UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE & TECHNOLOGY
DEPARTMENT OF MANAGEMENT SCIENCE

METHODS FOR INTEGER PROGRAMMING

by

MARY NICOLAS TRYPIA

A thesis submitted for
the Degree of Doctor of Philosophy
and the Diploma of Imperial College

March 1975

ABSTRACT

This thesis presents three new methods for the solution of the 0 - 1 linear programming problem. The method presented in Chapter 3 is the most promising and it is based on transforming the solution of an n-variable m-constraint problem into a sequence of solutions to k ($1 \leq k \leq m$) n-variable 2-constraint problems which are easier to solve.

The two other methods are tree search algorithms and mainly base their tests on the logical structure of the problem to be solved. The first one is a non-binary tree search while the other is a binary tree search.

A review of Integer Programming algorithms is also made.

ACKNOWLEGMENTS

The author of this thesis wishes to thank Dr. N. Christofides for his help and encouragement during this research, and also Professor S. Eilon Head of the Department of Management Science, Imperial College.

CONTENTS

Abstract 2

Acknowledgements 3

CHAPTER 1

Introduction

1.1. The Integer Programming problem 9

1.2. Some types of Integer Programming problems 11

1.3. Uses of 0 - 1 variables in modelling 17

1.4. Contribution of this thesis 23

CHAPTER 2

Review of Integer Programming Methods

2.1. Introduction 26

2.2. Cutting plane methods 28

2.2.1. Generating a cut 29

2.2.2. Gomory's fractional algorithm 31

2.2.3. Gomory's all-integer algorithm 32

2.2.4. Cuts with different values of h 33

2.2.5. Deep cuts 34

2.2.6. Intersection cuts 35

2.3. Enumerative methods 38

2.3.1. The branch & bound approach 38

2.4. Implicit enumeration 45

2.4.1. Balas' additive algorithm 48

2.4.2.	Glover's multiphase dual algorithm	50
2.4.3.	Balas' filter method	55
2.4.4.	Geoffrion's algorithm with a surrogate constraint	59
2.4.5.	The approach of Lemke & Spielberg	61
2.4.6.	The "pseudo-Boolean" approach of Hammer & Rudenau	63
2.4.7.	The Hammer & Nguyen approach	64
2.5.	Transformation of an Integer Programming problem in bounded variables to a knapsack problem in bounded variables	72

CHAPTER 3

A Sequential Approach to the 0 - 1 Linear Programming Problem

3.1.	Introduction	76
3.2.1.	The Method	80
3.2.2.	An iteration	82
3.2.3.	The solution of a problem $P_i(u)$	83
3.2.4.	The generation of the feasible solutions to the k constraints	87
3.3.	Description of the basic algorithm	93
3.4.	Improvements to the basic algorithm	95
3.5.1.	An example	98
3.5.2.	Problem $P_1(u)$	99
3.5.3.	Problem $P_2(u)$	104
3.5.4.	The iterations	106

CHAPTER 4

Computational aspects of the Sequential Approach of Chapter 3

4.1.	Introduction	112
4.2.	Storage requirements	112
4.3.	Choice of the k constraints	115
4.3.1.	Value of k	115
4.3.2.	Choice of k constraints	117
4.4.	Ways of limiting the size of the solution tree	120
4.4.1.	Ordering of the variables	121
4.4.2.	Use of LP and cutting planes	125
4.5.	Computational results	125

CHAPTER 5

Two Tree Search Algorithms using the Logical Structure of the Problem

5.1.	Introduction	129
5.2.	Reduced sets	129
5.2.1.	Definitions	129
5.2.2.	Calculation of the reduced sets	130
5.2.3.	An example	131
5.2.4.	Uses of reduced sets in the sequential approach of Chapter 3	131
5.3.	A NON-BINARY TREE SEARCH METHOD	133
5.3.1.	Introduction	133

5.3.2. The algorithm 136

5.3.3. An example 137

5.3.4. Improvements to the above algorithm 140

5.3.5. Computational aspects and results 141

5.4. A BINARY TREE SEARCH METHOD 143

5.4.1. Finding an initial feasible solution 143

5.4.2. Finding the optimal solution 146

5.4.3. The algorithm 147

5.4.4. An example 148

5.4.5. Computational results 150

CHAPTER 6

Conclusions 154

REFERENCES 156

APPENDIX I

The flow chart of the method presented in Chapter 3 166

APPENDIX II

The flow chart of the non-binary tree search method presented in Chapter 5 (section 5.3.) 168

APPENDIX III

The flow chart of the binary tree search method presented in Chapter 5 (section 5.4.) 170

C H A P T E R 1

INTRODUCTION

1.1. The Integer-Programming problem

In many real problems that can be formulated as Linear Programming (LP) problems the presence of non-integer values for some or all of the variables is physically meaningless. If, for example, variables represent people, machines, factories, e.t.c. these must obviously be in integers. In an LP problem the restriction of integrality for some variables leads to a new type of problem known as Integer Programming (IP). IP problems are in general much harder to solve than LP's and no known algorithm can solve efficiently general IP's with more than a few tens of variables as opposed to problems with tens of thousands of variables which can be solved quite routinely, as LP's.

If all the variables of the problem have to take integer values we have the case of All-Integer Programming; if only some of the variables need be integers, the others being permitted to take any non-negative continuous value we have the case of Mixed-Integer Programming. This thesis is concerned exclusively with All-Integer Programming problems and the general term Integer Programming is used to imply the all-integer case.

A typical IP problem has the form:

$$\text{(i)} \quad \min \; z = \sum_{j=1}^{n} c_j x_j' \qquad \left.\vphantom{\sum}\right)$$

$$\text{(ii)} \quad \text{subject to:} \; \sum_{j=1}^{n} a_{ij} x_j' \geq b_i \quad i = 1, \ldots, m \qquad \left.\vphantom{\sum}\right) \quad P'$$

(iii) $x'_j \geq 0$)

(iv) x'_j integer $j = 1, \ldots, n$)

The only difference of problem P' from an LP problem is the existence of constraints of type (iv).

A more general type of IP problem which is more often encountered than P' is the 0 - 1 linear programming problem:

$$\min \; z = \sum_{j=1}^{n} c_j x_j \qquad)$$

$$\text{subject to: } \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i = 1, \ldots, m \qquad) \quad P$$

$$x_j \in \{0, 1\} \quad j = 1, \ldots, n \qquad)$$

Obviously, it is possible to transform problem P' into P by replacing the variables x'_j of P' by $x'_j = \sum_{r=0}^{s} 2^r x_{jr}$ where s is chosen so that it does not limit the range of x'_j.

Many problems arise in practice which have the form of P directly. These are problems where "yes or no" decisions have to be made, and where one value of a binary variable represents a yes and the other a no.

The emphasis in this thesis is on the 0 - 1 linear programming problem P and three new methods are proposed for the solution of problem P.

1.2. Some types of IP's

It has already been mentioned that no good* algorithm is known for the solution of a general IP as defined by problem P'. Indeed it has been shown by Karp [49] that the IP problem is combinatorially equivalent to a whole class of other well known (and difficult) Operational Research problems, in the sense that if a good algorithm is found for one, that algorithm can be used to solve all other problems in the class. This class of problems contains most of the unsolved combinatorial Operational Research problems and therefore it is now believed (without proof) that no good algorithm for general IP's can exist [49]. However, some IP's with special structure can be solved quite easily and the best known of these types of problems will be briefly described.

(i) IP's whose LP solution is integer

There exist problems which are naturally integer in the sense that they can be solved by ordinary LP and still give integer values for the variables even if the integrality requirements on the n

---

Here we use "good" in the sense initially used by Edmonds [77]. An algorithm which solves an n-variable problem in $O(n^k)$ computations for any (constant) value of k is "good". If the number of computations is more than polynomially dependent on n then the algorithm is "not good".

variables are ignored. This is the case for problems having a totally unimodular matrix $A = [a_{ij}]$ and the data for $b = [b_i]$ is integer. (A square, integer matrix T is called unimodular if $|\det T| = 1$. An integer matrix A is totally unimodular if every square, nonsingular submatrix of A is unimodular). A typical example of this class is the transportation problem.

However, the LP solution to P' can be integer even if the matrix A is not totally unimodular. E.g. the matrix:

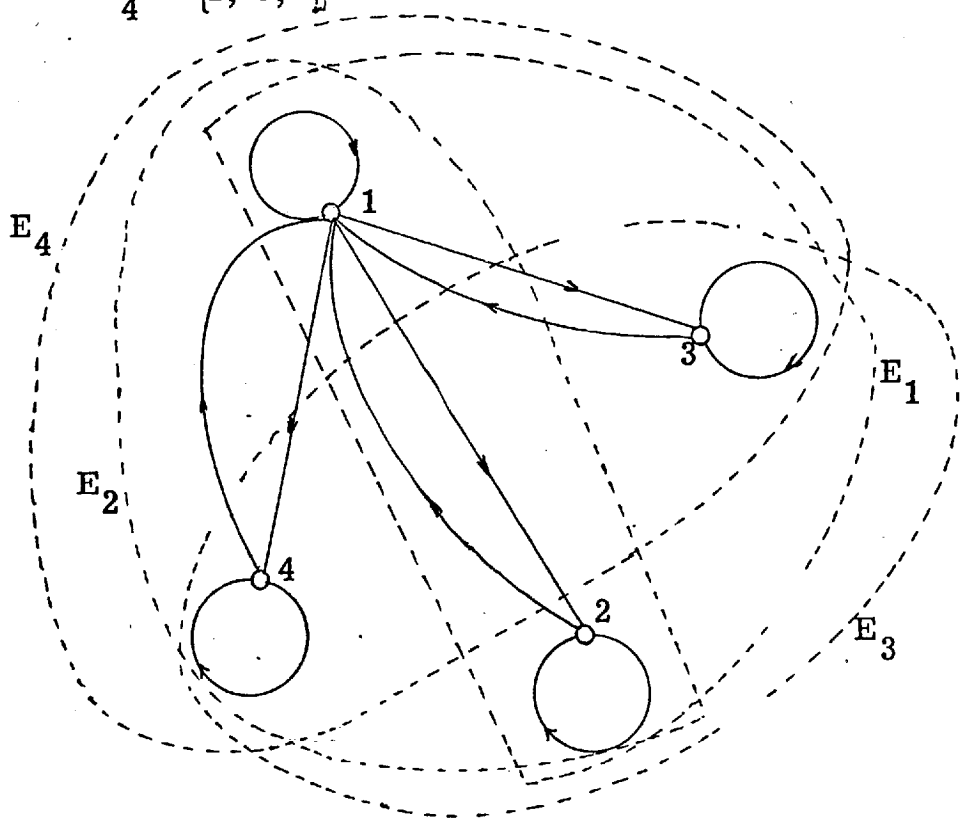
$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

is not totally unimodular but the LP solution to P' is integer if b is a non-negative integer vector. This is true for a 0 - 1 matrix A that is balanced [48] .

Balanced Matrices: A hypergraph $H = (X, \mathcal{E})$ consists of a finite set X of n elements called the vertices, together with a family $\mathcal{E} = (E_i / i \in I)$ of m non-empty subsets of X called the edges; if in addition we assume $\bigcup_{i \in I} E_i = X$, this permits us to define the hypergraph H by $(E_i / i \in I)$.

A hypergraph H is said to be balanced if every odd cycle $(x_1, E_1, x_2, E_2, \ldots, E_{2p+1}, x_1)$ has an edge E_i which contains at least three vertices x_j. A 0 - 1 matrix is said to be balanced if it is the incidence matrix of a balanced hypergraph.

Thus, when the matrix A is balanced and b is a non-negative integer vector, then the LP solution to P' is integer. For example, consider the hypergraph defined by: $X = \{x_1, x_2, x_3, x_4\}$ and $\mathcal{E} = \{E_1 = \{1, 2, 3\}, E_2 = \{1, 2, 4\}, E_3 = \{2, 3, 4\}, E_4 = \{1, 3, 4\}\}$



It can be observed that every odd cycle of this hypergraph (e.g. the odd cycle $x_1, E_1, x_2, E_2, x_4, E_4, x_1$) contains an edge E_i consisting of three vertices. Thus, its incidence matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

is balanced as the hypergraph is balanced.

However, the matrix A in general is neither totally unimodular nor balanced. Nevertheless, for certain other 0 - 1 matrices the property that all basic feasible solutions are integer remains true. This is the case for perfect matrices [50]. Again, this result stems from Graph theory.

Perfect matrices. Let G denote a finite undirected graph without self-loops and multiple parallel edges. Let a (G) denote the maximum cardinality of an independent* vertex set of G and $\Theta(G)$ denote the minimal number of cliques** which cover G. The graph G is called perfect if a (G') = $\Theta(G')$ for every vertex-induced subgraph G' of G. The matrix A is perfect if and only if it is the clique matrix of a perfect graph. For example, consider the graph of Figure 1:

---

* An independent vertex set of G is a set whose no two vertices are connected by an edge.

** A clique is defined as that set of vertices every two of which are connected by an edge.

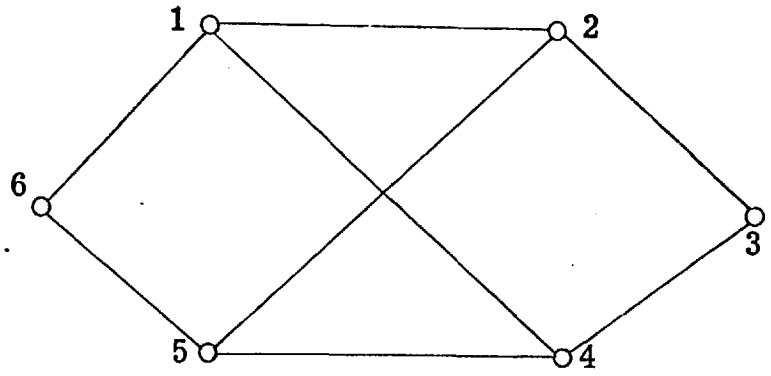


Figure 1

It can be observed that this graph is perfect and thus its clique matrix A

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & & & & & 1 \\ 1 & 1 & & & & \\ 1 & & & 1 & & \\ & 1 & & & 1 & \\ & 1 & 1 & & & \\ & & 1 & 1 & & \\ & & & 1 & 1 & \\ & & & & 1 & 1 \end{bmatrix} \end{array}$$

is perfect.

Clique matrices from graphs like the "triangulated graphs", the "rigid circuit graphs" are perfect.

(ii) "Matching problems"

A class of IP problems whose LP solution is not necessarily integer but which, nevertheless, can be solved exactly using a polynomially bounded algorithm is that of matching problems [41]. The IP problem P' is called a matching problem whenever $\sum_{i=1}^{m} |a_{ij}| \leq 2$ holds for all $j = 1, \ldots, n$ and all a_{ij}, b_i are integers.

The class of matching problems includes the transportation type problems but, in addition, includes problems for which omitting the integer restriction (iv) results in an LP whose optimum solution is not necessarily all integer.

However, when b is a unit vector and A is the incidence matrix of a graph G, all vertices of the associated convex polyhedron have the values (coordinates) either 1 or $\frac{1}{2}$ or 0. (When the graph is bipartite, its incidence matrix is totally unimodular and hence there are no vertices of value $\frac{1}{2}$.)

The matching problem is of interest as it can occur as a sub-problem in problems appearing in practice e.g. in the graph traversal problem [70], set covering problem [51], travelling salesman problem [71].

(iii) Set-covering problems

A very well known IP problem having a special structure is the set covering problem [51], [63], [64], [65], [66], which has the following form:

$$\begin{aligned} \min\ z &= \sum_{j=1}^{n} c_j x_j && && \\ \text{s.t.}\quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1 && i = 1, \ldots, m && \text{(P'')} \\ & a_{ij}, x_j \in \{0, 1\} && j = 1, \ldots, n && \end{aligned}$$

Problem P'' has a natural tendency towards integrality as very often its LP solution is all-integer.

The set covering problem has many applications e.g. airline crew scheduling [58] [59] [13] , information retrieval [52] disconnecting paths in a graph [53] , truck deliveries [54] political districting [55] [56] colouring problems [57] , designing optimal switching in circuits [60] , [61] .

1.3. Uses of 0 - 1 variables in modelling

Binary variables can be used to model a variety of complex situations such as non-linearities and non-convexities. A few examples of this usage are the following:

The fixed charge problem [62], [67], [68], [69].

In this type of problem a binary variable plays the role of a "valve" for releasing (or not releasing) a fixed cost. Here, a variable that involves a fixed charge d_j is expressed as $x_j + d_j y_j$ where $y_j \in \{0, 1\}$ and $x_j \geq 0$. If $x_j = 0 \longrightarrow y_j = 0$ and if $x_j > 0 \longrightarrow y_j = 1$. This is ensured by introducing the constraint: $x_j \leq M y_j$ where M is an upper bound on x_j. For example if the production on a machine j has two cost components, a setup cost

d_j and a variable cost per unit c_j, the total cost of production on machine j is $c_j x_j + d_j y_j$, and x_j is related to y_j by: $x_j \leq M y_j$ i.e. the total cost is 0 if $x_j = 0$ and $d_j + c_j x_j$ if $x_j > 0$.

A particular problem of this type is the fixed cost transportation problem [62] which is a transportation problem in which a fixed charge arises for every route used.

In addition, the plant location problem [67], [68], [69] is of this type. Suppose that there exist m possible locations, with location i admitting a plant of capacity a_i and requiring a fixed investment d_i and there are n customers the jth one demanding b_j units of a particular commodity. Let the cost per unit of supplying customer j from plant i be c_{ij}. Then the problem is:

$$\min \sum_{i=1}^{m} \left(\sum_{j=1}^{n} c_{ij} x_{ij} + d_i y_i \right)$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = b_j \qquad j = 1, \ldots, n$$

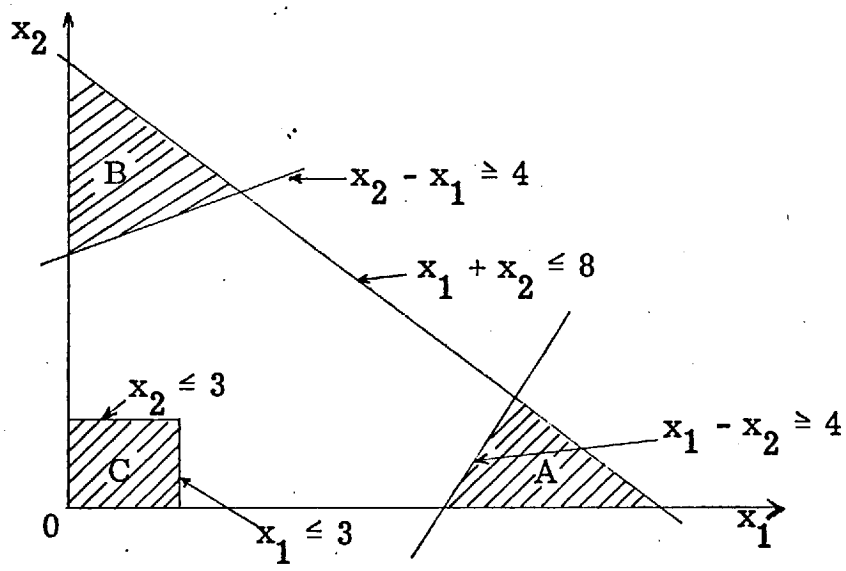
$$\sum_{j=1}^{n} x_{ij} \leq a_i y_i \qquad i = 1, \ldots, m$$

$$x_{ij} \geq 0 \quad y_i \in \{0, 1\} \quad \text{all i and j}$$

Non-convex feasible region

When the feasible region of a continuous problem is not convex binary variables are used to represent the non-convex set as a union of convex sets to indicate which convex set is currently active [11].

For example, if the feasible region has the shaded form:



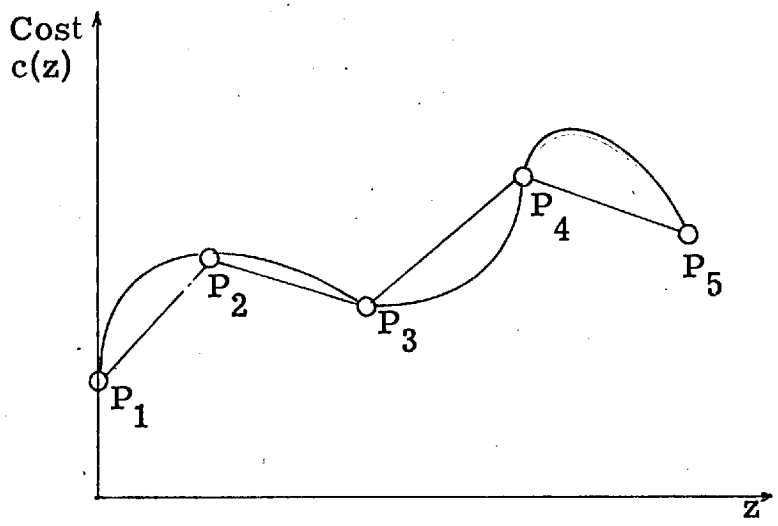
the three disjoint convex sets can be expressed as follows:

$$
\begin{aligned}
x_1 + x_2 &\leq 8 \\
x_1 + 7y_1 &\leq 10 \\
x_2 + 7y_1 &\leq 10 \\
-x_1 + x_2 - 19y_2 &\geq -15 \\
x_1 - x_2 - 19y_3 &\geq -15 \\
y_1 + y_2 + y_3 &= 1 \\
x_1,\ x_2 &\geq 0 \\
y_1,\ y_2,\ y_3 &\in \{0, 1\}
\end{aligned}
$$

When $y_1 = 1$ the solution is in region C, when $y_2 = 1$ the solution is in region B and when $y_3 = 1$ the solution is in region A.

