  454 RD=RD-RA
  455 F1=RE*RD
      TOP=ITK
      ITK=1.+F1*(TOP-1.)*.5
   62 L=KEW(ITK)
      GO TO MN,(552,56)
   56 DO 48 J=1,N
      IF(J-L)49,48,49
   49 LAMBDA=MAT(IR,J)
      DO 59 I=1,M
   59 MAT(I,J)=(MAT(I,L)*LAMBDA-MAT(I,J)*MAT(IR,L))/LD
   48 CONTINUE
      LDI=MAT(IR,L)
      GO TO 20
   50 PRINT 51
   51 FORMAT(///64H THE OPTIMAL INTEGER SOLUTION IS GIVEN IN THE FOLLOWI
     1NG TABLEAU.//)
  987 ASSIGN 200 TO KK
  903 GO TO (103,104,105,106,107,108,109,110),NN
  103 PRINT 52,((MAT(I,J),J=1,N),I=1,M)
   52 FORMAT(25X,3I8)
      GO TO KK,(982,985,200)
  104 PRINT 204,((MAT(I,J),J=1,N),I=1,M)
  254 FORMAT(18X,4I8)
      GO TO KK,(982,985,200)
  105 PRINT 205 ,((MAT(I,J),J=1,N),I=1,M)
  205 FORMAT(16X,5I8)
      GO TO KK,(982,985,255)
  106 PRINT 206,((MAT(I,J),J=1,N),I=1,M)
  256 FORMAT(14X,6I8)
      GO TO KK,(982,985,200)
```

```
107 PRINT 207,((MAT(I,J),J=1,N),I=1,M)
207 FORMAT(12X,7I8)
    GO TO KK,(982,985,200)
108 PRINT 208,((MAT(I,J),J=1,N),I=1,M)
208 FORMAT(10X,8I8)
    GO TO KK,(982,985,200)
109 PRINT 209,((MAT(I,J),J=1,N),I=1,M)
209 FORMAT(8X,9I8)
    GO TO KK,(982,985,200)
110 PRINT 210,((MAT(I,J),J=1,N),I=1,M)
210 FORMAT(6X,10I8)
    GO TO KK,(982,985,200)
200 PRINT 154,LD,IT,KUTPA,KUTNO
154 FORMAT(16H WITH DIVISOR = ,I4,1H,//25H WHICH WAS OBTAINED AFTER,I6
   1,41H ITERATIONS AND THE ADDITION OF A FURTHER,I6,8H PARTIAL/22H PI
   1VOT CONSTRAINTS AND,I6,25H TOTAL PIVOT CONSTRAINTS.)
    NUM=NUM+1
    IF(NOJOBS-NUM)60,60,303
 60 CALL EXIT
    END
```