Piecewise linear approximation of nonlinear functions

Consider the problem of approximating a non-linear function as shown [72] :



We have to know in which portion of the function we are and for this reason 4 binary variables y_1, y_2, y_3, y_4 are introduced with the constraints:

$$y_1 + y_2 + y_3 + y_4 = 1$$

$$\lambda_1 \leq y_1$$

$$\lambda_k \leq y_{k-1} + y_k \qquad k = 2, \ldots, 4$$

$$\lambda_5 \leq y_4$$

$$\sum_{k=1}^{5} \lambda_k = 1$$

$$y_1, y_2, y_3, y_4 \in \{0, 1\} \ , \ \lambda_k \geq 0 \qquad k = 1, \ldots, 5$$

where λ_k represent the weights attached to the points P_k and no

more than two λ_k (of the form λ_k, λ_{k+1}) are positive.

The above constraints ensure that whichever y_k is equal to 1 the λ_k's associated to P_k's which lie outside the range of the section corresponding to $y_k = 1$, must be zero. For example, if $y_2 = 1$, $y_1 = y_3 = y_4 = 0$ and $\lambda_2 \leq 1$, $\lambda_3 \leq 1$ all other $\lambda_k = 0$.

Capital budgeting [78]

Consider a firm which is faced with the problem of selecting some projects out of a total of n since it cannot undertake all n projects due to budget limitations. Say that c_j is the present value of project j and a_{ij} is the amount of investment required by project j in time period i, i = 1, ..., m, and b_i is the capital available in time period i. Then, the problem of maximizing the total present value subject to the budget constraints can be written as:

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m$$

$$x_j \in \{0, 1\} \qquad j = 1, \ldots, n$$

where $x_j = 1$ if project j is selected, $x_j = 0$ otherwise.

Problems of this type have a special structure since c_j, a_{ij}, $b_i \geq 0$ and this can be exploited accordingly for their solution.

Scheduling jobs on machines [73], [74], [75], [76].

Consider the problem of scheduling n jobs, i = 1, ..., n on m machines, k = 1, ..., m. Let each job require exactly one

operation on each machine and assume that the operations on each job must be done in a specified order. Let r_{ijk} be defined as follows:

$r_{ijk} = 1$ if the jth operation of job i requires machine k

$r_{ijk} = 0$ otherwise

Let the processing time of job i on machine k be t_{ik} and let x_{ik} denote the starting time of job i on machine k. To ensure that no two jobs are processed on the same machine at the same time, the following constraints are introduced:

$$x_{rk} - x_{sk} \geq t_{sk} - My_{rsk}$$

$$x_{sk} - x_{rk} \geq t_{rk} - (1 - y_{rsk})\, M$$

$$y_{rsk} \in \{0, 1\}$$

for all pairs of jobs r and s and all machines k and where M is arbitrarily large; $y_{rsk} = 1$ if job r precedes job s on machine k and $y_{rsk} = 0$ otherwise.

Another set of constraints is required for each job i to ensure that its operations are done in the specified order:

$$\sum_{k=1}^{m} r_{ijk}(x_{ik} + t_{ik}) \leq \sum_{k=1}^{m} r_{i,\, j+1,\, k} x_{ik} \qquad j = 1, \ldots, m - 1$$

where $\sum_{k=1}^{m} r_{ijk} x_{ik}$ denotes the starting time of the jth operation of job i.

There are many possible objective functions that can be used

for the above problem. A common one is to minimize the sum of the starting time of the last operation of each job:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{m} r_{imk} x_{ik}$$

1.4. Contribution of this thesis

The subject of this thesis is the 0 - 1 linear programming problem and three new methods are presented for the solution of this problem.

The method of Chapter 3 is the most significant of the three and it is a new approach to Integer Programming. It is a departure from the "traditional" approaches to this problem as it is neither in the vein of cutting plane algorithms nor in the vein of branch and bound methods. Nevertheless, concepts arising from the cutting plane approach can be easily incorporated in this method and further reduce the computing times. The computational aspects of the method are presented in Chapter 4 and its performance is demonstrated to be exceptionally good especially for dense problems having "small" coefficients. Its effectiveness in solving randomly generated problems (unstructured) of 100 variables is noteworthy.

The two other methods presented in Chapter 5 are tree-search algorithms with their search method based on the logical structure of the problem to be solved. The main deviation

of these methods from other related schemes is in the branching process and in the way that a partial solution is fathomed.

The computational results of these methods are not as good as those of the Chapter 3 method but nevertheless they can be promising especially if stronger bounds are introduced.

CHAPTER 2

REVIEW OF INTEGER PROGRAMMING METHODS

2.1. Introduction

Unlike Linear Programming, Integer Programming has no general method comparable to the Simplex to offer. Unfortunately, so far, no IP method has been capable of solving problems of an arbitrary nature of modest size in a reasonable amount of time, even if certain methods have been very successful in solving large problems of a special structure [41]. A basic feature of the IP methods is that they do not behave uniformly over all classes of problems, i.e. a method can be better for a certain class of problems than it is for others.

IP methods fall into two main categories:

1. Cutting plane methods
2. Enumerative methods

The approach of the first category was historically the first that was proposed for solving general IP problems and the foundations were laid by Gomory in 1958 [18] .

The second category of enumerative methods is represented by a very large number of algorithms. These algorithms come under the headings of "branch and bound", "tree search", and "implicit enumeration", with implicit enumeration being the name often given to a class of "branch and bound" algorithms designed specifically for the case where the variables are binary.

Other types of methods are the "Group theoretic approach" and the "knapsack type approach". The former solves an LP problem over its corner polyhedron while the latter transforms the IP problem of more than one constraints into a knapsack problem of one constraint.

It is widely felt that a combination of the cutting plane approach with the branch and bound approach might lead to better algorithms for Integer Programming.

The methods to be reviewed here are some of the "corner-stones" of Integer Programming as these have laid the foundations of further developments in the field, e.g. [18] , [16] , [1] , [3]

Also, a description is made, of some of the tools to be used in the new methods presented in Chapters 3, 4, 5. The emphasis is on implicit enumeration algorithms as these are closer to our subject matter, the 0 - 1 linear programming problem. Here, some preliminary tree-search concepts, used later in the methods of Chapter 5, are explained.

Apart from the cutting-plane, branch and bound, implicit enumeration algorithms, a review is also made of the knapsack transformation of an IP problem, since this is loosely related to the method given later on in Chapter 3.

2.2. CUTTING PLANE METHODS

These methods use LP iteratively and at each iteration derived constraint(s) are added which "cut off" part of the convex set so that the current solution is eliminated but no integer solution is excluded. These derived constraints are called cuts. The objective, here, is to reduce the convex set by adding cuts without eliminating any point of the convex integer hull so that in due course an optimal extreme point of the reduced convex set coincides with an optimal extreme point of the integer convex hull. Understandably, the crux of the problem is to generate efficient cuts that reduce the convex set appreciably each time a cut is added, so that in a finite, hopefully small, number of iterations an optimal integer solution is found. An even better approach on these lines would be the "efficient" generation of facets of the integer convex hull in the region of the LP optimum. These facets cannot unfortunately be generated efficiently but they would result in the strongest possible cutting planes.

Usually, these methods are dual in the sense that the first feasible integer solution found is also optimal. This can be a disadvantage in practice as one might wish only a feasible solution, not being prepared to pay for the cost of finding the optimal solution. Computational experience with cutting plane

algorithms has been confined to small problems [17] and the behaviour of the early algorithms has been somewhat erratic. An exception has been the work of Martin [12] [13] who has had outstanding success in solving large problems of a special structure, which have natural tendency towards integrality i.e. their LP optimums have a large part of their basic variables at an integer value. (Problems related to networks, the traveling salesman problem, set covering problems are of this category). Martin's experience has been on crew scheduling problems and he reports outstanding computational times [13] .

We now come to the description of how to derive the "fundamental cut".

2.2.1. Generating a cut

Consider the IP problem:

$$\left.\begin{array}{l} \max\ c\,x \\ \text{s.t.}\quad Ax = b \\ x \geq 0,\ \text{integer} \end{array}\right\} \quad F$$

where c is an n-component row vector, b is an m-component column vector and A is an $m \times n$ matrix.

Suppose that problem F, without any integer constraints has the LP representation:

$$x_{B_i} = y_{io} - \sum_{j \in R} y_{ij} x_j \qquad i = 0, 1, \ldots, m \qquad (1)$$

where R is the set of the non-basic variables so that the basic solution determined by (1) is $x_{B_i} = y_{io}$ $i = 0, 1, \ldots, m$ and $x_j = 0$ $j \in R$. Multiplying (1) by $h \neq 0$ yields:

$$hx_{B_i} + \sum_{j \in R} hy_{ij} x_j = hy_{io} \qquad (2)$$

Let $[a]$ denote the integer part of a and f its fractional part so that $a = [a] + f$, where $0 \leqslant f < 1$. The requirement $x \geq 0$ implies that

$$[h]\, x_{B_i} + \sum_{j \in R} [hy_{ij}]\, x_j \leq hy_{io} \qquad (3)$$

Moreover, since x is required to be integer the left hand side of (3) must be an integer and it should not exceed the integer part of the right hand side of (3). This implies that:

$$[h]\, x_{B_i} + \sum_{j \in R} [hy_{ij}]\, x_j \leq [hy_{io}] \qquad (4)$$

Eliminating x_{B_i} between (1) and (4) we have:

$$\sum_{j \in R} ([h] y_{ij} - [hy_{ij}])\, x_j \geq [h]\, y_{io} - [hy_{io}] \qquad (5)$$

Relation (5) is the fundamental cut [15] and by using it in various ways a number of cuts and cutting plane algorithms can be developed.

In the sections that follow some of the most characteristic cutting plane approaches are described.

2.2.2. Gomory's fractional algorithm [18]

When $h = 1$ in equation (5) we have the cut of the fractional algorithm. Thus, letting $h = 1$, (5) becomes:

$$\sum_{j \in R} (y_{ij} - \lceil y_{ij} \rceil) x_j \geq y_{io} - \lceil y_{io} \rceil \qquad (6)$$

and introducing $y_{ij} = \lceil y_{ij} \rceil + f_{ij}$ we obtain:

$$\sum_{j \in R} f_{ij} x_j \geq f_{io} \qquad (7)$$

or

$$s = -f_{io} + \sum_{j \in R} f_{ij} x_j, \quad s \geq 0 \qquad (8)$$

Moreover, s must be integer since from equation (1) we can write:

$$x_{B_i} = -(-f_{io} + \sum_{j \in R} f_{ij} x_j) + (\lceil y_{io} \rceil - \sum_{j \in R} \lceil y_{ij} \rceil x_j),$$

and the quantity $(\lceil y_{io} \rceil - \sum_{j \in R} \lceil y_{ij} \rceil x_j)$ is integer. If $f_{io} > 0$, (8) is violated by the basic solution determined from (1) and so the relation (8) can be used to exclude the current basic solution. Also, (8) does not exclude any feasible integer solution since it is implied by the necessity of integrality. Thus, the basic algorithm can be stated as follows:

Step 1 Solve problem F by LP (using the dual simplex technique)
Go to step 2.

Step 2 If the solution is integer, it is optimal. If not, go to step 3.

Step 3 Choose a row r with $f_{ro} > 0$ and add to the bottom of the LP

tableau the constraint (8) with i = r. Reoptimize using the dual simplex technique. Go to step 2.

A basic weakness of the above algorithm is that after a few initial iterations massive degeneracy can occur. This phenomenon frequently arises in medium and large-scale problems and sometimes even small-scale problems have require thousands of iterations to achieve convergence [17] .

Another weakness of the method is that it is subject to round-off errors, while it is essential to recognise integers. Failing to recognise an integer could cause unnecessary iterations, invalid cuts and even loss of an optimal solution. Conversely, the improper identification of an integer could cause false termination.

2.2.3. Gomory's all-integer algorithm [15]

This method assumes that all initial coefficients are integers and keeps all the coefficients integer throughout the pivoting operations. The starting point of the method is a dual feasible solution. A cut is generated in such a way so to keep the pivot element equal to -1 and ensure the integrality of the tableau. By using (5) as a cut there exists an h, $0 < h < 1$ so that the current basic solution is eliminated and the pivot element is -1. Selecting any primal infeasible row, say the rth, and rewriting (5) as an equality with $0 < h < 1$ we obtain:

$$[hy_{ro}] = \sum_{j \in R} [hy_{rj}]\, x_j + s,\ s \geq 0,\ \text{integer} \qquad (9)$$

It can be shown [11] that, in order

1. to retain an integral tableau
2. to retain dual feasibility
3. to have the largest decrease in the objective function, the value of h should be:

$$h \leq \min_{j \in R_r} \frac{M_j}{y_{rj}} = h^* < 1$$

where $R_r = \{ j / y_{rj} < 0,\ j \in R \}$, k is the pivot column, $y_{ok} = \min_{j \in R_r} y_{oj}$, and $M_j = -\left[\frac{y_{oj}}{y_{ok}} \right]$. If $h^* \not< 1$ any $h < 1$ for which $h \leq \min_{j \in R_r} \frac{M_j}{y_{rj}}$ suffices to satisfy the conditions 1 and 2 above.

In the all-integer algorithm only one cut can be added at a time while in the fractional algorithm there exists the possibility of adding many cuts simultaneously.

2.2.4. Cuts with different values of h [18], [19]

For any integer h, the fundamental cut (5) can be written as:

$$\sum_{j \in R} (hf_{ij} - [hf_{ij}])\, x_j \geq hf_{io} - [hf_{io}] \qquad (10)$$

It can be noted that when h = 1, $[hf_{ij}] = 0$ and (10) specializes to (7). Sometimes, when h > 1 (10) can yield stronger cuts than (7). We could optimize the value of h after specifying a certain criterion,

for example we could maximize $(hf_{io} - [hf_{io}])$ and thus use the best integer value of h. Also, additional cuts of the form (5) can be generated from integer linear combinations of source rows.

Finally, it may be possible to obtain stronger cuts when the value of h is non-integer [19].

2.2.5. Deep cuts [20]

Consider the Gomory fractional cut given by $\sum_{j \in R} f_{ij} x_j \geq f_{io}$. If this cut is not satisfied as an equality by any $x \in S$, where S is the set of feasible integer solutions, then the Gomory cut can be made deeper in the sense that there exists a $t > f_{io}$ such that the cut:

$$\sum_{j \in R} f_{ij} x_j \geq t$$

is still valid, i.e. it does not exclude any feasible integer solution. The deepest possible cut, then, corresponds to the smallest value of t for which the equality $\sum_{j \in R} f_{ij} x_j = t$ is satisfied by some $x \in S$. Thus,

$$t = \min \sum_{j \in R} f_{ij} x_j \quad x \in S \qquad (11)$$

However, to solve problem (11) is difficult, so a relaxation on the solution space can be introduced to make the proposal practical.

2.2.6. Intersection cuts [21] , [22] , [23]

One can develop cuts which can exclude a current solution but no integer solution and are based on geometric properties.

Let S represent the set of feasible integer solutions and T represent the set:

$$T = \left\{ x / \; x = y_o - \sum_{j \in R} y_j x_j, \; x_j \geq 0 \;\; j \in R \right\} \qquad (12)$$

where $y_j = (y_{1j}, \ldots, y_{nj})$ and $x = (x_1, \ldots, x_n)$.

T does not require the basic variables to be either non-negative or integer, so that $S \subset T$. Geometrically, T is an n-dimensional cone with apex at $x = y_o$ and edges given by the extreme rays

$$r_j = y_o - \theta_j y_j, \;\; \theta_j \geq 0, \; j \in R \qquad (13)$$

These n extreme rays are linearly independent and therefore have a unique intersection point at y_o. Now, if we consider a closed and bounded convex set C in n-space containing y_o in its interior ($y_o \in C - C_b$, where C_b is the boundary of C) it can be shown that there exists a unique θ_j^o, $\theta_j^o > 0$ for each $j \in R$, such that:

$$y_o - \theta_j^o y_j = \; p_j^o \in C_b$$

The points p_j^o (see Fig. 1) where the rays of the cone meet the boundary of the convex set, determine the hyperplane: $d^o x = d_o^o$

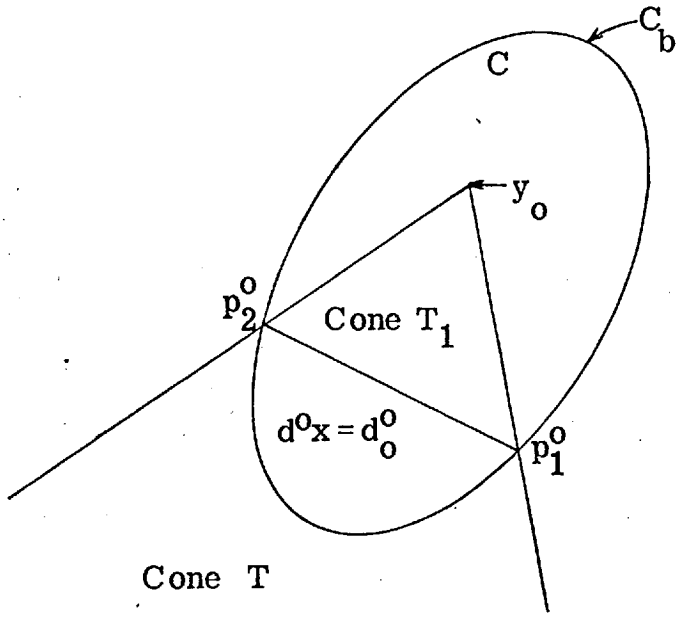


Fig. 1

This hyperplane does not contain y_o since every ray r_j has a unique intersection point $p_j^o \neq y_o$ with the hyperplane, (because y_o is a vertex of the polyhedron) and we can assume that $d^oy_o > d_o^o$. If we partition T into T_1 and T_2 as follows:

$$T_1 = T \cap \{x/d^ox > d_o^o\}$$

$$T_2 = T \cap \{x/d^ox \leq d_o^o\}$$

then $y_o \in T_1$. $T_1 \subset T_3 = T \cap \{x/d^ox \geq d_o^o\}$ and T_3 has extreme points $\{P^o, y_o\}$ ($P^o = \{p_j^o/j \in R\}$) and each of these extreme points belongs to C. Thus $T_3 \subset C$. The only elements of T_3 that are in C_b are the elements of P^o. Thus $T_1 \subset C - C_b$. So, if $(C - C_b) \cap S = \emptyset$ then $T_1 \cap S = \emptyset$ so that $S \subset T_2$. Thus, the constraint:

$$d^ox \leq d_o^o \qquad (14)$$

excludes y^o but no integer points.

The cut (14) is an intersection cut. The problem is to find a

suitable convex set C so that:

(i) $y_o \in C - C_b$

(ii) $(C - C_b) \cap S = \emptyset$

and for computational reasons C should be chosen such that the coefficients (d^o, d_o^o) can be easily found.

Balas [21] proposed as convex set C the hypersphere circumscribing the unit hypercube which contains the LP optimum and has integer vertices. The cut is then generated by the intersected points of the half-lines originating at the LP optimum with the hypersphere. There have been many proposals recently for choosing a convex set C. e.g. [26] , [27] , [22] , [28] , [29] , [23] .

While this approach of the intersection cut is very appealing theoretically, it can present many computational problems as n intersection points with the boundary of the convex set have to be found. Convex sets that give good cuts present computational problems and vice-versa. Burdet & Breu in [25] report unsatisfactory experimental results on intersection cuts.

A common feature of the cutting plane approaches is that they can be very effective in the initial iterations while in the subsequent iterations they are slow. This discrepancy of performance, if used intelligently in combination with other IP approaches, can be advantageously exploited.

2.3. ENUMERATIVE METHODS

This category includes the branch and bound type of algorithms and the implicit enumeration type. The latter is specifically applied to the 0 - 1 linear programming problem. These two types are very similar in nature; both use a tree structure to represent the solution process and a backtracking procedure to examine if any undeveloped branches need further investigation. The methods under the heading of branch and bound tend to base their fathoming tests on the associated LP problems while implicit enumeration approaches primarily base their tests on the logical implications of the constraints. However, there are algorithms which incorporate both approaches.

All methods employ heuristic rules of a kind that we shall describe.

2.3.1. THE BRANCH AND BOUND APPROACH [32] , [33] , [34] , [35] , [36] , [37]

The pioneering work in the field was that of Land & Doig [16] , which was modified by Dakin [30] . Usually, the branch and bound algorithms can be also applied to the mixed integer programming problem. Here we shall examine its application to the all integer programming problem.

Consider the IP problem:

$$\left.\begin{array}{l} \max\ cx \\ \\ \text{s.t. } Ax = b \\ \\ x \geqq 0, \text{ integer} \end{array}\right\} \quad F$$

The traditional branch and bound algorithms solve a series of LP problems derived from F and which differ in their solution domain. Each node of the tree corresponds to a certain LP problem with its respective solution space. A node is said to be fathomed when any further exploration from that node is not necessary. A node that is not fathomed is called live.

A node has to be chosen among the live ones so that this node becomes the next one to consider for fathoming or separation. In the latter case, the solution space corresponding to the chosen node is partitioned by:

1) Selecting a variable x_j which has a fractional value in the LP optimal solution corresponding to that node,

$$x_j = [x_j] + f \quad 0 < f < 1,$$

and 2) applying the dichotomy:

$$\left.\begin{array}{l} x_j \geqq [x_j] + 1 \\ \\ x_j \leqq [x_j] \end{array}\right\} \quad (15)$$

so that two nodes are created.

The variable x_j is then called the separation variable. In the case where the node is fathomed one either backtracks until a live node

is encountered or one chooses a new live node to branch from by other considerations; the course of action depends on the specific algorithm. The enumeration is complete when there are no live nodes left. The whole procedure involves the calculation of bounds on the objective function in order to accelerate the process and curtail enumeration.

A basic branch and bound algorithm is as follows:

Let a lower bound to the optimal value of the objective function be z^o.

Step 1 Solve the LP version of problem F without any integrality requirements. If all the variables take integer values, this is the optimal solution of F. If not, go to step 2.

Step 2 Choose a variable which has a non-integer value. Employ the dichotomy (15) and add the two resulting LP problems to the list of unsolved problems (live nodes). Go to step 3.

Step 3 Choose an unsolved problem from the list and go to step 4. If there is no problem in the list, then the process terminates. The best feasible integer solution found so far, if any, is the optimal. Otherwise the problem has no feasible solution.

Step 4 Solve the chosen problem as an LP. If it has no feasible solution or if the resulting optimal value of the objective function z is less than or equal to z^o then the node is fathomed. Go to step 3. Otherwise go to step 5.

Step 5 If the optimal solution to the LP problem satisfies the integrality requirements, record it and let its optimal value replace the value z^o, go to step 3. Otherwise, go to step 2.

The different branch and bound algorithms mainly differ in:

a. The bounds that they employ to facilitate the fathoming process of Step 4. (In the above description only the LP solution is used for bounding).

b. How one selects the separation variable in step 2,

c. and how one chooses the live node on which to branch

a. Calculating bounds

An idea that had found widespread application is that of penalties, [32],[33],[34],[35]. The values of the penalties are extracted from the optimal LP tableau and express the amount by which the value of the objective function is decreased if a certain constraint (15) is imposed. Penalties can be found for every basic variable and for each constraint of the form (15), so that each variable has a "down-penalty" and an "up-penalty".

Consider the variable $x_{B_i} = [y_{io}] + f_{io}$. If the constraint $x_{B_i} \leq [y_{io}]$ is introduced the corresponding decrease in the value of the objective function after the first dual simplex iteration is: $\frac{f_{io} y_{ok}}{y_{ik}}$

Thus, we say that d_i is a down-penalty:

$$d_i = \min_{j \in R} \left[\frac{f_{io} y_{oj}}{y_{ij}}, \; y_{ij} > 0 \right]$$

where R is the set of the non-basic variables; d_i is a lower bound on the decrease of the value of the objective function if the separation variable is x_{B_i}. Similarly an up-penalty corresponding to the constraint $x_{B_i} \geq [y_{io}] + 1$ is:

$$u_i = \min_{j \in R} \left[(f_{io} - 1) \frac{y_{oj}}{y_{ij}}, \; y_{ij} < 0 \right]$$

Since, either $x_{B_i} \leq [y_{io}]$ or $x_{B_i} \geq [y_{io}] + 1$ must be imposed,

$$p_i = \min (u_i, d_i)$$

is a lower bound on the decrease of the objective function when the separation is based on x_{B_i}.

Another bound can be derived by taking into account the integrality requirement on the non-basic variables. For example, some non-basic variable must become positive and therefore not less than one. Thus, a simple valid bound which does not depend on the partitioning row, can be:

$$p = \min_{j \in R} y_{oj}$$

b. The choice of the separation variable

Several rules have been proposed to select the separation

variable and one that has been widely adopted is to select that variable x_{B_k} with the greatest "up-penalty" or "down penalty" [30]. The associated bounds of the two descendant problems that are put in the list are then:

$$z + u_k$$

$$z + d_k$$

where z is the value of the objective function of the parent node problem.

c. Choice of a node to consider next

There are basically two strategies for choosing the node to consider next. The first, which is the one most often used, is the "depth-first" approach where one branches on the most recently created problem. This approach facilitates the computing process as no searching of the list of problems (which await solving) is necessary, if one puts the newest problem on the top of the list, and "push down" the problems already in the list. So, each time that step 3 has to be performed the "top" problem is examined. The aim of this strategy is, obviously, to find an integer feasible solution quickly. This policy has the advantage that a minimum of computer storage is required, since the "top" problem in the push-down stack can always be generated from the current state of the search without explicitly storing the whole stack.

In, the other approach: "Branch from lowest bound" one selects

that problem for consideration which has the lowest bound associated with it. The advantage of this policy is that the total number of branching operations is at a minimum.

2.4. IMPLICIT ENUMERATION

Implicit enumeration algorithms apply to the all-integer problem and specifically to the 0 - 1 linear programming problem:

$$\left.\begin{array}{ll} \min\ z = \sum_{j=1}^{n} c_j x_j & \\ \text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i & i = 1, \ldots, m \\ x_j \in \{0, 1\} & j = 1, \ldots, n \end{array}\right\} \quad P$$

A "partial solution" S is defined as an assignment of binary values to a subset N^S of variables, $N^S \subset N$ where N is the set $\{1, \ldots, n\}$. The variables of the set $(N - N^S)$ are called "free". A partial solution is augmented by fixing a variable x_j, $j \in N - N^S$, at the value 1 or 0. A completion of a partial solution is the solution resulting from fixing all free variables each of them at either 1 or 0.

A partial solution S is fathomed when one of the following conditions is true:

(i) S has no feasible completion.

(ii) The best feasible completion of S has a value z, which is worse than the value of the incumbent z^o (best feasible solution already found). i.e. $z \geq z^o$ since problem P is being minimized.

(iii) S has a feasible completion whose value z is better than the currently known value z^o. In this case, z^o is replaced by z,

and the currently best solution is updated.

If a partial solution cannot be fathomed, it is augmented by setting one of the free variables to either 1 or 0.

The whole procedure can be viewed as a solution tree where each node corresponds to a partial solution S and the arcs indicate how a solution was obtained from a previous partial solution. A basic feature of the algorithms which come under the heading implicit enumeration is the use of the backtrack procedure.

The pioneering work in the field of implicit enumeration was done by Balas [1] . Other work in this area is [2] , [3] , [4] , [5] , [6] , [7] , [8] , and many more.

These algorithms can mainly be distinguished by their method of fathoming a partial solution, and by their method of augmenting a partial solution. The general framework of these algorithms is as follows [2] :

The notational convention adopted is: if j is an element of S this implies $x_j = 1$ and if $-j \in S$ this implies $x_j = 0$. Hence, if $n = 5$ and $S = \{3, 4, -2\}$ then $x_3 = 1$, $x_4 = 1$, $x_2 = 0$ and x_1, x_5 are free. The underline of an element of S denotes that the partial solution which is identical to S up to the underlined element but with that element replaced by its complement, has been fathomed. E.g. $S' = \{3, 4, \underline{2}\}$ denotes that $S = \{3, 4, -2\}$ has been fathomed.

The simplified procedure

Initially S = $\emptyset$ or any other partial solution without any underlines. The general procedure then, is:

Step 1 Attempt to fathom S.

If the attempt is successful go to step 3.

If not go to step 2.

Step 2 (Augmentation) Augment S by adding on the right j or -j where x_j is any free variable. Go to step 1.

Step 3 (i) If all completions of S are infeasible go to step 4.

(ii) If the best feasible completion of S is better than the incumbent solution store it as the new incumbent and go to step 4.

(iii) If no feasible completion of S is better than the incumbent go to step 4.

Step 4 (Backtracking) Locate the rightmost element of S which is not underlined. If none exists terminate, the optimum value z^* is equal to the current z^o. Otherwise, replace the element by its underlined complement and delete all elements to the right. Go to step 1.

The procedure just described applies the depth-first approach. As it was mentioned earlier, the algorithms differ mainly in their application of steps 1 and 2. The tools that these algorithms use to accelerate the fathoming process can be classified into three main

categories:

a. Logical analysis of the structure of the problem.

b. Use of a surrogate constraint.

c. Use of LP.

The main limitation of the algorithms under category (a) is that they tend to apply their logical tests to only one constraint at a time, so that the joint implications of several constraints are lost. A remedy to this shortcoming is the introduction of the surrogate constraint which can capture more of the joint implications of the set of constraints.

The most efficient algorithms under the heading of implicit enumeration use a combination of (a), (b), and (c) and of those, Geoffrion's [42] where LP is used throughout the process of the algorithm, seems to be the most efficient. Balas, in his filter method [43] uses LP at the beginning of the algorithm, to create a surrogate constraint which, however, is "loose" and is not adapted to the new information generated during the process of the algorithm.

We come now to the description of some of the most representative methods of the field of implicit enumeration.

2.4.1. Balas method [1] as formulated by Geoffrion [2]

Consider problem P in matrix form:

$$\left.\begin{array}{l} \min\ cx \\ \\ \text{s.t.}\ \ Ax \geq b \\ \\ c \geq 0,\ \ x_j \in \{0, 1\} \end{array}\right\} \quad P$$

The fathoming of a partial solution S applies the tests (i) and (ii) as shown below. Define the set of variables T^S:

$T^S = \{ j \text{ free: } cx^S + c_j < z^O \text{ and } a_{ij} > 0 \text{ for some i such that } y_i^S < 0 \}$

where x^S is the completion of S when all free variables take the value 0, z^O is the current value of the incumbent, and $y^S = Ax^S - b$. (obviously, if there exist some $y_i^S < 0$, the solution x^S is not feasible to problem P.)

(i) If $T^S = \emptyset$ then there cannot be a feasible completion of S that is better than the incumbent since no free variable can take the value 1 and contribute to feasibility without leading to a higher value than z^O.

(ii) If $y_i^S + \sum_{j \in T^S} \max [0, a_{ij}] < 0$ for some i such that $y_i^S < 0$ then there is no way to select free variables so to eliminate infeasibility.

The augmentation stage of this algorithm proceeds as follows: Select that j, $j \in T^S$ which creates the least amount of total infeasibility in the next x^S i.e. select that j which makes

$$\sum_{i=1}^{m} \min [y_i^S + a_{ij}, 0]$$

an algebraic maximum.

Once a solution is fathomed the algorithm backtracks.

Improvements on the above algorithm.

Many improvements can be applied to the above algorithm and some of them are as follows:

In step 2 (section 2.4.1.) only one variable is added in S at a time. In general, a partial solution S can be augmented by a collection of elements rather than by one element. This obviously does not exclude any feasible completions of S that are better than the incumbent [2].

The infeasibility measure used in step 2 for augmenting S is only one of many alternatives. For example, one could measure infeasibility by the most infeasible constraint or by the number of infeasible constraints [2].

Glover & Zionts [4] proposed an improvement for the test (ii) of Balas, which is the following:

If there exists a constraint i such that $y_i^s < 0$ and $\left| y_i^s \right| \frac{c_j}{a_{ij}^+} \geq z^o - cx^s$ for all $j \in T^s$ $(a_{ij}^+ > 0)$ then there are no feasible completions of S that will improve on the best solution already found.

2.4.2. The "multiphase-dual" algorithm of Glover [3]

Consider problem P in matrix form as in 2.4.1.

The basic feature of the method is the use of a surrogate constraint or s-constraint which is defined as a non-negative linear combination of the constraints of P, and which has the form:

$$ax \geq b_o$$

where a is an n-component row vector and b_o a scalar, and $a = wA$, $b_o = wb$ where w is an 1x m row vector $w = (w_1, w_2, \ldots, w_m)$ such that $w \geq 0$ and $w_i > 0$ for at least one i, $1 \leq i \leq m$. It may be noted that each of the constraints of P is a special case of the s-constraint, obtained by assigning a weight of 1 to the indicated constraint and by assigning all other constraints a weight of 0. The function of the surrogate constraint is to serve as a substitute for some of the original problem constraints in guiding the progress of the algorithm.

Consider the one-constraint problem:

$$\left.\begin{array}{l} \min\ cx \\ \text{s.t.}\ \ ax \geq b_o \\ \qquad x_j \in \{0, 1\} \end{array}\right\} \quad P''$$

P" is a relaxation of the original problem P. If the optimal solution of P" is feasible to P, then it is the optimal solution of P.

Given two s-constraints derived from the same problem P, the first is "stronger" if the optimal solution to P" obtained with that constraint yields a greater value for cx than the optimal solution to P" obtained with the second constraint.

The aim, of course, is

1. to find a strong s-constraint
2. to find solutions of P" that are also feasible to P.

Before illustrating the method of deriving a strong s-constraint, we have to describe how one can obtain a good approximate solution to P", something that will be used in deriving a strong s-constraint.

Consider problem P". In any optimal ~~fractional~~ solution to P" and in any optimal integer solution to P", the following is true:

$$\text{if } c_j \geq 0 \quad \text{and} \quad a_j \leq 0 \longrightarrow x_j = 0$$

$$\text{if } c_j < 0 \quad \text{and} \quad a_j > 0 \longrightarrow x_j = 1$$

Thus, problem P" can be reduced to a smaller size problem where for each j, c_j, $a_j > 0$, by making the substitution $x'_j = 1 - x_j$ for those j for which c_j, $a_j < 0$. An auxiliary indexing of the variables can now be defined such that:

$$p < q \quad \text{if} \quad \frac{c_{j_p}}{a_{j_p}} < \frac{c_{j_q}}{a_{j_q}}$$

Further, let r be the least integer ($1 \leq r \leq n$) for which $\sum_{p \leq r} a_{j_p} \geq b_o$ Then, an optimal ~~fractional~~ solution to P" is given by:

$$x_{j_p} = 1 \quad \text{if} \quad p < r$$

$$x_{j_p} = 0 \quad \text{if} \quad p > r$$

$$\text{and } x_{j_r} = \frac{(b_o - \sum_{p>r} a_{j_p})}{a_{j_r}}.$$

of course, if r does not exist (i.e. $\sum_{k \leq n} a_k < b_o$) then P'' has no feasible solution.

An approximation to the optimal integer solution of P'' is given by:

Begin with $q = r - 1$ and then decrease q by steps of 1, removing each a_{j_q} from $\sum_{p \leq r} a_{j_p}$ which allows the sum of the remaining terms to equal or exceed b_o. Then set:

$x_{j_p} = 0$ if $p > r$ or if a_{j_p} was removed from the summation,

$x_{j_p} = 1$ otherwise.

Creation of the best surrogate constraint

Let $\bar{x}$ denote a feasible solution to P'' where P'' is defined relative to a given surrogate constraint Φ. Then Φ may be divided into two component constraints F and G (i.e. $\Phi = F + G$) where F is a linear combination of those constraints of problem P that are satisfied by $\bar{x}$ and G is a linear combination of those constraints of P that are unsatified by $\bar{x}$ where the weight given to each constraint composing F and G is the same weight as the constraint had in the surrogate constraint Φ. Further let f denote the amount by which $\bar{x}$ oversatisfies F and g denote the amount by which $\bar{x}$ undersatisfies G. (Obviously $f \geq g$ since Φ is satisfied by $\bar{x}$).

The constraint $F + kG$ will be satisfied by $\bar{x}$ for $0 < k \leq \frac{f}{g}$ and will fail to be satisfied by $\bar{x}$ when $k = \frac{f}{g} + \varepsilon$ where ε is a small positive number. In [3] ε is taken as $\varepsilon = 0.1$. Using the above, the strongest surrogate constraint is then derived as follows:

A "stopping rule" is established to limit the number of s-constraints that will be produced as candidates for the selected strongest s-constraint. Say that this number is λ.

Step 1 Create an s-constraint by assigning each constraint of P a positive weight, say unity. Go to step 2.

Step 2 Find a good approximate solution to P'' (as described above) where P'' is defined relative to the current s-constraint. If P'' has no feasible solution neither does P and the process terminates. If P'' has a feasible solution $\bar{x}$, go to step 3.

Step 3 If this step has not been visited before, go to step 4. If $\bar{x} < \bar{\bar{x}}$, where $\bar{\bar{x}}$ denotes the value of the feasible solution of P'' obtained relative to the previous s-constraint go to step 6. If $\bar{x} > \bar{\bar{x}}$ go to step 4.

Step 4 If $\bar{x}$ is feasible to P or if the current s-constraint is the λth generated go to step 7. If not, go to step 5.

Step 5 Identify the components F and G of the current s-constraint. Replace the s-constraint F + G by the new s-constraint $F + (\frac{f}{g} + \epsilon)G$, which is now designated current. Go to step 2.

Step 6 The previous s-constraint is the selected s-constraint. Go to step 8.

Step 7 The current s-constraint is the selected s-constraint.

Step 8 End

Different tests of a logical nature are employed by the algorithm to determine either that no feasible completion of a partial solution exists or that some of the problem variables not already assigned a specific value must be set equal to a given value. Also a test is employed to determine whether a constraint of P is locally non-binding, that is whether the constraint can be discarded while investigating the current partial solution without eliminating any optimal solution of P. Additionally, the objective function is treated as a constraint.

2.4.3. Balas' filter algorithm or accelerated additive algorithm [43]

In this algorithm, Balas has used the concept of a surrogate constraint together with the notions used in his additive algorithm. Consider the corresponding continuous problem to P, P' and let the dual of P' be G'. Let $\bar{x}$ and $(\bar{w}, \bar{v})$ be a pair of optimal solutions to P' and G' respectively, where w_i and v_j are the dual variables associated with the ith constraint and the jth upper-bound constraint $(0 \leq x_j \leq 1)$ respectively. Now, consider the "filter-problem" D:

$$\left.\begin{array}{lll} \min\ cx & & \\ \text{s.t.}\ \ ax \geq b_o & (16) & \\ x_j \in \{0, 1\} & & \end{array}\right\} \quad D$$

where $a = \bar{w}A$, $b_o = \bar{w}b$. Let D' be the continuous version of D.

It can be noted that constraint (16) is a special case of Glover's surrogate constraint.

The algorithm is a tree-search where more than two branches can emanate from a node and it is dual in the sense that the first feasible solution to P found is also optimal. In the process, feasible solutions to D are generated such that:

a. The sequence of associated values cx is non-decreasing and

b. When a certain value cx^S has been reached, then all feasible solutions to D such that $cx < cx^S$ have been explicitly or implicitly enumerated. Thus, the first term of the sequence that is a feasible solution to P, is also optimal.

The algorithm aims at finding feasible solutions to problem D which are then tested for feasibility to P. Every node of the tree corresponds to a partial solution S, which might or might not be a feasible solution to D. With every node of the tree (i.e. with every partial solution S) there is a value $z'_S = cx^S$ associated with it, where x^S is an optimal solution to the continuous problem D'_S. (D'_S is the resulting problem from D' after introducing the values of the fixed variables x_j ($j \in S$) to the problem D'.) Obviously, if x^S is integer it is feasible to D.

An indexing of the variables is used by the algorithm which is as follows: we have $j_1 < j_2$ (assuming $c \geq 0$) whenever one of the three following situations holds.

1. $a_{j_1} > 0 \quad a_{j_2} \leq 0$

2. $a_{j_1} > 0 \quad a_{j_2} > 0, \quad \frac{c_{j_1}}{a_{j_1}} < \frac{c_{j_2}}{a_{j_2}}$

3. $a_{j_1} \leq 0 \quad a_{j_2} \leq 0 \quad a_{j_1} > a_{j_2}$

Any indexing that satisfies the above rule is called optimal.

Selecting the node on which to branch: that node is chosen which corresponds to the minimum value of z'_s. (It is assumed that all, values z'_s together with the corresponding S solutions are stored.)

(i) If the solution x^s corresponding to z'_s is feasible to problem D, "it has passed through the filter" i.e. it satisfies a necessary condition for feasibility to problem P. If it is also feasible to P, it is optimal, if it is not feasible to P, but it is integer, x^s is submitted to some tests, derived from the original additive algorithm, to examine the possibility of having a feasible solution to P along the current branch. If the tests are inconclusive, we branch on the node in question in a way to be described later. If the tests show that there exists no possibility of having a feasible solution to P the present node can be abandoned since none of its descendants can yield a feasible solution to P.

(ii) If the solution x^s corresponding to z'_s is not feasible to D then branching takes place immediately.

Branching The descendants of a partial solution S can be found as follows: consider the t free variables x_j, $j \notin S$, in their order of optimal indexing and then construct the solutions $S_\ell = \{ S, j_\ell \}$ for $\ell = 1, 2, \ldots, t$. i.e. add one free variable at a time in the order of optimal indexing of these variables. For each index ℓ, proceed as follows:

lst part Compute the new right hand side:

$$b'_o = b_o - \sum_{j \in S_\ell} a_j x_j$$

If $b'_o \leq 0$ compute $z^{S_\ell} = \sum_{j \in S_\ell} c_j x_j$ and go to part 2. If $b'_o > 0$ look for an index $r \in N - S_\ell$ such that

$$\sum_{\substack{j \leq r-1 \\ j \in N - S_\ell}} a_j < b'_o \leq \sum_{\substack{j \leq r \\ j \in N - S_\ell}} a_j \qquad (17)$$

If such an index r exists, compute the value of the objective function z'_{S_ℓ} :

$$z'_{S_\ell} = \sum_{j \in S_\ell} c_j x_j + \sum_{\substack{j \in N - S_\ell \\ j \leq r-1}} c_j + \frac{c_r}{a_r} \left(b'_o - \sum_{\substack{j \in N - S_\ell \\ j \leq r-1}} a_j \right)$$

and go to part 2.

If no index r exists for which (17) is satisfied eliminate the solution

S_ℓ from the list and go back to the choice step, i.e. select a new node to branch on.

2nd part If $N - S_\ell \neq \emptyset$ augment by 1 the value of ℓ construct the next partial solution $S_{\ell+1} = \{S, j_{\ell+1}\}$ and reapply the first part of the branching step.

2.4.4. Geoffrion's algorithm using a surrogate constraint

Geoffrion [42] uses the schema of the basic backtracking algorithm presented in section 2.4. together with a surrogate constraint. The surrogate constraint used is the "strongest", relative to a partial solution S, and its computation involves the use of LP. Here LP is used not once, as it is done by Balas in his filter algorithm, but many times during the process of the algorithm, namely whenever it is necessary to add a surrogate constraint so as to fathom a partial solution.

Computing strongest surrogate constraints by LP

Consider problem P (section 2.4.1.)

Definition The surrogate constraint $w^1(Ax - b) + (z^o - cx) \geqslant 0$ is said [42] to be stronger relative to S than the surrogate constraint $w^2(Ax - b) + (z^o - cx) \geqslant 0$ if the maximum of the left hand side of the first constraint is less than the maximum of the left hand side of the second constraint, where the maximum are taken over binary values of the free variables and where w is a non-negative m-vector and z^o is the value of the currently best

known feasible solution of P.

Finding a strongest surrogate constraint is, then, the problem of minimizing over all $w \geq 0$ the expression:

$$\sum_{i=1}^{m} w_i \Big(\sum_{j \in S} a_{ij} x_j - b_i \Big) + z^o - \sum_{j \in S} c_j x_j \; + \\ + \max \Big\{ \sum_{j \notin S} \Big(\sum_{i=1}^{m} w_i a_{ij} - c_j \Big) x_j / x_j \in \{0,1\},\ j \notin S \Big\} \qquad (18)$$

Now for any $w \geq 0$ we have:

$$\max \Big\{ \sum_{j \notin S} \Big(\sum_{i=1}^{m} w_i a_{ij} - c_j \Big) x_j / x_j \in \{0,1\},\ j \notin S \Big\}$$

$$= \max \Big\{ \sum_{j \notin S} \Big(\sum_{i=1}^{m} w_i a_{ij} - c_j \Big) x_j / 0 \leq x_j \leq 1,\ j \notin S \Big\} = \qquad (19)$$

$$= \min \Big\{ \sum_{j \notin S} v_j \;\; v_j \geq 0 \text{ and } v_j \geq \sum_{i=1}^{m} w_i a_{ij} - c_j,\ j \notin S \Big\} \qquad (20)$$

where (20) follows from the LP dual theorem.

Using (20) in (18) we have the following LP problem:

$$\left.\begin{array}{l} \min \quad \sum_{i=1}^{m} w_i b'_i + z^o - z^s + \sum_{j \notin S} v_j \\ \text{s.t.} : \quad v_j \geq \sum_{i=1}^{m} w_i a_{ij} - c_j \quad j \notin S \\ \qquad\quad v_j \geq 0 \quad j \notin S \\ \qquad\quad w \geq 0 \\ \text{where } b'_i = \sum_{j \in S} a_{ij} x_j - b_i \text{ and } z^s = \sum_{j \in S} c_j x_j \end{array}\right\} \; (LP_s)$$

If the optimal solution to the above problem is not positive then the resulting surrogate constraint has no binary solution and the current partial solution S is fathomed. If, on the other hand, the optimal solution of (LP_s), is positive, any optimal w in (LP_s) yields a strongest surrogate constraint relative to S. If, in addition, the dual variables are all integer a feasible solution to P is produced.

2.4.5. The approach of Lemke & Spielberg [6]

Lemke & Spielberg improved the additive algorithm of Balas [1] with an eye towards computational efficiency. Consider problem P, section 2.4.1. The main additional feature that they have introduced concerns the augmentation of a partial solution S and is as follows:

It is advantageous to delimit the set of free variables to a smaller subset of preferred variables and Lemke & Spielberg (L & S) found that the following preferred set of variables is quite effective computationally: for some i such that $\sum_{j\in S} a_{ij}x_j - b_i = y_i^s < 0$ the ith constraint can be written as $\sum_{j\notin S} a_{ij}x_j \geq -y_i^s$ Let D_i be the set:

$$D_i = \{ j/j \text{ free}, a_{ij} > 0 \}$$

The number of elements in D_i can be reduced further by a process

of "complete reduction" and to this end the coefficients a_{ij} ($a_{ij} > 0 \quad j \notin S$) are ordered in descending order of magnitude. E.g. consider $4x_2 + 2x_4 + x_3 - 5x_5 \geq 5$ which implies $x_2 \geq 1$ and thus the associated D set was reduced by one.

One may construct one preferred set for each row i, and that row is "preferred" which is associated with that set D* having the minimum number of elements. The Balas test is then applied: i.e. for each $j \in D^*$ the following value is computed:

$$t_j = \sum_{i-} (y_i^s + a_{ij})$$

where $\sum_{i-}$ denotes that the sum is over those i for which $y_i^s + a_{ij} \leq 0$. That variable is then selected for augmenting S, which yields the maximal value for t_j.

L & S report "that this (the above) procedure reduces computing time substantially, sometimes by factors from 3 to 5" [6]

Another feature of the L & S algorithm is the focusing on the cost constraint (objective function) in order to fix more variables at 0 and delimit the set of free variables. The c_j's are put in ascending order, $c_j \leq c_{j+1}$. If for some q one has $z^s + c_q \geq z^o$ all free variables x_j such that $j \geq q$ are fixed at 0, where $z^s = \sum_{j \in S} c_j x_j$ and z^o is the value of the best feasible solution to P found so far.

A test that is employed for fathoming a partial solution is:

Define $\sigma_i = \sum_{j \notin S} a^+_{ij}$ for i = 1, 2, ..., m + 1, where $a^+_{ij} > 0$ and the (m + 1) th constraint is the cost constraint. Compute $d_i = y^S_i + \sigma_i$ for every i. If any $d_i < 0$ backtrack as no augmentation of S can lead towards feasibility.

L & S have also incorporated the above into an algorithm for the Mixed Integer Programming case.

2.4.6. The 'pseudo-Boolean' approach of Hammer & Rudenau [5]

A linear function of binary variables is called here, a pseudo-Boolean function. Hammer & Rudenau employ a depth-first tree-search where all the coefficients of the problem are made positive through the transformation $x'_j = 1 - x_j$. This implies that two "variables" x_j, x'_j can correspond to the index j, if for some i, $a_{ij} > 0$, and if for some other i $a_{ij} < 0$. The problem P is said to be in canonical form if all its coefficients are positive and if for every i the a_{ij}'s are in descending order.

In [5] the way of fathoming a partial solution S is basically the quantity $(\sum_{j \notin S} a^+_{ij} - b'_i)$, where $\sum a^+_{ij}$ denotes the sum of the positive coefficients. If for some i this quantity is negative the branching stops and we backtrack to the last created node. If $\sum_{j \notin S} a^+_{ij} - b'_i \geq 0$ the branching proceeds and that variable is fixed at 1 which has the biggest coefficient in the system of constraints.

2.4.7. Hammer & Nguyen approach: "A partial order in the solution space of bivalent programs" [7]

In this approach, the emphasis is on the structure of the problem. "Order relations" are constructed for the variables of the problem which can lead to forced values for certain variables, equality or non-equality of certain pairs of variables, etc. The results of this structure-analysis of the problem are then used in an algorithm that will be described later. Consider the system of m inequalities in the canonical form:

$$\sum_{j=1}^{n} a_{ij} x_j^{\alpha_{ij}} \leq b_i \qquad i = 1, \ldots, m \qquad (21)$$

where $a_{ij} \geq 0$ for all i and j, $x_j \in \{0, 1\}$

and $\alpha_{ij} \in \{0, 1\}$

$$x_j^{\alpha_{ij}} = x_j \quad \text{if} \quad \alpha_{ij} = 1$$

$$x_j^{\alpha_{ij}} = x_j' = 1 - x_j \quad \text{if} \quad \alpha_{ij} = 0$$

Order relations

An order-relation is of the form $x_j^{\alpha} \leq x_k^{\beta}$. Let R be the set of all such order-relations which hold in the system (21).

Construction of R:

I. $\exists i, j, a_{ij} > b_i \longrightarrow x_j^{\alpha_{ij}} = 0 \in R$

II. $\exists i, j_1, j_2 \quad a_{ij_1} + a_{ij_2} > b_i \longrightarrow x_{j_1}^{\alpha_{ij_1}} \leq x_{j_2}^{\bar{\alpha}_{ij_2}} \in R$

If the case I can be applied the process stops because it is forcing

a variable, and hence the whole process can be repeated after setting that variable to its value. In case II one of the variables j_1, j_2 has to take the value 0. Once R is constructed, the obvious conclusions that can be drawn from it are the following:

(i) if $x_j^{\alpha} \leq 0 \longrightarrow x_j = \bar{\alpha}$

i.e. if $\alpha = 1$ $x_j \leq 0 \longrightarrow x_j = 0$

if $\alpha = 0$ $x_j' = 1 - x_j = 0 \longrightarrow x_j = 1$

(ii) if $x_j^{\alpha} \leq x_j^{\bar{\alpha}} \longrightarrow x_j = \bar{\alpha}$ (i.e. if $\alpha = 1$ we have $x_j \leq 1 - x_j$, $x_j = 0$)

if $\alpha = 0$ $1 - x_j \leq x_j \longrightarrow x_j = 1$

(iii) if $x_j^{\alpha} \leq x_k^{\beta}$ and $x_j^{\alpha} \leq x_k^{\beta} \longrightarrow x_j = \bar{\alpha}$

e.g. say $\alpha = \beta = 1$ We then have $x_j \leq x_k$ & $x_j \leq 1 - x_k$ which obviously leads to $x_j = \bar{\alpha} = 0$

(iv) if $x_j^{\alpha} \leq x_k^{\beta}$ and $x_k^{\beta} \leq x_j^{\alpha} \longrightarrow x_j^{\alpha} = x_k^{\beta}$

It can easily be seen that in the first three cases a variable has a forced value while in case (iv) the system can be condensed.

An example as found in [7]

Consider the system:

$$-6x_1 + 8x_2 + 6x_3 - x_4 - 5x_5 + 3x_6 \leq -1$$

$$-2x_1 - 7x_2 + 5x_3 - 5x_4 + 5x_5 + 4x_6 \leq -4$$

$$-8x_1 - x_2 + 4x_3 + 4x_4 + 4x_5 - 3x_6 \leq -1$$

where $x_j \in \{0, 1\}$ $j = 1, \ldots, 6$

Rewriting the system into the canonical form we have:

$$6\bar{x}_1 + 8x_2 + 6x_3 + \bar{x}_4 + 5\bar{x}_5 + 3x_6 \leq 11$$
$$2\bar{x}_1 + 7\bar{x}_2 + 5x_3 + 5\bar{x}_4 + 5x_5 + 4x_6 \leq 10$$
$$8\bar{x}_1 + \bar{x}_2 + 4x_3 + 4x_4 + 4x_5 + 3\bar{x}_6 \leq 11$$

The first inequality yields:

$$\bar{x}_1 \leq \bar{x}_2$$
$$\bar{x}_1 \leq \bar{x}_3$$
$$x_2 \leq \bar{x}_3$$
$$x_2 \leq x_5$$

The second inequality yields:

$$\bar{x}_2 \leq \bar{x}_3$$
$$\bar{x}_2 \leq x_4$$
$$\bar{x}_2 \leq \bar{x}_5$$
$$\bar{x}_2 \leq \bar{x}_6$$

and the third yields:

$$\bar{x}_1 \leq \bar{x}_3$$
$$\bar{x}_1 \leq \bar{x}_4$$
$$\bar{x}_1 \leq \bar{x}_5$$

Thus R consists of the inequalities

$$\bar{x}_1 \leq \bar{x}_2$$
$$\bar{x}_1 \leq \bar{x}_3$$
$$\bar{x}_1 \leq \bar{x}_4$$
$$\bar{x}_1 \leq \bar{x}_5$$
$$x_2 \leq \bar{x}_3$$

$$x_2 \leq x_5$$
$$\bar{x}_2 \leq \bar{x}_3$$
$$\bar{x}_2 \leq x_4$$
$$\bar{x}_2 \leq \bar{x}_5$$
$$\bar{x}_2 \leq \bar{x}_6$$

But if $(x_j^{\alpha} \leq x_k^{\beta}) \in R$ this implies that $(x_k^{\bar{\beta}} \leq x_j^{\bar{\alpha}}) \in R$

Thus the following set of relations is also included in R:

$$x_2 \leq x_1$$
$$x_3 \leq x_1$$
$$x_4 \leq x_1$$
$$x_5 \leq x_1$$
$$\longrightarrow \quad x_3 \leq \bar{x}_2$$
$$\bar{x}_5 \leq \bar{x}_2$$
$$\longrightarrow \quad x_3 \leq x_2$$
$$\bar{x}_4 \leq x_2$$
$$x_5 \leq x_2$$
$$x_6 \leq x_2$$

We see that $(x_3 \leq \bar{x}_2) \in R$ and $(x_3 \leq x_2) \in R$ which leads to $x_3 = 0$.

Introducing $x_3 = 0$ in R we get R_1 consisting of the following relations:

$$\bar{x}_1 \leq \bar{x}_2$$
$$\bar{x}_1 \leq \bar{x}_4$$
$$\bar{x}_1 \leq \bar{x}_5$$
$$x_2 \leq x_5$$

$$
\begin{array}{ll}
\bar{x}_2 \leq x_4 &) \\
\bar{x}_2 \leq \bar{x}_5 &) \\
\bar{x}_2 \leq \bar{x}_6 &) \\
x_2 \leq x_1 &) \\
x_4 \leq x_1 &) \quad R_1 \\
x_5 \leq x_1 &) \\
\bar{x}_5 \leq \bar{x}_2 &) \\
\bar{x}_4 \leq x_2 &) \\
x_5 \leq x_2 &) \\
x_6 \leq x_2 &)
\end{array}
$$

where the trivial realtions $\bar{x}_1 \leq \bar{x}_3 \leq 1$ etc. were omitted. The relations $\bar{x}_1 \leq \bar{x}_2$ and $\bar{x}_2 \leq x_4$ yield the relation $\bar{x}_1 \leq x_4$. The latter combined with the relation $\bar{x}_1 \leq \bar{x}_4$ imply $\bar{x}_1 = 0$ i.e. $x_1 = 1$.

Intorducing $x_1 = 1$ we get R_2 consisting of the following:

$$
\begin{array}{lll}
\longrightarrow & x_2 \leq x_5 &) \\
 & \bar{x}_2 \leq x_4 &) \\
 & \bar{x}_2 \leq \bar{x}_5 &) \\
 & \bar{x}_2 \leq \bar{x}_6 &) \\
 & \bar{x}_5 \leq \bar{x}_2 &) \quad R_2 \\
 & \bar{x}_4 \leq x_2 &) \\
\longrightarrow & x_5 \leq x_2 &) \\
 & x_6 \leq x_2 &)
\end{array}
$$

The relations $x_2 \leq x_5$ and $x_5 \leq x_2$ yield $x_2 = x_5$ and we have R_3 consisting of the following:

$$\left.\begin{array}{l} \bar{x}_2 \leq x_4 \\ \bar{x}_2 \leq \bar{x}_6 \\ \bar{x}_4 \leq x_2 \\ x_6 \leq x_2 \end{array}\right) \quad R_3$$

No further conclusion can be drawn from R_3.

Thus returning to the original system, introducing $x_3 = 0$, $x_1 = 1$ and replacing x_5 by x_2 we have:

$$\begin{aligned} 3x_2 - x_4 + 3x_6 &\leq 5 \\ -2x_2 - 5x_4 + 4x_6 &\leq -2 \\ 3x_2 + 4x_4 - 3x_6 &\leq 7 \end{aligned}$$

Obviously the third inequality is redundant and hence the system reduces to:

$$\begin{aligned} 3x_2 + \bar{x}_4 + 3x_6 &\leq 6 \\ 2\bar{x}_2 + 5\bar{x}_4 + 4x_6 &\leq 5 \end{aligned}$$

For this example which was specially constructed so to be favourable to the use of order-relations, we see that the system was considerably simplified.

Relations between triplets of variables can also be used but this option requires substantial computing time and the authors do not justify their use.

Tree search with binary order relations

The authors have incorporated the binary order-relations into a tree-search algorithm for the 0 - 1 problem 'P which uses LP and a surrogate constraint. This algorithm has the following skeleton:

Step 1 Construct R. If any conclusions can be drawn from it, introduce the results into R until arriving to a new R where no conclusion can be drawn. Go to step 2.

Step 2 Solve the continuous version of P, P'. Let the LP solution to P' be $(x'_1, \ldots, x'_n)$ of value z (LP). Introduce the objective function as a constraint and enlarge R; if this enlargement of R is possible go to step 1, if not go to step 3.

Step 3 Add to P' those inequalities of R which are not satisfied by $(x'_1, \ldots, x'_n)$ and return to step 2. If no such inequality of R is violated by $(x'_1, \ldots, x'_n)$ go to step 4.

Step 4 Add to P' a surrogate constraint and enlarge R by the resulting binary relations. If this is possible return to step 1, otherwise go to step 5.

Step 5 Choose a variable x_j and examine separately the subproblems where $x_j = 1$ and $x_j = 0$.

The authors of [7] report that the above algorithm is computationally efficient. However, they report that "the use of order relations does not seem to influence substantially the total computing time, but reduces the number of iterations". In addition, very few

problems of any practical size can succumb to the extraction of order-relations.

2.5. Transformation of an IP problem in bounded variables to a knapsack problem in bounded variables.

Any IP problem in bounded variables can be solved as an equality constrained knapsack problem in bounded variables by assigning certain weights $w_1, \ldots, w_m$ to the original constraints [39], [40], [11]. It will be shown how two constraints can be combined into one without changing the set of feasible solutions. Then, by combining the constraints two at a time, it is clear that m constraints can be combined into one.

Consider the two constraints:

$$\left.\begin{array}{l} \sum_{j=1}^{n} d_j x_j - b_d = 0 \\ \sum_{j=1}^{n} f_j x_j - b_f = 0 \end{array}\right\} \qquad (22)$$

and assume that the coefficients d_j, f_j $j = 1, \ldots, n$ are integers.

Let

$$\lambda^+ = \max \sum_{j=1}^{n} d_j x_j - b_d \qquad 0 \leq x_j \leq u_j \text{ integer}, j = 1, \ldots, n$$

$$\lambda^- = \min \sum_{j=1}^{n} d_j x_j - b_d \qquad 0 \leq x_j \leq u_j \text{ integer}, j = 1, \ldots, n$$

i.e. $\lambda^+ = \sum_{j=1}^{n} d_j^+ u_j - b_d$ and $\lambda^- = \sum_{j=1}^{n} d_j^- u_j - b_d$

where d_j^+, d_j^- are the positive and negative coefficients respectively.

Finally, define $\lambda = \max\left[\lambda^+, |\lambda^-|\right]$

$0 \leq x_j \leq u_j$ integer, $j = 1, \ldots, n$

Theorem:

The integer vector x^o, $0 \leq x^o \leq u$ is a solution to (22) if and only if

$$\sum_{j=1}^{n} (d_j + \alpha f_j) x_j^o - b_d - \alpha b_f = 0$$

where α is any integer satisfying $|\alpha| > \lambda$. The proof of this theorem can be found in [11]. Unfortunately the weights derived in this way are usually very large numbers and consequently this hinders the method from being computationally efficient.

An example

$$\begin{aligned} \max \quad & z = 2x_1 + x_2 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 5 \\ & -x_1 + x_2 + x_4 = 0 \\ & 6x_1 + 2x_2 + x_5 = 21 \\ & x_1, \ldots, x_5 \geq 0, \text{ integer} \end{aligned}$$

where $x_1 \leq 3$, $x_2 \leq 3$ $x_3 \leq 5$ $x_4 \leq 3$ $x_5 \leq 21$

For the third constraint we have:

$$\lambda^+ = 6(3) + 2(3) + 1(21) - 21 = 24$$

and $\lambda^- = -21$

$$\therefore \lambda = 24$$

Choosing $\alpha = 25$ and combining the third and second constraints we have:

$$6x_1 + 2x_2 + x_5 + 25(-x_1 + x_2 + x_4) = 21$$

$$\text{or } -19x_1 + 27x_2 + 25x_4 + x_5 = 21 \qquad (23)$$

For the first constraint we have $\lambda^+ = 6$ $\lambda^- = -5$ and $\lambda = 6$

Weighting (23) by 7 and combining it with the first constraint we have:

$$-132x_1 + 190x_2 + x_3 + 175x_4 + 7x_5 = 152$$

and making the coefficient of x_1 positive we have the problem:

$$\max \; z = 6 - 2x_1' + x_2$$

$$\text{s.t. } 132x_1' + 190x_2 + x_3 + 175x_4 + 7x_5 = 548$$

CHAPTER 3

A SEQUENTIAL APPROACH TO INTEGER PROGRAMMING

3.1. Introduction

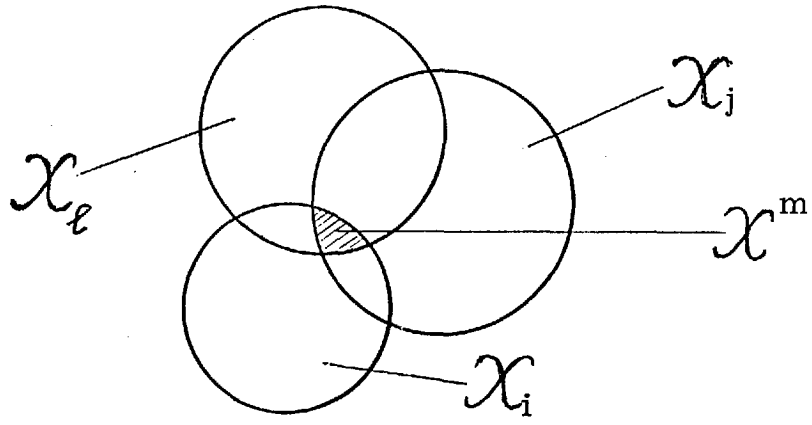
An IP problem is easier to solve if it is decomposed into smaller problems each one having either a few variables and all the constraints or all the variables and a few of the constraints as the solution of the latter problems is simpler. One could either "partition" the original problem "vertically" i.e. create problems having the same number of constraints as the original problem but each of them having only a subset of the variables of the original problem, or one could "partition" the original problem "horizontally" i.e. create problems of smaller size having the same number of variables as the original problem but each derived problem having only a subset of the constraints of the original problem. The former course of action seems more difficult, while the latter, which is followed here, can be very attractive.

This chapter gives a sequential method of minimising a linear function of 0 - 1 variables subject to linear constraints. The method is based on transforming an n-variable m-constraint problem into a sequence of n-variable 2-constraint problems which are easier to solve. The algorithm has been tested on test problems found in the literature and on a set of 80 randomly generated problems and good results have been obtained.

Consider the problem P

$$\left.\begin{array}{ll} \min\ z = \sum_{j=1}^{n} c_j x_j & \\ \text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i & i = 1, \ldots, m \\ x_j \in \{0, 1\}, \quad c_j \geq 0 & j = 1, \ldots, n \end{array}\right\} \text{P}$$

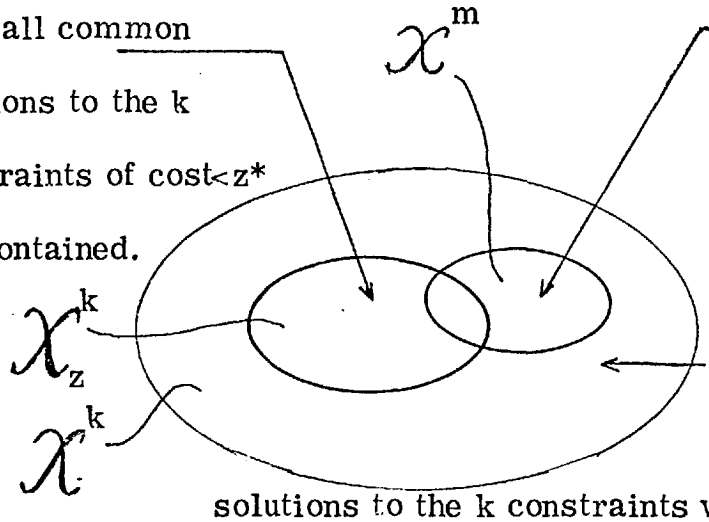
With every constraint i of the above problem P there is a set $\mathcal{X}_i$ associated with it, whose elements are all the 0 - 1 vectors that can satisfy this particular constraint i. The intersection of the m sets $\mathcal{X}_i$ i = 1, ..., m is a set $\mathcal{X}^m = \bigcap_{i=1,\ldots,m} \mathcal{X}_i$ which contains all the feasible solutions to problem P. That element of $\mathcal{X}^m$ which has the minimum cost is the optimal solution to problem P.



Let an optimum solution X* to problem P exist, and let its value be z*. Consider k constraints of problem P where $1 \leq k \leq m$. Let the intersection set of the k sets $\mathcal{X}_i$ be $\mathcal{X}^k$ and that subset of $\mathcal{X}^k$ which contains all elements of cost z, $z \leq z^*$ be $\mathcal{X}^k_z$.

Obviously $\mathcal{X}^m \subset \mathcal{X}^k$. The optimal solution(s) to problem P X^*, is then

$$X^* = \mathcal{X}_z^k \cap \mathcal{X}^m$$



Of course, neither of the sets $\mathcal{X}^k$, $\mathcal{X}_z^k$, $\mathcal{X}^m$ is known in advance, nor need they be found. The method proposed here implicitly enumerates all elements of $\mathcal{X}_z^k$ (a small number of such elements exists as it will be seen later) in increasing order of cost: $z^0, z^1, \ldots, z^p, \ldots, z^*$ i.e. all common solutions to the k constraints that correspond to the increasing costs $z^0, z^1, \ldots, z^p, \ldots, z^*$ will be implicitly enumerated. At every step p of this "path", the solution(s) corresponding to the cost z^p (and which are feasible to the k constraints) are tested for feasibility to the remaining m - k constraints. If a solution is feasible to these m - k constraints this is the optimal solution to the complete problem

P, if not, z^p becomes the lower bound of the value of the objective function at the next step $(p + 1)$ and so on, until the optimum z^* is found, i.e. until we find that element(s) of $\mathcal{X}_z^k$ which is a member of $\mathcal{X}^m$.

The method that follows is dual in the sense that the first feasible solution found to P is the optimal one. The method is iterative and involves the transformation of an n-variable m-constraint problem into $k (k \leq m)$ n-variable 2-constraint subproblems in which only the right hand sides vary from iteration to iteration. These smaller subproblems are ideally suited for solution by dynamic programming. During this iterative process a progressively increasing lower bound to the optimal value of the objective function is derived.

This sequence of bounds is then $z^0, z^1, \ldots, z^p, z^{p+1}, \ldots, z^*$. At each step of the procedure, the common solutions to the chosen k n-variable 2-constraint subproblems are generated by a level by level intersection of the individual "solution trees" of the subproblems. An essential feature of the proposed algorithm (and one which contributes greatly to its efficiency) is that these individual trees are not explicitly found but are represented by the dynamic programming tableaux. In this way, solutions which are not common to the k chosen subproblems are implicitly eliminated early on during the "solution tree" intersections. Thus at the pth step common solutions to the k n-variable 2-constraint subproblems of value

z^p are found by using z^{p-1} as the bound. An iteration then involves the checking of these solutions for feasibility to the complete problem P. If feasibility is found the optimal answer z^* has been obtained and $z^* = z^p$. Otherwise the trial value of the objective function is increased to z^{p+1} (as given later on) and the process repeated by "resolving" the k chosen subproblems and so on.

3.2.1. The Method

Consider the problem P above where all c_j, b_i and a_{ij} coefficients are assumed to be integers. A lower bound to the optimum value z^* could be the corresponding LP optimum z(LP).

Consider k constraints of problem P($1 \leq k \leq m$) and let the constraints be renumbered so that the k chosen ones are numbered 1 to k. Let the following problem $P_i(z^p)$ be associated with each constraint i, $1 \leq i \leq k$ of the original problem P.

$$
\left.
\begin{aligned}
&\min\ w_i = \sum_{j=1}^{n} c_j \xi_j && (1)\\
&\text{s.t.} \quad \sum_{j=1}^{n} c_j \xi_j \geq z^p + 1 && (2)\\
&\qquad \sum_{j=1}^{n} a_{ij} \xi_j \geq b_i && (3)
\end{aligned}
\right\} \quad P_i(z^p)
$$

The solution(s) to problem $P_i(z^p)$ will then have a value w_i^* which is the smallest value greater than the current lower

bound z^p of z and which satisfy the ith constraint of problem P. As the current lower bound z^p is updated during the process the problem $P_i(z^p)$ would have to be solved for different values of the right hand side (rhs) of constraint (2) above and this can be done most conveniently by solving this problem once by dynamic programming for all $0 \leq u \leq U$, where U is an upper bound on the value of the optimal solution z* to problem P.

k dynamic programming tableaux would thus be constructed each one corresponding to a problem $P_i(u)$, $1 \leq i \leq k$, and from these tableaux all solutions to the individual problems $P_i(u)$ can be generated, (although this is not explicitly required by the present method). The dynamic programming tableaux are used by the present algorithm to derive all common solutions of value z^p (or show that none exist), to the k problems $P_i(z^{p-1})$.

The algorithm proceeds as follows:

1. At the pth step, generate all common solutions of value z^p to the k problems $P_i(z^{p-1})$ and let the set of these solutions be X^p.
2. If no such common solutions exist or if none of these is feasible to problem P, proceed to the next step and repeat (1) with z^{p-1} replaced by z^p.

If a common solution feasible to problem P is found, this is the optimal solution of value $z^* = z^p$.

3.2.2. An iteration

The step of increasing the value of the objective function from z^p to z^{p+1} and the testing of the solution(s) X^{p+1} for feasibility to problem P, will be called an iteration.

If after p iterations no feasible solution has been found, then the value of z^{p+1} of the objective function at the next iteration must obviously satisfy:

$$z^{p+1} \geq \max_{1 \leq i \leq k} \{w_i\}$$

where w_i is the minimum value of the objective function of problem $P_i(z^p)$.

If all w_i are equal then it is possible that some solution(s) exists with value equal to w_i and which is feasible for all k constraints. If such solutions exist they will be tested for feasibility to the initial problem P. The method of finding if such solutions exist and if so generating all of them is described in section 3.2.4.

If not all w_i are equal, then obviously no feasible solution to all k constraints of value less than $\Delta = \max_{1 \leq i \leq k} \{w_i\}$ can exist. In this case the rhs of constraint(2) are replaced by Δ to obtain new w_i and so on until all w_i are the same. Let this common value of the w_i be w^p.

3.2.3. The solution of a problem $P_i(u)$

By transforming the solution of problem P into a sequence of solutions to the k subproblems $P_i(u)$ it is implicitly assumed that the 0-1 programming problem $P_i(u)$ with n-variables and 2-constraints can in fact be solved quite routinely for many values of the right hand sides entries. This is indeed the case and problem $P_i(u)$ will be solved by a dynamic programming technique as follows:

The standard dynamic programming recursive equation for two state variables u and v , at stage r, is:

$$g_r(u,v) = \min_{\xi_r} \left[c_r \xi_r + g_{r-1}(u - c_r \xi_r,\ v - a_{ir} \xi_r) \right] \qquad (4)$$

In the present case, u and v are the right hand sides of equations (2) and (3) respectively, and all variables ξ_r $r=1,\ldots,n$ can take two values: 0 and 1. We can then write for the problem $P_i(u)$:

$$g^i_r(u,v) = \min \left[g^i_{r-1}(u,v),\ c_r + g^i_{r-1}(u-c_r,\ v-a_{ir}) \right] \qquad (5)$$

The value $g^i_r(u,v)$ is the minimum value of the objective function w_i given that only the first r variables can be chosen the remaining ones $\xi_{r+1}, \ldots, \xi_n$ being set to 0. Equation (5) can be written for any $0 \leq u \leq U$ and $V''_i \leq v \leq V'_i$ where U is a known upper bound on the value of z^* ($\sum_{j=1}^{n} c_j$ could be used if no such bound is available) V'_i and V''_i are upper and lower bounds on the value of the right hand side of constraint (3) during the computations. These bounds can be set as follows:

Value of V'_i

If all the coefficients of constraint i, a_{ij}, are positive then

$V_i' = b_i$ and $V_i'' = 0$. But if some a_{ij} are negative the term $g_{r-1}^{i}(u - c_r, v - a_{ir})$ in equation (5) might have as second argument a number that exceeds b_i e.g. for $v=b_i$ and $a_{ir} < 0$, $(v-a_{ir}) > b_i$. Thus V_i' (i.e. the largest value of the rhs of constraint i can be taken as

$$V_i' = \min \left[\sum_{j=1}^{n} a_{ij}^{+} , \; b_i + \left| \sum_{j=1}^{n} a_{ij}^{-} \right| \right] \qquad (6)$$

where b_i is the right hand side of constraint i and a_{ij}^{+}, a_{ij}^{-} are the positive and negative coefficients of constraint i respectively.

The first term in equation (6) signifies the maximum value of v which is possible (i.e. by the sum of its positive coefficients) while the second term signifies the maximum value of v ever required, i.e. if all variables which have negative coefficients a_{ij}^{-} are set to 1, $(b_i + \sum_{j=1}^{n} |a_{ij}^{-}|)$ will be the maximum value of v ever required. Whichever of the two terms of equation (6) is the minimum, it can be taken as the upper bound of v, V_i'.

Value of V_i''

If only one problem P_i had to be solved V_i'' could be taken equal to 0, if the variables were ordered in such a way so that those variables with negative coefficients had the highest indices. However, k problems P_i will be solved here and variables having negative coefficients in some constraints might have positive coefficients in other constraints and vice-versa. Thus, V_i'' is taken as:

$$V_i'' = \max \left[\sum_{j=1}^{n} a_{ij}^{-} , \; b_i - \sum_{j=1}^{n} a_{ij}^{+} \right] \qquad (7)$$

Thus, the DP tableau for the problem P_i will be of dimensions

U x V ($V = V_i' + |V_i''|$). The functions $g_r^i(u,v)$ can be calculated iteratively from equation (5) starting with arbitrarily large entries in the initial tableau $[g_o^i(u,v)]$ until the final tableau $[g_n^i(u,v)]$ is calculated. Obviously in order to find a tableau $[g_r^i(u,v)]$ only the previous tableau $[g_{r-1}^i(u,v)]$ needs to be stored. It is also obvious that $g_r^i(u,v) = 0$ if $u,v \leq 0$ for any r as all cost coefficients c_j have been assumed non-negative. Once the final tableau $[g_n^i(u,v)]$ is derived all values of the objective function w_i of problem P_i for any value of the rhs of equation (2) can be read off this tableau directly as $w_i = g_n^i(u,b_i)$.

The DP Tableau

The DP tableau is constructed as follows:

(1) Initialize $g_o(u,v) = \infty$ for all $u \neq 0$ and $v > 0$ and

$g_o(0,v) = 0$ for all $v \leq 0$.

Let $j = 0$

(2) $j = j + 1$

Set $u = 0$ and $v = V_i''$

(3) Compute $D = c_j + g_{j-1}(\underbrace{u - c_j}_{u'}, \underbrace{v - a_{ij}}_{v'})$

If $u - c_j = u' < 0$ set $u' = 0$ and if $v' = v - a_{ij} < V_i''$

set $v' = V_i''$.

If $D < g_{j-1}(u,v)$ set $g_j(u,v) = D$

If $D = g_{j-1}(u,v)$ set $g_j(u,v) = g_{j-1}(u,v)$

If $D > g_{j-1}(u,v)$ set $g_j(u,v) = g_{j-1}(u,v)$.

(4) $v = v + 1$ if $v > \min\left[\sum_{\ell=1}^{j} a^{+}_{i\ell}\ ,\ V'_i\right]$ go to step 5

Otherwise go to step 3.

(5) $u = u + 1$ if $u > \min\left[\sum_{\ell=1}^{j} c_\ell\ ,\ U\right]$ go to step 6.

Otherwise go to step 3.

(6) If $j<n$ go to step 2.

Otherwise, terminate. All the values $g_n(u,v)$ have been computed. Go to step 7.

(7) Keep that column of the tableau $\left[g^i_n(u,v)\right]$ that corresponds to $v = b_i$ i.e. keep those values $g^i_n(u,v)$ for $0 \leq u \leq U$ and $v = b_i$.

It should be noted that the solution of problem $P_i(u)$ by dynamic programming rather than by any other technique, is not accidental but is dictated by the fact that the solution of subsequent problems $P_i(u)$ with different values of the right hand side of constraint (2) becomes a trivial matter. This is a most important property for such an iterative method and one which is not shared by other techniques (such as tree-search) available for solving problem $P_i(u)$.

Since (with all data assumed integer) the minimum difference between z^{p+1} and z^p is 1 and since we can start with a value $z^0 = \lfloor z(LP) \rfloor$ there is an upper bound on the number of iterations given by $U - \lfloor z(LP) \rfloor$ where $\lfloor z(LP) \rfloor$ is the smallest integer $\geq z(LP)$.

3.2.4. The generation of the feasible solutions to the k constraints

In the process of constructing the final dynamic programming tableau $\left[g_n^i(u,v)\right]$ of a problem $P_i(u)$ another tableau T^i of the same dimensions is constructed in parallel. This tableau will be used for backtracking in order to generate all the solutions corresponding to the values $g_n^i(u,v)$. If the optimal solutions of a problem $P_i(u)$ have to be found, when the right hand sides of constraints (2) and (3) are u and v respectively, the backtracking will start from the element $t_n^i(u,v)$ - which in general will be a set of indices $\left\{j^1,\ldots,j^s\right\}$. These are the largest indices of variables which take the value 1 in the optimal solution(s). Thus, if $t_n^i(u,v) = \left\{q_1,\ldots,q_s\right\}$ then there is at least one optimal solution to problem $P^i(u,v)$ which has $x_{q_1} = 1$ and for all $q > q_1$ has $x_q = 0$, at least one optimal solution which has $x_{q_2} = 1$ and for all $q > q_2$ has $x_q = 0$, etc. All the elements of the tableau T^i are initialized to $\emptyset$: $t_o^i(u,v) = \emptyset$ for all u and v. At the r^{th} stage, the set of indices corresponding to element $t_r^i(u,v)$ of the tableau T^i for problem $P_i(u)$ is updated as follows:

(i) if the first term in the square brackets of equation (5) is smallest, the entries in $t_r^i(u,v)$ remain unchanged.

(ii) if the second term is the smallest, the entries in $t_r^i(u,v)$ are replaced by $\{r\}$, i.e. the r^{th} bit of the computer word corresponding to $t_r^i(u,v)$ is set to one and the remaining bits are set to 0.

(iii) if the two terms of equation (5) are equal then r is sinply

added to the other indices of $t_r^i(u,v)$, i.e. the r^{th} bit of the computer word $t_r^i(u,v)$ is set to 1.

Thus, the entries in $t_n^i(u,v), \{q_1, q_2, \ldots q_s\}$ imply that at stage q_1 the optimal value $g_n^i(u,v)$ was achieved ($g_n^i(u,v) = g_{q_1}^i(u,v)$) by setting ξq_1 to 1 and setting to 0 all variables with indices $q < q_1$ contained in $t_{q_1-1}^i(u,v)$. At the next stage q_2, the same value $g_n^i(u,v)$ can be retained by setting ξq_2 to 1 ($g_{q_1}^i(u,v) = g_{q_2}^i(u,v) = g_n^i(u,v)$). The same is true for all subsequent stages $q_3, \ldots q_s$. Thus $g_{q_1}^i(u,v) = g_{q_2}^i(u,v) = \ldots = g_{q_s}^i(u,v) = g_n^i(u,v)$ Consequently, when in the backtracking process a set $\{q_1, q_2, \ldots q_s\} = t_n^i(u,v)$ is encountered, this would mean that any of the variables $\xi q_1, \xi q_2, \ldots, \xi q_s$ can be the variable with the highest index in some of the optimal solutions to problem $P_i(u)$ when the right hand sides of constraints (2) and (3) are u and v respectively.

During the p^{th} iteration - (i.e. in going from a solution X^p to another X^{p+1}) - let the stage be reached where all objective functions of the k problems $P_i(z^p)$ take the same value w^p, and consider the set of indices Q_1 defined by:

$$Q_1 = t_n^1(w^p, b_1) \cap t_n^2(w^p, b_2) \cap \ldots \cap t_n^k(w^p, b_k) \qquad (8)$$

If Q_1 is non-empty, it will contain a set of indices $Q_1 = \{j_1, j_2, \ldots, j_s\}$ which will correspond to the variables $\xi_{j_1}, \xi_{j_2}, \ldots, \xi_{j_s}$. These variables when set to 1 will form the first level of the tree, as shown in Fig. 1, which tree will eventually generate all common solutions (of value w^p), to the k constraints. If the set Q_1 is empty, then no common solution of cost w^p exists, so the

development of the solution tree discontinues and the rhs of constraint (2) can be replaced by w^{p+1} etc. as described in section 3.2.2.

If $Q_1 \neq \emptyset$ then the sets $Q_2(j_\ell)$ of the second level of the solution tree being generated can be calculated, for every $j_\ell \in Q_1$, as follows:

Let $Q_2'(j_\ell) = t_n^1(w^p - c_{j_\ell}, b_1 - a_{1j_\ell}) \cap t_n^2(w^p - c_{j_\ell}, b_2 - a_{2j_\ell})$

$$\cap \ldots, \cap\, t_n^k(w^p - c_{j_\ell}, b_k - a_{kj_\ell}) \qquad (9)$$

Then, $Q_2(j_\ell) = \left\{ j'_\ell / j'_\ell \in Q_2'(j_\ell) \text{ and } j'_\ell < j_\ell \right\}$ (10)

i.e. elements $j'_\ell \in Q_2'(j_\ell)$ where $j'_\ell \geq j_\ell$ are not included in $Q_2(j_\ell)$. The elimination of these variable $j'_\ell \geq j_\ell$ is possible because of the (arbitrary) way that the variables have been numbered so that in deriving the final dynamic programming tableau $\left[g_n^i(u, v)\right]$ variable ξ_h is considered after variable ξ_j if $h > j$. Therefore, if j_ℓ is the variable set to one at the first backtracking level only variables $j'_\ell < j_\ell$ need be considered for the second level.

If any $Q_2(j_\ell)$ is empty, further development of the corresponding tree-branch stops and the development continues to the third level from those branches with $Q_2(j_\ell) \neq \emptyset$ (see Fig. 1). Thus, the development of the tree proceeds until:

either; all branching has stopped, in which case the k problems $P_i(z^p)$ have no common solution of value w^p and the method continues as described in section 3.2.2.

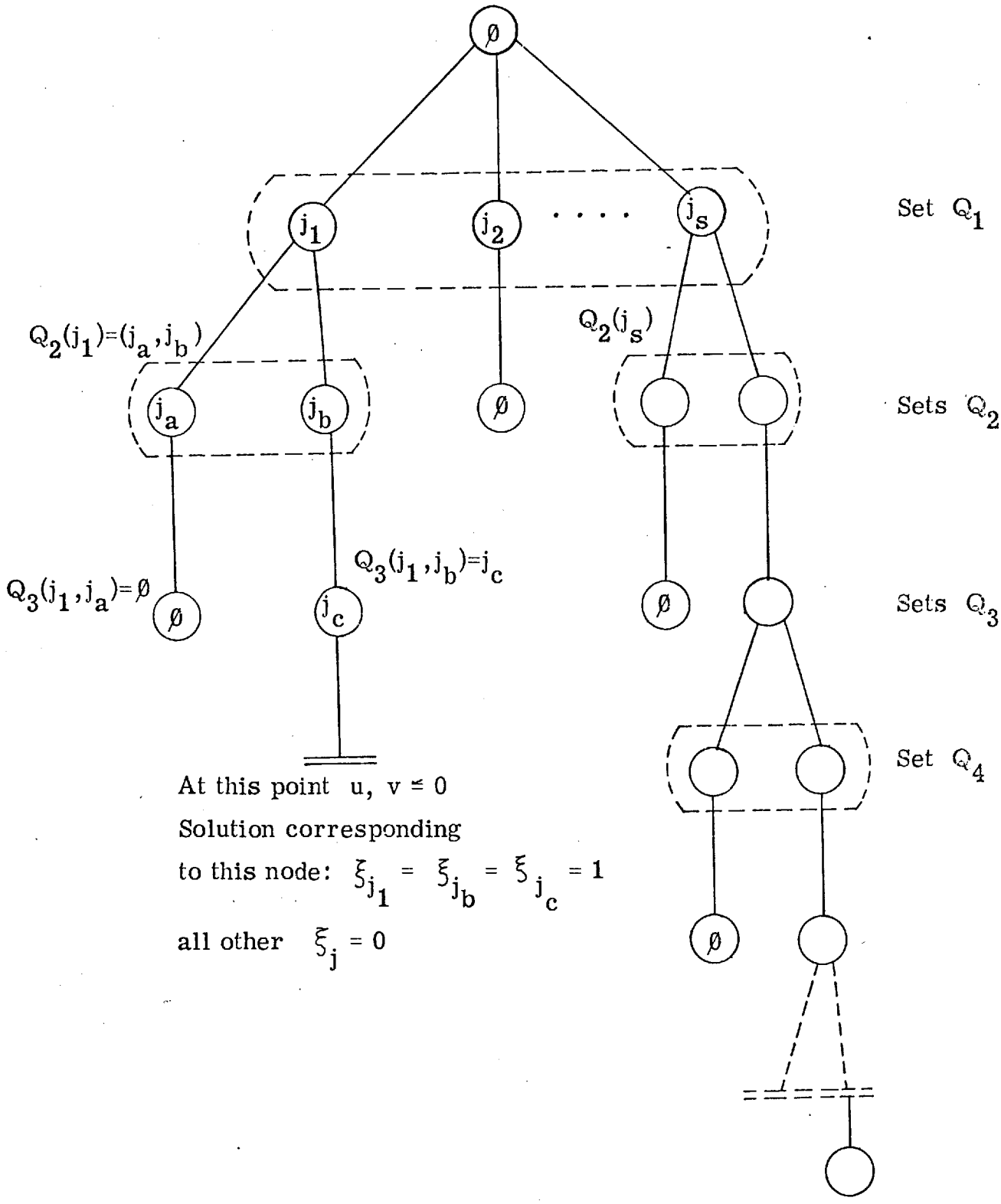


Fig. 1 THE SOLUTION TREE

or: some branches have reached the stage f at which

$Q_f = t_n^1(u,v_1) \cap t_n^2(u,v_2) \cap \ldots \cap t_n^k(u,v_k)$

where u and all $v_i \leq 0$.

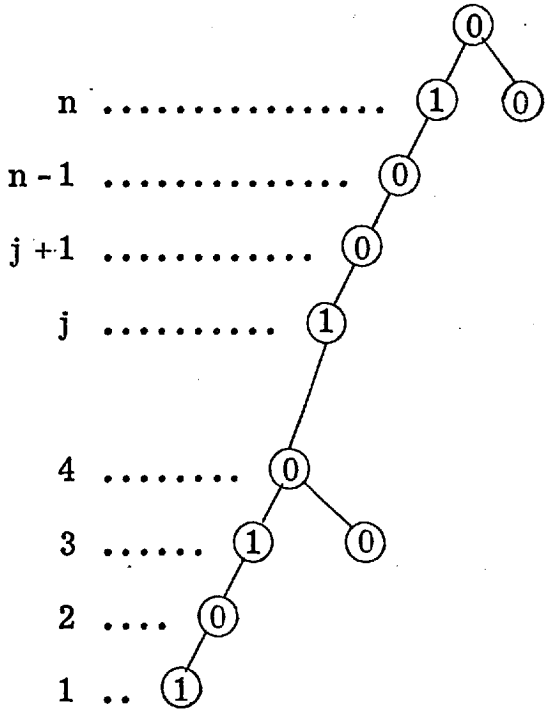
In this case there exist common solutions X^{p+1} with cost $z^{p+1} = w^p$. These solutions are then tested for feasibility to the initial problem P and if any of them is feasible this is the optimum, if not the rhs of constraint (2) becomes $z^{p+1} + 1$ and so on.

The backtracking procedure that was just described corresponds to a "breadth-first" generation of the solution-tree. The alternative way which was in fact used here is a "depth-first" generation of the tree, i.e. it produces one solution at a time and tests each one for feasibility before the next solution is produced. Thus, a vector E of size n is introduced whose elements can take the values 0,1,-1. This vector will be used for generating the whole solution tree.

At the start of the backtracking all elements E(j), j=1,...,n are set to 0. If during the backtracking process any of the sets Q_i contains more than one element i.e. $Q_i = \{j_1, j_2, \ldots, j_s\}$ then the largest index j_s is set of -1 ($E(j_s) = -1$) where the minus sign signifies that Q_i has more than one elements. If $Q_i = \{j_s\}$ then $E(j_s) = 1$. For example, let a common solution to the k constraints of an m-constraint n-variable problem, of value w^p, be expressed by the following state of the vector E:

$$\underset{1}{[1}\ \underset{2}{0}\ \underset{3}{-1}\ 0 \ldots 0\ \underset{j}{+1}\ 0 \ldots 0\ \underset{n}{-1}] \quad (x_1 = x_3 = x_j = x_n = 1 \text{ all the rest} = 0)$$

We want to find all solutions of value w^p and the existence of minus signs in the vector E indicates that other solutions of value w^p might exist. The above state of the vector E represents the non-developed tree:



To find all common solutions to the k constraints of value w^p the following steps have to be followed:

1. Locate the leftmost element of value -1, $E(\ell)$
2. Set $E(\ell)$ and all $E(j)$ $j < \ell$ equal to 0.
   Leave all $E(j)$ $j > \ell$ at their current values.
3. Start the backtracking from that set Q_i which gave rise to the index ℓ, by setting to 1 the next lower index to ℓ, $\ell' \in Q_i$
   Continue backtracking until either a solution is found or a set Q is encountered which is empty. If a solution is found this has to be tested for feasibility to P. If it is infeasible or if an empty Q set is encountered go to step 4.

4. If there exists an element of E at -1, go to step 1.

   Otherwise, if all elements of E are non-negative all (if any). solutions of value W^p have been found. Proceed to the next iteration.

3.3. Description of the basic algorithm

Let the initial lower bound of the objective function of problem P, $\triangle$, be 0.

Step 1. Construct the k dynamic programming tableaux $\left[g_n^i(u,v)\right]$ $i = 1,\ldots,k$ and the k value tableaux T^i.

Keep those values $g_n^i(u,b_i)$ of the k tableaux $\left[g_n^i(u,v)\right]$ that correspond to $u = 1,\ldots,U$ and $v = b_i$ for all $i = 1,\ldots,k$ and discard all other values $g_n^i(u,v)$.

Step 2. Find $w_i = g_n^i(\triangle, b_i)$ for all $i = 1,\ldots,k$.

If all w_i are equal go to step 4.

If not, go to step 3.

Step 3. Set $\triangle = \max\limits_{1 \le i \le k} \{ w_i \}$

Go to step 2.

Step 4. Let the common value of the w_i $i = 1,\ldots,k$, be w^p.

Set $E(j)$ $j = 1,\ldots,n$ equal to 0.

Start the backtracking and find if any common solutions to the k constraints, of value w^p, exist i.e. Find

$$Q_1 = \bigcap_{i=1,\ldots,k} t_n^i(w^p, b_i) = \{ j_1, j_2, \ldots, j_s \}$$

$$Q_2(j_s) = \bigcap_{i=1,\ldots,k} t_n^i(w^p - c_{j_s}, b_i - a_{ij_s}) = \{ j_a, j_b, \ldots, j_y \}$$

$$\vdots$$

e.t.c.

$$\vdots$$

$$\vdots$$

4(a) $Q_i(j_s, j_y, \ldots j_h) =$

$$= \bigcap_{i=1,\ldots k} t_n^i (w^p - c_{j_s} - c_{j_y} - \ldots - c_{j_h}\ ,\ b_i - a_{ij_s} - a_{ij_y} - \ldots - a_{ij_h})$$

If in the above backtracking process a set Q_i contains more than one element its rightmost element r is considered first and $E(r) = -1$ e.g. for Q_1, $E(j_s) = -1$

In this case, the k pairs of values (u,v) that gave rise to the set Q_i containing more than one element, are kept for future reference. If a set Q_i contains just one element j then $E(j) = 1$. If an empty set Q_i is encountered, go to step 6. If a common solution is found (which will be signified by $Q_f = \bigcap_{i=1,\ldots k} t_n^i(u, v_i)$ u and all $v_i \leq 0$), go to step 5.

Step 5. Test this common solution for feasibility to the remaining m-k constraints. If this solution happens to be feasible this is the optimum*, go to step 7. If not go to step 6.

Step 6. Locate the leftmost element ℓ, of the vector E that has the value -1, $E(\ell) = -1$. Set all $E(j) = 0$ $j \leq \ell$

* If all optima need to be found, instead of going to step 7 we could proceed to step 6 and find all common solutions to the k constraints of value z^*. Those that are feasible to the remaining m-k constraints are optimal solutions.

Leave all $E(j)_{j>\ell}$ unchanged.

Start backtracking from those values of u and $v_1,\ldots v_k$ that produced that set Q which contained the index ℓ.

Go to step 4(a). If all E(j)'s j = 1,...,n are non-negative, all common solutions to the k constraints of value w^p (if any) have been found and none of them is feasible to the complete problem P. Set $\triangle = w^p + 1$ and proceed to the next iteration, go to step 2.

Step 7. Stop. The current state of the vector E is the optimal solution to problem P, where $E(j) = \pm 1$ signifies $x_j = 1$ and $E(j) = 0$ implies $x_j = 0$.

3.4. Improvements to the basic algorithm

The underlying objective in this algorithm is to have at each iteration a "thin" solution tree to speed up the iterations. We desire a small number of common sulutions to the k chosen constraints (a "thin" tree) and this obviously depends on the value of k. As k increases the number of solutions that have to be tested for feasibility to the remaining m-k constraints, obviously decreases. We also desire that the algorithm should detect the non-existence of common solutions of value w^p or the non-usefullness of developing some branches early on during the backtracking. We'll see that the ordering of the variables could help in detecting the eventual termination of a branch early on. A analysis about the value of k, which k

constraints to choose once the value of k has been decided and a way of ordering the variables in a beneficial manner in such a way as to keep the solution trees "thin" will be presented in the next chapter.

The use of LP

Here, we will present another method of "early detection" of the termination of some branches, which will involve the use of Linear Programming. After having fixed a set of variables J to the values 0 or 1 the following LP problem could be solved whose optimum will determine if it is worth developing further the branch in question:

$$\left.\begin{array}{lll} \min \Psi = \sum_{j \notin J} c_j x_j & & \\ & & \\ \text{s.t.} \quad \sum_{j \notin J} c_j x_j \leq w^p - \sum_{j \in J} c_j x_j & & R \\ & & \\ \sum_{j \notin J} a_{ij} x_j \geq b_i - \sum_{j \in J} a_{ij} x_j & i=1,\ldots,m & \\ & & \\ 0 \leq x_j \leq 1 & j=1,\ldots,n & \end{array}\right)$$

where w^p is the current trial value of the objective function. If the problem is infeasible, (or if the LP optimum Ψ is greater than $(w^p - \sum_{j \in J} c_j x_j)$) then it is not worth developing the branch in question further. The above LP problem could be solved at steps of say 10 variables, i.e. every time we fix an additional set of 10 variables.

In addition, a cutting plane can be introduced to make the use of LP more efficient. It is known, that the cutting plane approach can be very effective in the initial cuts while it slows down later. This fact can be exploited in a beneficial manner if after a certain number of cuts has been introduced, the solving of problem R is stopped. This, obviously, can be done here as the aim is not an integer solution to problem R but a good bound arising from the value of the objective function of R.

Morever, one can make an analysis of the logical structure of the original problem and "intersect" constraints which are derived from this analysis, instead of "intersecting" original constraints. In Chapter 5, where the concept of a "reduced set" is defined, a way by which logical analysis (in this case the "reduced sets") can be used in this algorithm is presented.

The algorithm could be greatly improved if use of surrogare constraints is made. For example, one could group the constraints of the original problem and develop a surrogate constraint corresponding to each group. Then, instead of "intersecting" the original constraints one would intersect the derived surrogate constraints where a DP tableau would correspond to each surrogate constraint. If, in addition, the grouping of the original constraints is done in an intelligent manner the use of surrogate constraints can be found very effective.

3.5.1. AN EXAMPLE

Consider the following 6-variable, 4-constraint problem:

$$
\left.\begin{array}{lrcl}
\min & z = & 2x_1 + 4x_2 + 3x_3 + x_4 + 2x_5 + x_6 & \\
\text{s.t.} & 4x_1 + 3x_2 + 2x_3 + 2x_4 + 2x_5 - x_6 & \geq 7 & \\
& 7x_1 + 2x_2 - 3x_3 - x_4 - x_5 - x_6 & \geq 4 & \\
& 3x_1 - 3x_2 + 3x_3 - 2x_4 - x_5 + 3x_6 & \geq 3 & \\
& x_1 + 2x_2 + 3x_3 + 3x_4 - x_5 + 3x_6 & \geq 3 &
\end{array}\right\} \quad P
$$

Say that k is chosen to be 2 i.e. two problems $P_i(u)$ will be solved and that these involve the first and second constraint respectively.

$$
\left.\begin{array}{lrcl}
\min & w_1 = & 2x_1 + 4x_2 + 3x_3 + x_4 + 2x_5 + x_6 & \\
\text{s.t.} & 2x_1 + 4x_2 + 3x_3 + x_4 + 2x_5 + x_6 & \geq u & \\
& 4x_1 + 3x_2 + 2x_3 + 2x_4 + 2x_5 - x_6 & \geq 7 &
\end{array}\right\} \quad P_1(u)
$$

$$
\left.\begin{array}{lrcl}
\min & w_2 = & 2x_1 + 4x_2 + 3x_3 + x_4 + 2x_5 + x_6 & \\
\text{s.t.} & 2x_1 + 4x_2 + 3x_3 + x_4 + 2x_5 + x_6 & \geq u & \\
& 7x_1 + 2x_2 - 3x_3 - x_4 - x_5 - x_6 & \geq 4 &
\end{array}\right\} \quad P_2(u)
$$

3.5.2. Problem $P_1(u)$

The dimensions of the dynamic programming tableaux will be taken as $U = 8$, $V = b_1 + \left|\sum a_{1j}^-\right| = 7 + 1 = 8$. The final tableau $\left[g_6^1(u,v)\right]$ will be constructed sequentially as follows:

The first tableau $\left[g_0^1(u,v)\right]$ is initialized so that,

$$g_0^1(u,v) = \infty \quad \text{for all } u,\ v > 0$$

and $$g_0^1(u,v) = 0 \quad \text{for all } u \text{ and } v \leq 0$$

$$t_0^1(u,v) = \emptyset \quad \text{for all } u,v.$$

The tableau $\left[g_1^1(u,v)\right]$ is constructed using equation (5).

For example:

$$g_1^1(2,3) = \min\left[\infty\ ,\ c_1 + g_0^1(2\text{-}2, 3\text{-}4)\right] = \min\left[\infty, 2\right] = 2$$

and $t_1^1(2,3) = \{1\}$ according to section 3.2.4.(ii).

Hence, the tableau $\left[g_1^1(u,v)\right]$ becomes :

Table 1

$\left[g_1^1(u,v)\right] =$ (u ↓, v →)

u \ v	0	1	2	3	4	5	6	7	8
0	0	2	2	2	2	∞			∞
1	2	2	2	2	2				
2	2	2	2	2	2				
3	∞					∞			
4									
5									
6									
7									
8	∞								∞

Similary $\left[g_2^1 (u,v) \right]$ is derived from $\left[g_1^1 (u,v) \right]$ and so on as shown in tables 2 to 6.

Table 2

$\left[g_2^1 (u,v) \right] =$

	0	1	2	3	4	5	6	7	8
0	0	2	2	2	2	6	6	6	∞
1	2	2	2	2	2	6	6	6	1
2	2	2	2	2	2	6	6	6	1
3	4	4	4	4	6	6	6	6	1
4	4	4	4	4	6	6	6	6	1
5	6	6	6	6	6	6	6	6	1
6	6	6	6	6	6	6	6	6	1
7	∞	-	-	-	-	-	-	-	1
8	∞	-	-	-	-	-	-	-	1

Table 3

$\left[g_3^1 (u,v) \right] =$

	0	1	2	3	4	5	6	7	8
0	0	2	2	2	2	5	5	6	9
1	2	2	2	2	2	5	5	6	9
2	2	2	2	2	2	5	5	6	9
3	3	3	3	4	5	5	5	6	9
4	4	4	4	4	5	5	5	6	9
5	5	5	5	5	5	5	5	6	9
6	6	6	6	6	6	6	6	6	9
7	7	7	7	7	7	7	9	9	9
8	9	9	9	9	9	9	9	9	9

Table 4

$\left[g_4^1(u,v)\right]$ =

	0	1	2	3	4	5	6	7	8
0	0	1	1	2	2	3	3	6	6
1	1	1	1	2	2	3	3	6	6
2	2	2	2	2	2	3	3	6	6
3	3	3	3	3	3	3	3	6	6
4	4	4	4	4	4	5	5	6	6
5	5	5	5	5	5	5	5	6	6
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	9

Table 5

$\left[g_5^1(u,v)\right]$ =

	0	1	2	3	4	5	6	7	8
0	0	1	1	2	2	3	3	5	5
1	1	1	1	2	2	3	3	5	5
2	2	2	2	2	2	3	3	5	5
3	3	3	3	3	3	3	3	5	5
4	4	4	4	4	4	4	4	5	5
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8

Table 6

$\left[g_6^1(u,v)\right] =$

	0	1	2	3	4	5	6	7	8
0	0	1	1	2	2	3	3	5	5
1	1	1	1	2	2	3	3	5	5
2	2	2	2	2	2	3	3	5	5
3	3	3	3	3	3	3	3	5	5
4	4	4	4	4	4	4	4	5	5
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8

The solution tableau $\left[t_6^1(u,v)\right]$ is derived during the computations and is as shown in Table 7. A crossing out of an element r of this tableau signifies that at a later stage $r' > r$ a better value $g_{r'}^1(u,v) < g_r^1(u,v)$ was achieved. For example, for $u = 3$, $v = 4$ the following changes of element (3,4) of the final T^1 tableau took place:

$g_1^1(3,4) = \quad t_1^1(3,4) = \emptyset$

$g_2^1(3,4) = 6 \quad t_2^1(3,4) = \{2\}$

$g_3^1(3,4) = 5 < 6 \quad t_3^1(3,4) = \{3\}$ and 2 is removed

$g_4^1(3,4) = 3 < 5 \quad t_4^1(3,4) = \{4\}$ and 3 is removed

$g_5^1(3,4) = 3 \quad t_5^1(3,4) = \{4,5\}$

$g_6^1(3,4) = 3 \quad t_6^1(3,4) = \{4,5\}$

Table 7 Tableau T^1

v ⟶

u ↓

	0	1	2	3	4	5	6	7	8
		1 4	1 4	1	1	2 3 4	2 3 4	2,4 5	3 4 5
1	1 4	1 4	1 4	1	1	2 3 4	2 3 4	2,4 5	3 4 5
2	1,5,6	1,5,6	1,5	1	1	2 3 4	2 3 4	2,4 5	3 4 5
3	2 3,4,5,6	2 3,4,5,6	2 3,4,5,6	2 4,5,6	2 3 4,5	2 3 4	2 3 4	2,4 5	3 4 5
4	2,4,5,6	2,4,5,6	2,4,5,6	2,4,5,6	2 3 4,5,6	2 3 4 5,6	2,3 5	2,4 5	3 4 5
5	2 3,4,5,6	2 3,4,5,6	2 3,4,5,6	2 3,4,5,6	2 3,4,5,6	2 3,4,5,6	2 3,5	2,4 5	3 4 5
6	2,4,5,6	2,4,5,6	2,4,5,6	2,4,5,6	2,4,5,6	2,4,5,6	2,5,6	2,4,6	3 4
7	3,4,5,6	3,4,5,6	3,4,5,6	3,4,5,6	3,4,5,6	3,4,5,6	3,4,5,6	3,4,5,6	3 4,5
8	3 4,5,6	3 4,5,6	3 4,5,6	3 4,5,6	3 4,5,6	3 4,5,6	3 4,5,6	3,4,5,6	3 5

3.5.3. Problem $P_2(u)$

The final dynamic programming tableau $\left[g_6^2(u,v)\right]$ for problem P_2 is given in Table 8. (Dimensions $U = 8$ and $V = 9$ are taken and are sufficient).

Table 8

$v \longrightarrow$

$\left[g_6^2(u,v)\right] =$ ($u \downarrow$)

u \ v	0	1	2	3	4	5	6	7	8	9
0	0	2	2	2	2	2	2	2	6	6
1	2	2	2	2	2	2	2	2	6	6
2	2	2	2	2	2	2	2	2	6	6
3	3	3	3	3	3	3	3	6	6	6
4	4	4	4	4	4	4	4	6	6	6
5	5	5	5	5	5	5	6	6	6	6
6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	∞
8	8	8	8	8	8	8	8	8	8	∞

The tableau T^2 is shown in Table 9.

Table 9 Tableau T^2

u \ v	0	1	2	3	4	5	6	7	8	9
0	∅	1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1	1	2	2
2	1	1	1	1	1	1	1	1	2	2
3	~~2 3~~ 4,6	~~2 3~~ 4,6	~~2,3~~,4,6	~~2 3~~ 4,6	~~2 3~~ 4,6	~~2~~ 4,6	~~2~~ 4,6	2	2	2
4	2,5,6	2,5,6	2,5,6	~~2 3~~ 5,6	~~2 3~~ 5,6	~~2~~ 5,6	~~2~~ 5	2	2	2
5	~~2~~ 3,4,5,6	~~2~~ 3,4,5,6	~~2~~ 3,5,6	~~2~~ 3,5,6	~~2~~ 3,5,6	~~2~~ 5,6	2	2	2	2
6	2,5,6	2,5,6	2,6	2,6	2,6	2	2	2	2	2
7	~~3~~ 4,5,6	~~3~~ 4,6	~~3~~ 4,6	~~3~~ 4,6	~~3~~ 4,6	~~3~~ 4,6	~~3~~,4,6	4,6	4,6	∅
8	~~3~~,5	~~3~~ 5,6	~~3~~ 5,6	~~3~~ 5,6	~~3~~ 5,6	~~3~~ 5,6	~~3~~ 5,6	5,6	5	∅

3.5.4. The iterations

Starting with a lower bound z^0 of value 0, we observe that $w_1 = g_6^1(0,7) = 5 \neq w_2 = g_6^2(0,4) = 2$. So, no feasible solution of cost< 5 can exist.

At the next step, $g_6^1(5,7) = g_6^2(5,4) = 5$ and so, there exists the possibility of having a common solution of value 5 to the first and second constraints of problem P.

(i) First iteration

From equation (8) we can write:

$$\Omega_1 = t_6^1(5,7) \cap t_6^2(5,4) = \{5\} \cap \{3,5,6\} = \{5\}$$

and from equations (9) and (10) we get:

$$\Omega_2'(\xi_5) = t_6^1(5-c_5, 7-a_{15}) \cap t_6^2(5-c_5, 4-a_{25}) =$$

$$= t_6^1(3,5) \cap t_6^2(3,5) = \{4\} \cap \{4,6\} = \{4\}$$

and $\Omega_2(\xi_5) = \{4\}$

Similarly: (see Fig. 2(a))

$$\Omega_3'(\xi_5, \xi_4) = t_6^1(3-c_4, 5-a_{14}) \cap t_6^2(3-c_4, 5-a_{24}) =$$

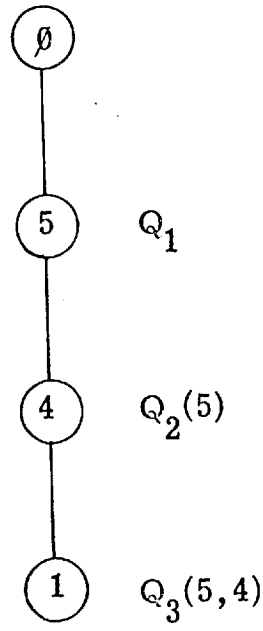
$$= t_6^1(2,3) \cap t_6^2(2,6) = \{1\} \cap \{1\} = \{1\}$$

and $\Omega_3(\xi_5, \xi_4) = \{1\}$

$$\Omega_4'(\xi_5, \xi_4, \xi_1) = t_6^1(2-c_1, 3-a_{11}) \cap t_6^2(2-c_1, 6-a_{21}) =$$

$$= t_6^1(0,-1) \cap t_6^2(0,-1)$$

So, the stage has been reached at which all u and v are



1st iteration

solution: $\xi_5 = \xi_4 = \xi_1 = 1$

$\xi_2 = \xi_3 = \xi_6 = 0$

Fig. 2(a)

non-positive and the branch represents the solution $\xi_1 = \xi_4 = \xi_5 = 1$ $\xi_2 = \xi_3 = \xi_6 = 0$ as shown diagrammatically in Fig. 2(a). This solution of value $z^p = 5$ is infeasible to the third constraint of problem P, so we proceed to the next iteration. The rhs's of the first constraint of both problems are replaced by $z^p + 1 = 6$. (See section 3.2.2.)

(i.i) Second iteration

It is observed that $g_6^1(6,7) = g_6^2(6,4) = 6$. Thus from equation (8):

$$Q_1 = t_6^1(6,7) \cap t_6^2(6,4) = \{2,4,6\} \cap \{2,6\} = \{2,6\}$$

Considering variable ξ_2 first: (See Fig. 2(b).)

$$Q'_2(\xi_2) = t_6^1(\underbrace{6-c_2}_{2}, \underbrace{7-a_{12}}_{4}) \cap t_6^2(\underbrace{6-c_2}_{2}, \underbrace{4-a_{22}}_{2}) = \{1\}$$

and $Q_2(\xi_2) = Q'_2(\xi_2)$.

$$Q'_3(\xi_2, \xi_1) = t_6^1(2-c_1, 4-a_{11}) \cap t_6^2(2-c_1, 2-a_{21}) =$$

$$= t_6^1(0,0) \cap t_6^2(0,-5)$$

and since both u and v are non-positive a common solution has been found.

The solution ($\xi_1 = \xi_2 = 1$ all other $\xi_j = 0$) is infeasible to problem P.

Considering variable 6 next:

$$Q'_2(\xi_6) = t_6^1(\underbrace{6-c_6}_{5}, \underbrace{7-a_{16}}_{8}) \cap t_6^2(\underbrace{6-c_6}_{5}, \underbrace{4-a_{26}}_{5}) = \{5\} \cap \{5,6\} = \{5\}$$

and $Q_2(\xi_6) = Q'_2(\xi_6) = \{5\}$

$Q'_3(\xi_6, \xi_5) = t_6^1(5\text{-}c_5, 8\text{-}a_{15}) \cap t_6^2(5\text{-}c_5, 5\text{-}a_{25}) = \{4\} \cap \{4,6\} = \{4\}$

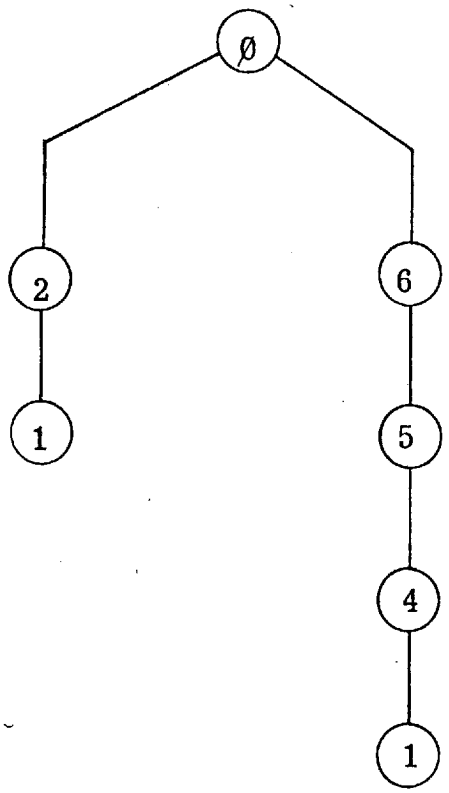
and $Q_3(\xi_6, \xi_5) = Q'_3(\xi_6, \xi_5) = \{4\}$

$Q'_4(\xi_6, \xi_5, \xi_4) = t_6^1(\underbrace{3\text{-}c_4}_{2}, \underbrace{6\text{-}a_{14}}_{4}) \cap t_6^2(\underbrace{3\text{-}c_4}_{2}, \underbrace{6\text{-}a_{24}}_{7}) = \{1\}$

and $Q_4(\xi_6, \xi_5, \xi_4) = Q'_4(\xi_6, \xi_5, \xi_4) = \{1\}$

Continuing to the 5^{th} level it is indicated that a solution has been reached. This solution ($\xi_1 = \xi_4 = \xi_5 = \xi_6 = 1$, $\xi_2 = \xi_3 = 0$) is feasible to problem P and is therefore the optimal solution. The solution-tree of the last iteration is shown in Fig. 2(b).

(It can be observed that in this small example where $Q_1 = \{2, 6\}$ in the second iteration, variable ξ_2 was considered first instead of ξ_6).



2nd iteration

solution: $\xi_1 = \xi_2 = 1$

$\xi_3 = \xi_4 = \xi_5 = \xi_6 = 0$

$\xi_1 = \xi_4 = \xi_5 = \xi_6 = 1$

$\xi_2 = \xi_3 = 0$

Fig. 2(b)

CHAPTER 4

COMPUTATIONAL ASPECTS OF THE SEQUENTIAL APPROACH OF CHAPTER 3

4.1. Introduction

This chapter examines the computational efficiency of the basic algorithm proposed in Chapter 3.

The effects of choosing different values for those parameters of the algorithm which can be fixed arbitrarily are investigated in some detail. These include the choice of the value of k (number of "active" constraints chosen) and which k (out of the total of m constraints) to treat as "active". In addition, although the method has been described earlier assuming a fixed ordering of the variables in constructing the DP tableaux - and hence in developing the tree of common solutions - this ordering is arbitrary and a heuristic rule for a "good" ordering of the variables is given.

The computational storage and time requirements of the basic method (not including the use of LP, cutting planes, generation of logical constraints etc.) are examined and test results for 80 randomly generated problems and for some problems found in the literature are presented. The computational performance of the method is, therefore, demonstrated to be at least as good as that reported for any other existing algorithm, and generally superior for dense problems having "small" coefficients.

4.2. Storage Requirements

The cost coefficients c_j were, without any loss of generality, taken to be nonnegative.

The basic storage requirements of the method are:

(i) Two tables of dimensions U x $\max_{1 \leq i \leq k} \{V_i\}$ - (see section 3.2.3) - used several times - n times for each problem $P_i(u)$, i = 1, ..., k - in order to derive the final tableaux $\left[g_n^i(u,v)\right]$.

(ii) A table of dimensions U x k which is used to store the columns $g_n^i(u,b_i)$, u = 0, ..., U from each final tableau $\left[g_n^i(u,v)\right]$ corresponding to each problem $P_i(u)$, i = 1, ..., k. Note that only this one column for each problem is necessary to perform the iterations described in section 3.2.2.

(iii) k tables T^i each of dimensions U x V_i to perform the back-tracking. Each computer word corresponding to an element of T^i is assumed to store a set of up to d indices where the word length is d bits. If d < n then $\left\lfloor \frac{n}{d} \right\rfloor$. k, instead of just k, tables will be required, where $\lfloor x \rfloor$ is the smallest integer greater than x.

It is quite apparent from what has been said above that the storage requirements of the method can become excessive if the coefficients c_j, a_{ij} or b_i are large. In these circumstances several courses of action are available for reducing these requirements.

(a) The objective function and each constraint can be divided through by the greatest common divisor of their corresponding coefficients.

(b) It is quite likely that after the application of (a) above, the

coefficients of some constraints are on average smaller than those of other constraints. In addition some constraints may be appreciably more sparce than others and hence lead to tables of smaller size. Since, as mentioned later, one is free to choose any k (of the m constraints) in order to apply the method of Ch 3, it would be beneficial - at least from the storage point of view - to choose constraints which lead to tables of small size.

(c) One can use "logical" constraints derived from the structure of the problem where these constraints can have small coefficients and fully express the particular problem. For example, the idea of a reduced set, as described in Chapter 5, can be used for extracting "logical" constraints having coefficients equal to 1. (See section 5.2.4)

(d) Since for a given inequality with 0 - 1 variables there are many other "equivalent" inequalities with exactly the same 0 - 1 feasible solutions, one can find that equivalent inequality with smallest coefficients [45] . In [45] an algorithm which constructs the minimum equivalent inequality is introduced by first determining all the "roofs" and "ceilings" of the given inequality and then solving an associated LP problem. This is one of the most promising courses of action for coefficient reduction.

(e) Finally, by the introduction of surrogate constraints with the appropriate weight coefficients, one can reduce the size of the coefficients of the problem.

4.3. Choice of the k constraints

We will now investigate the effects that the value of k and the choice of which k constraints to use in the method, have on the computing times.

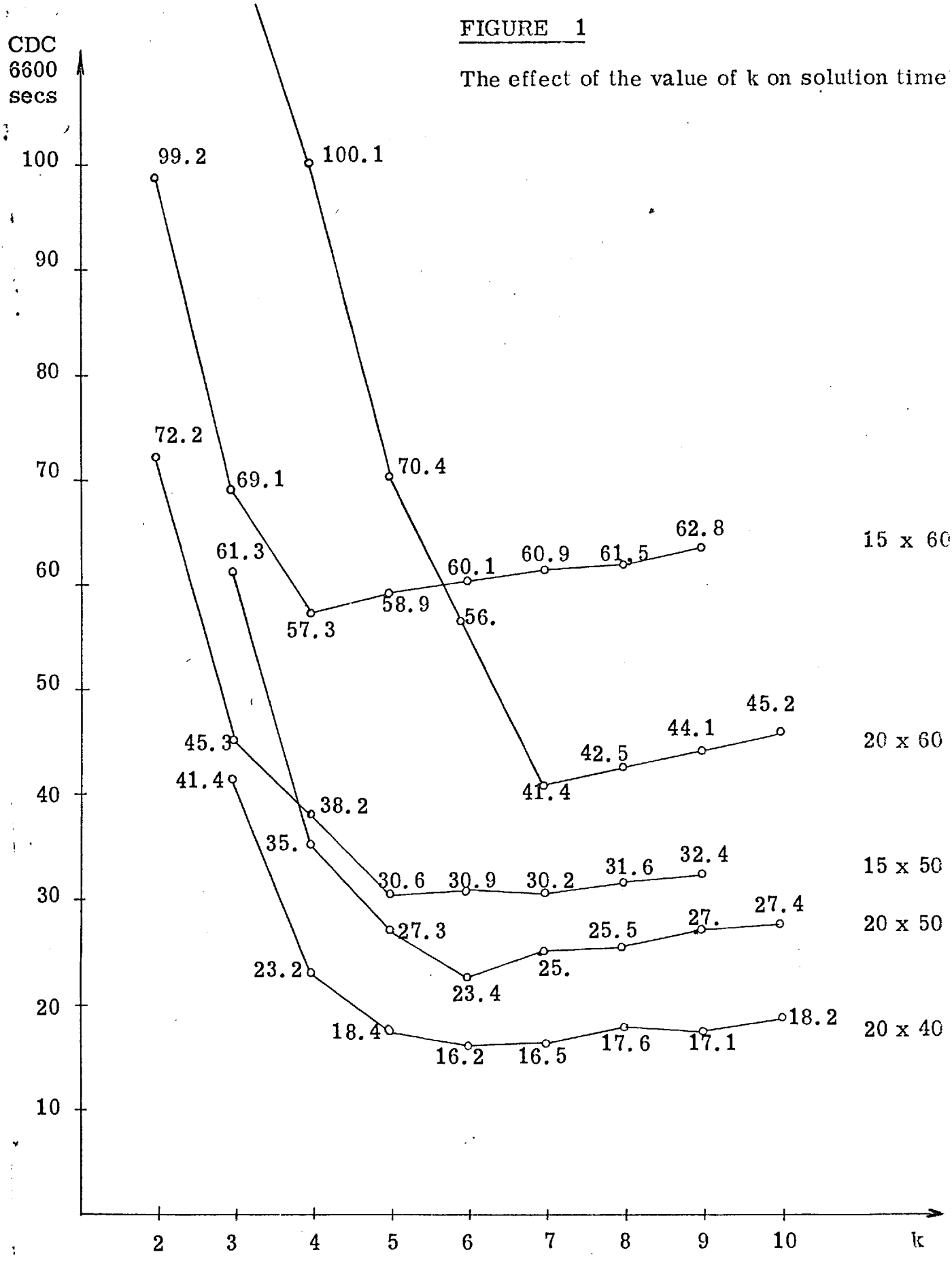
4.3.1. Value of k

The value of k is related to the number of solutions that have to be tested for feasibility to the remaining m - k constraints. When k is small the number of common solutions to the k constraints is large and as k increases the number of common solutions decreases until we reach k = m where the common solutions to the k constraints are the optimal ones only. We have to find an acceptable value of k between the two extremes k = 1 and k = m. At the one end of the range of k, (k = 1), a situation of a combinatorial nature might arise where a very large number of solutions will have to be tested for feasibility to the remaining m - 1 constraints while at the other end, k = m, no testing for feasibility has to take place, at the end of an iteration, but m DP tableaux have to be constructed.

The effect of the value of k on solution times is shown in Fig. 1. From this figure it is seen that the choice of the value of k is important but that it is much better to overestimate rather than

FIGURE 1

The effect of the value of k on solution time



*The heuristic rule for ordering the variables (section 4.4.1.) has not been applied in the above problems.

underestimate the value of k_{min} at which a minimum solution time is achieved. In particular, it is noted that the curves are quite shallow in the range $k_{min} \leq k \leq m$.

Thus, in practice, k should be taken to be quite large (near m), since at worst any additional computing time that may result from the inclusion of $(k - k_{min})$ constraints will be due to the construction of the additional Dynamic Programming tableaux.

4.3.2. Choice of k constraints

It was mentioned earlier, if k is small the number of solutions that have to be tested for feasibility is large and thus the choice of which k constraints to intersect is important. This is so, because the number of 0 - 1 vectors that can satisfy each individual constraint varies, as this number depends on the structure and right hand side of the particular constraint, and thus we should choose those k constraints which can be satisfied by a small number of 0- 1 vectors. It is intuitively obvious that as the value of k increases the importance of which k constraints to choose decreases. Again, at k = 1 it is extremely important to choose the most "tight" constraint while at k = m the problem of choice disappears. However, as shown by the experimental results in Table 1, for relatively large k it is almost immaterial which k constraints are chosen. In this Table, 15 randomly generated problems are each solved twice, once with a set of k constraints and a second time with another set

of k constraints. The first five 20- constraint problems were solved once for each of the two disjoint sets of k = 10 constraints. The rest of the problems were solved once for a given set of k constraints and a second time with another set of k constraints only a third of which were in common.

If, however, due to different reasons (e.g. to reduce storage requirements) a small value of k has to be taken, the following heuristic criterion could help in choosing those constraints that will not yield an exceptionally large number of common solutions:

Classify the m constraints into two categories: category I includes those constraints with $b_i > 0$ and category II those with $b_i \leq 0$.

1. For a constraint i in category I,

$$\text{let: } a_i = \frac{\sum_{j=1}^{n} a_{ij}^{+} - b_i}{b_i}$$

For a constraint i in category II

$$\text{let: } a_i = \frac{1}{r_i}$$

where a_{ij}^{+} are the positive coefficients of constraint i and r_i is the number of a_{ij}'s for which $a_{ij} < b_i \leq 0$.

If category I contains more than k constraints, choose those k with the lowest values of a_i, else choose the remainder from category II again choosing according to the lowest a_i. It is apparent that many heuristic criteria could be proposed. For example other possible criteria are: the value of the right hand sides b_i, the

Table 1

Effect of the choice of k constraints on computing times

Problem No.	N	M	k	Time (CDC 6600 secs) Set 1	Set 2
1	90	20	10	12.4	11.6
2	90	20	10	21.7	41.8
3	90	20	10	53.4	35.8
4	90	20	10	11.9	9.9
5	90	20	10	11.6	17.
6	80	30	18	107.6	115.5
7	80	30	18	23.2	23.3
8	80	30	18	38.	57.2
9	80	30	18	48.5	48.1
10	80	30	18	36.8	27.5
11	90	20	12	17.9	17.1
12	90	20	12	25.6	24.9
13	90	20	12	51.3	51.
14	90	20	12	17.1	16.3
15	90	20	12	16.8	15.7

ratio $\frac{\sum_{j=1}^{n} a_{ij}^{+} - b_i}{\sum_{j=1}^{n} a_{ij}^{-}}$ etc. In the former case those k constraints are chosen which have the largest b_i's while in the latter case those k are chosen which have the smallest ratios of $\frac{\sum_{j=1}^{n} a_{ij}^{+} - b_i}{\sum_{j=1}^{n} a_{ij}^{-}}$.

Another heuristic criterion is the ratio, $\frac{\sum a_{ij}^{+} - b_i}{\sum a_{ij}^{-} + b_i}$ and in this case, the most "tight" constraint is the one having the smallest value for this ratio, so that those k constraints are chosen which correspond to the smallest values of $\frac{\sum a_{ij}^{+} - b_i}{\sum a_{ij}^{-} + b_i}$. All these criteria are based on the intuitive reasoning of which constraints are "tight".

4.4. Ways of limiting the size of the solution tree

A consideration of prime importance is the existence of a small number of solutions that have to be tested to the non-intersected m - k constraints and the main contributing factor was seen to be the value of k. However, even when there exist few common solutions we might have a large solution tree. Its excessive size would be then due to the fact that unnecessary branchings have taken place. Here we will present ways of detecting an infeasibility early on and thus discontinue, early in the search, branches that would eventually terminate at a later stage. One of the main methods of limiting an unnecessarily large size of the solution tree is the ordering of the variables which we discuss in the next section.

4.4.1. Ordering of the variables

Consider the following 2 - constraint, 5 - variable problem:

$$\min \quad z = x_1 + x_2 + x_3 + x_4 + x_5$$

$$\text{s.t.} \quad -3x_1 - 4x_2 + 3x_3 + 2x_4 + 4x_5 \geq 5$$

$$4x_1 + 6x_2 - 3x_3 - 1x_4 + 5x_5 \geq 5$$

Two problems of the $P_i(u)$ type will be solved: $P_1(u)$ and $P_2(u)$.

We observe that for $u = 0$, $w_1 = 2$ and $w_2 = 1$.

Thus $u = \max_{i=1,2} (w_i) = 2$. For $u = 2$, $w_1 = 2$ and $w_2 = 2$.

Consequently, there exists the possibility of having a common solution to both problems $P_1(u)$ and $P_2(u)$.

We have $t_5^1 (2,5) = \{4,5\}$ and $t_5^2 (2,5) = \{2, 4, 5\}$

$Q_1 = t_5^1 (2, 5) \cap t_5^2 (2, 5) = \{4, 5\} \cap \{2, 4, 5\} = \{4, 5\}$

and the solution tree has to be developed at least up to the first level. If, however, we order the variables in the sequence x_5, x_1, x_4, x_2, x_3 we'll have the following problem, after assigning the new indices of the variables:

$$\min \quad z = x_1 + x_2 + x_3 + x_4 + x_5$$

$$\text{s.t.} \quad 4x_1 - 3x_2 + 2x_3 - 4x_4 + 3x_5 \geq 5$$

$$5x_1 + 4x_2 - 1x_3 + 6x_4 - 3x_5 \geq 5$$

For $u = 2$, $t_5^1 (2, 5) = \{3, 5\}$ and $t_5^2 (2, 5) = \{2, 4\}$

$\therefore Q_1 = \emptyset$ and hence no development of the tree is at all

necessary. The beneficial effect of an ordering of the variables is apparent. When ordering the variables, it would be better to assign to those variables having a great chance of being members of an optimal solution, the smallest indices, while to those variables with a small chance the largest indices should be assigned. (This is so, because the tree is developed in descending order of the variables index size). If the reverse was done we might have progressed far enough in the development of the tree just to realize later that there exist no common solutions to the k constraints. Of course the underlying objective is to make the set intersections of the section 3.2.4 either empty or "small" early enough in the process. Assuming that the problem to be solved has a uniform structure, the variables that have a better chance of being members of an optimal solution are those which have positive a_{ij}'s in the constraints, while those variables that have "contradictory" coefficients a_{ij} (i.e. positive a_{ij} in some constraints and negative in others) have a lesser chance of being members of an optimal solution. Of course, variables having negative a_{ij}'s in almost all constraints have the least chance of being members of an optimal solution. Many heuristic rules were tried for ordering the variables but the most successful one was as follows:

1. Order the coefficients a_{ij} of every constraint i $1 \leq i \leq k$ in descending order: $a_{ij_1} \geq a_{ij_2} \geq \ldots \geq a_{ij_n}$

2. Define the set $a_{j_i} = \{a_{1j_i}, a_{2j_i}, \ldots, a_{kj_i}\}$

3. The s ($s \leq k$) different variables corresponding to the set a_{j_n} are assigned the indices n, n - 1, ..., n - s + 1. The s' different variables corresponding to the set $a_{j_{n-1}}$ are numbered n - s, n - s - 1, ..., n - s - s' + 1 and so on until all variables have been numbered.

For the example of the present section the above rule was applied:

The constraints of the problem were:

$$-3x_1 - 4x_2 + 3x_3 + 2x_4 + 4x_5 \geq 5$$

$$4x_1 + 6x_2 - 3x_3 - x_4 + 5x_5 \geq 5$$

Setting the coefficients in descending order we have

$$4x_5 + 3x_3 + 2x_4 - 3x_1 - 4x_2 \geq 5$$

$$6x_2 + 5x_5 + 4x_1 - x_4 - 3x_3 \geq 5$$

Thus, the ordering of the variables is:

$$x_5\ x_1\ x_4\ x_2\ x_3$$

and the new structure is:

$$4x_1 - 3x_2 + 2x_3 - 4x_4 + 3x_5 \geq 5$$

$$5x_1 + 4x_2 - x_3 + 6x_4 - 3x_5 \geq 5$$

Petersen's problems* [44] 6 and 7 were solved once without any ordering of the variables and a second time after applying the

---

* As found in [44] after rescaling their coefficients to lie in the range 0 - 50.

above heuristic rule and the results are as shown in Table 2.

TABLE 2

	Time CDC 6600 secs		
	Without any ordering of the variables	By ordering the variables	Value of k
Petersen's problem 6 N = 39 M = 5	11.0	1.5	4
Petersen's problem 7 N = 50 M = 5	24.0	4.7	4

Other random problems were solved once without any ordering of the variables and a second time after applying the heuristic rule just presented and the results are as shown in Table 3.

TABLE 3

		CDC 6600 secs	
M	N	Without any ordering of the variables	By ordering the variables
20	40	48.	32.
15	45	38.	25.
20	50	70.	56.
15	50	86.	16.7
15	55	166.	20.3
50	60	178.	94.3

4.4.2. Use of LP and cutting planes

Another way of limiting the size of the solution tree, which makes use of LP and cutting planes was presented in section 3.4. However, in the computational results of the next section no use of LP is made at any stage, and thus the computing times reported are on the conservative side.

4.5. Computational results

The computational performance of the method has been tested on problems found in the literature and on 80 random problems. The results of the problems of the literature are as shown in Table 4 and the results for the random problems are shown in Table 5. The heuristic rule for ordering the variables (section 4.4.1) was applied for all the problems in the following tables.

Table 4

Computational results for problems in the literature

Problem No.	Number of variables n	Number of constraints m	Time (CDC 6600 secs)	Value of k	
1	10	6	.15	3	PETERSEN'S Problems
2	10	10	.34	5	
3	15	10	.28	5	
4	20	10	.46	5	
5	28	10	.79	5	
6	39	5	1.5	4	
7	50	5	4.7	4	
8	17	15	.8	5	HALDI'S Problems
9	15	35	4.3	10	

The first 7 problems are Petersen's problems as found in [44], after rescaling their coefficients to lie in the range 0 - 50.

Problems 8 and 9 are Haldi's problem 5 and 9 respectively, as found in [46].

Table 5

Computational results for random problems

n	m	Number of problems solved	CDC 6600 secs			Value of k
			Maximum time	Minimum time	Average time	
30^{+}	15	3	8.5	1.9	5.06	10
40^{+}	20	3	13.4	4.3	7.96	10
50^{+}	15	3	18.2	9.9	13.06	10
60^{+}	15	3	43.0	11.5	28.86	10
65*	10	5	90.60	24.32	38.87	8
65	20	2	57.1	12.2	34.6	15
70*	10	7	109.69	15.52	59.11	8
70	20	2	30.2	13.5	21.8	15
70	30	4	61.1	33.2	44.5	24
75	20	2	33.9	13.2	23.5	15
80	20	4	63.2	14.4	36.6	15
80	30	5	107.1	23.0	32.6	24
85	20	7	53.3	13.3	25.4	15
90	20	7	51.5	17.0	25.4	15
90	30	3	165.2	52.6	111.2	24
95	20	5	38.8	18.3	28.5	15
100	20	5	68.3	20.6	30.6	15
100	30	3	103.2	28.9	55.4	24
105	20	5	53.9	22.9	31.0	15
110	20	4	43.8	24.6	30.2	15

$^{+}$: These problems leave a density of 66% and the ranges of the coefficients are: $1 \leq c_j \leq 5$, $10 \leq b_i \leq 30$, $-9 \leq a_{ij} \leq 9$.

*: These problems have a density of 33% and the ranges of the coefficients are: $1 \leq c_j \leq 5$, $10 \leq b_i \leq 20$, $-3 \leq a_{ij} \leq 9$.

The problems without a superscript have a density of 33% and the ranges of the coefficients are: $1 \leq c_j \leq 5$, $10 \leq b_i \leq 20$, $-3 \leq a_{ij} \leq 15$.

CHAPTER 5

TWO TREE SEARCH ALGORITHMS USING THE LOGICAL STRUCTURE OF THE PROBLEM

5.1. Introduction

In this chapter, two implicit enumeration algorithms are developed. These algorithms, use logical tests that arise from the structure of the problem and are employed when trying to fathom a node of the search tree and in deciding which variable(s) to branch on.

The first method to be presented, is a breadth-first non-binary tree search where more than two branches can emanate from a node while the other is a depth-first binary tree search with two branches enamating from a node every time. Both methods use the concept of a "reduced set", which we will now describe.

5.2. Reduced sets

5.2.1. Definitions

Consider the 0 - 1 programming problem of m constraints and n variables:

$$\left.\begin{aligned} \min \ & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \ & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i = 1, \ldots, m \\ & x_j \in \{0, 1\} \quad j = 1, \ldots, n \\ & \text{and } c_j \geq 0 \end{aligned}\right\} \quad P$$

A reduced set R_i is associated with every constraint i of problem P, and is defined as follows:

a. When the right hand side b_i is positive:

A reduced set is that (smallest) set of variables at least one of which must be equal to one if constraint i is to be satisfied, i.e. at least one of the variables in the set R_i must take the value 1 in the optimal solution of problem P.

b. When the right hand side of a constraint i is non-positive

Obviously, a constraint with a non-positive right hand side can be satisfied by setting all the variables x_j, $j = 1, \ldots, n$ equal to 0. A set R_i, in this case, is defined as that set of variables everyone of which can take the value 1 (individually) without violating the feasibility of constraint i. In other words,

$$\left\{R_i \;/\; b_i \leq 0\right\} = \left\{x_{j_1}, x_{j_2}, \ldots, x_{j_s}\right\}$$

where for any $x_{j_\ell} \in R_i$ we have $b_i - a_{ij_\ell} \leq 0$.

5.2.2. Calculation of the reduced sets

If all b_i $i = 1, \ldots, m$ are non positive then, the optimal solution to problem P is: $x_j = 0$ for all $j = 1, \ldots, n$, and there is no need to calculate any R_i. But if some b_i's are positive, the m sets R_i are calculated as follows:

a. For those constraints i with $b_i > 0$ [47]

The a_{ij}'s are ordered in descending order so that we have the sequence $a_{ij_1}, a_{ij_2}, \ldots, a_{ij_s}, a_{ij_{s+1}}, \ldots, a_{ij_n}$ where $a_{ij_s} \geq a_{ij_{s+1}}$.

Then a variable x_{j_s} is a member of R_i if and only if:

$$\sum_{r=s}^{q} a_{ij_r} \geq b_i \text{ for some } q = s, s+1, s+2, \ldots, n$$

When a variable x_{j_s} cannot be a member of the set R_i it is obvious that none of the variables $x_{j_{s+1}}, x_{j_{s+2}}, \ldots, x_{j_n}$ can belong to R_i.

b. When the right hand side of constraint i is negative or zero

The set R_i contains those variables x_{j_s} for which $a_{ij_s} \geq b_i$

5.2.3. Example

Consider the problem:

$$\begin{aligned} \min \quad z = {} & x_1 + 3x_2 + 5x_3 + 7x_4 + 10x_5 \\ \text{s.t.} \quad & -4x_1 + x_2 + x_3 - 3x_4 + 5x_5 \geq 2 \\ & 2x_1 - 2x_2 - 2x_3 + 6x_4 - 3x_5 \geq -2 \\ & -x_1 - x_2 - x_4 + 2x_5 \geq 1 \end{aligned}$$

The sets R_i are:

$$R_1 = \{5, 2\}$$
$$R_2 = \{1, 2, 3, 4\}$$
$$R_3 = \{5\}$$

5.2.4. Uses of reduced sets in the sequential method of Chapter 3

The idea of the reduced set can also be applied to the method developed in Chapter 3, where it can facilitate the exploration of the existence of a common feasible solution to the m constraints, in the following way:

Suppose that k constraints are "intersected" the remaining m - k constraints being "passive", and that the backtracking process is at "stage" j_i i.e. $n - j_i + 1$ variables are fixed at either 0 or 1. Then,

at stage j_i the sets R_i can be found for the m - k constraints where the right hand sides of these are changed accordingly as $n - j_i + 1$ variables are fixed. Then, if any $R_i = \emptyset$, for any i, $i = k + 1, k + 2, \ldots, m$, the current development of the tree of common solutions can stop and a new iteration, corresponding to a higher value of the objective function can start.

In addition, if $R_i \neq \emptyset$ for all $i = k + 1, \ldots, m$

and $\sum_{j \geq j_i} c_j x_j + \max_{k+1 \leq i \leq m} \min_{j \in R_i} c_j > u$

(where u is the current value of the objective function) the current iteration can stop as there is no possibility of having a common solution to the m constraints, of cost u.

Moreover, if $R_i \neq \emptyset$ for all $i = k + 1, \ldots, m$ and some R_i's say $R_{\ell_1}, \ldots, R_{\ell_n}$ $k + 1 \leq \ell_i \leq m$ have an empty intersection $Q = R_{\ell_1} \cap R_{\ell_2} \cap \ldots \cap R_{\ell_n} = \emptyset$ the development of the tree of solutions can stop if the following is true:

$$\sum_{j \geq j_i} c_j x_j + \sum_{\ell_i} \min_{j \in R_{\ell_i}} c_j > u$$

Finally, as it was stated in section 4.2. (c) the reduced sets can help in reducing the storage requirements of the method presented in Chapter 3. In this case, instead of intersecting k original constraints one can intersect the following m constraints:

$$\sum_{j \in R_i} x_j \geq 1 \quad i = 1, \ldots, m$$

5.3. A NON-BINARY TREE SEARCH METHOD BASED ON REDUCED SETS

5.3.1. Introduction

Non- binary tree search both breadth-first and depth-first have been used previously in algorithms for solving the IP problem. e.g. [43], [47]. The main disadvantages of a breadth-first approach, is that it might require a lot of computer storage, especially when applied to the general IP problem. The main advantage, however, of a breadth-first approach is that, it requires, less branchings than those required by depth-first search.

The present method is a non-binary breadth-first tree search and it deviates from the other algorithms of breadth-first tree search in:

a. selecting a variable to branch

b. the branching strategy

c. the way a node is fathomed

where in a, b, and c extensive use of the sets R_i is made.

In the method proposed here, every branch that emanates from a node corresponds to fixing a variable at 1. A node corresponds to a partial solution S and the cost associated with that node is $\sum_{j \in S} c_j$. That node is chosen for branching which is associated with the lowest cost among the live ones. Once a node, corresponding to a partial solution S, is chosen the right hand sides

$b_i' = b_i - \sum_{j \in S} a_{ij}$ $i = 1, \ldots, m$ are found. If $b_i' \leq 0$ for all i $i = 1, \ldots, m$, then the solution $x_j = 1$ $j \in S$, $x_j = 0$ $j \notin S$ is the optimal. If $b_i' > 0$ for some i, then the m sets R_i are found, and if any of these is empty, the current node under consideration is fathomed since the following is true:

a. If $R_i = \emptyset$ $b_i' > 0$, obviously, the ith constraint cannot be satisfied.

b. $R_i = \emptyset$ $b_i' \leq 0$ In this case the ith constraint is satisfied by $x_j = 0$ $j \notin S$, $x_j = 1$, $j \in S$, but the fact that some other constraint r has $b_r' > 0$ necessitates that some variable(s) have to be fixed at 1; this, however, is not permitted by constraint i.

Thus, if any $R_i = \emptyset$ the node under consideration is fathomed. If all $R_i \neq \emptyset$, then a set $R_k = \{\ell_1, \ell_2, \ldots, \ell_t\}$ is selected and t nodes are created that correspond to the partial solutions $\{S, \ell_1\}$, $\{S, \ell_2\}$, $\ldots$, $\{S, \ell_t\}$ and to the costs $\sum_{j \in S} c_j + c_{\ell_1}$, $\sum_{j \in S} c_j + c_{\ell_2}$, $\ldots$, $\sum_{j \in S} c_j + c_{\ell_t}$. These t nodes, then join the list of live nodes, and the process continues.

That set R_k should be chosen, so that one proceeds to the optimum quickly. Some alternative heuristic ways of choosing the set R_k are:

a. Select that set R_k which contains the variable corresponding to $\max_{1 \leq i \leq m} \min_{x_j \in R_i} c_j$ and all the other variables $x_j \in R_k$ have larger c_j's.

b. Let the set R_k be the one which has the minimum number of elements. Call this set R_{min}.

The method presented here employs the second alternative.

Also, several bounds can be introduced in this algorithm, and some of them are as follows:

a. Solve the LP probelm:

$$\min \; z_s = \sum_{j \notin S} c_j x_j$$

$$\text{s.t.} \quad \sum_{j \notin S} a_{ij} x_j \geq b_i' \quad i = 1, \ldots, m$$

$$x_j \geq 0$$

Then the bounds associated with the t nodes generated, are:

$$\sum_{j \in S} c_j + \max \left[c_{\ell_q}, \; \lfloor z_s \rfloor \right] \qquad q = 1, \ldots, t$$

where $\lfloor z_s \rfloor$ is the smallest integer greater than z_s.

b. Solve the covering problem:

$$\min \; z_s' = \sum_{j \notin S} c_j x_j$$

$$\text{s.t.} \quad \sum_{j \notin S} d_{ij} x_j \geq 1 \quad i = 1, \ldots, m$$

$$x_j \geq 0$$

where $d_{ij} = 0$ if $x_j \notin R_i$

$d_{ij} = 1$ if $x_j \in R_i$

Then, the bounds associated with the t nodes are:

$$\sum_{j \in S} c_j + \max \left[c_{\ell_q}, \; \lfloor z_s' \rfloor \right] \qquad q = 1, \ldots, t$$

c. Find the variable x_j, having the cost coefficient

$\max_{1 \leq i \leq m} \min_{j \in R_i} c_j = p$ Then, the bound associated with every newly generated nodes is:

$$\sum_{j \in S} c_j + \max \left[p, c_{\ell_q} \right]$$

The bound presented in paragraph c is, of course, the easiest to calculate and it has been used in the problems whose computational results appear in Table 1 of section 5.3.5.

5.3.2. The algorithm

The algorithm then using the bound presented in paragraph c above proceeds as follows:

Step 1* Find that node ℓ, among the live ones,** which has a minimum cost, z_ℓ associated with it, and corresponds to the partial solution S_ℓ. Go to step 2.

Step 2 Check if any other live nodes of the same cost z_ℓ (if any) correspond to the same partial solution S_ℓ. If so, delete them from the list. Go to step 3.

Step 3 Find $b'_i = b_i - \sum_{j \in S} a_{ij}$ $i = 1, \ldots, m$

If $b'_i \leq 0$ for all $i = 1, \ldots m$ the optimum has been found

$z^* = z_\ell \quad x_j = 0 \quad j \notin S_\ell, \quad x_j = 1 \quad j \in S_\ell$

If there exist some $b'_i > 0$, go to step 4.

Step 4 Find the m sets R_i $i = 1, \ldots, m$ where only free variables are considered and the right hand sides are b'_i. If any set R_i is empty the node is fathomed, go to step 1.

If $R_i \neq \emptyset$ for all $i = 1, \ldots, m$ go to step 5.

*At the start, the starting node ℓ has $z_\ell = 0$, $S_\ell = \emptyset$

**If the list of live nodes is empty, the problem is infeasible.

Step 5 Find that the set which contains the minimum number of elements R_{min} (if more than one sets have the same number of elements, say these sets are the jth, kth, choose that R_{min} which contains the variable with cost coefficient: $\max_{i=j,k} \min_{j \in R_i} c_j$). Go to step 6.

Step 6 Say $R_{min} = \{\ell_1, \ell_2, \ldots, \ell_t\}$ and $p = \max_{1 \le i \le m} \min_{j \in R_i} c_j$

Create the t nodes corresponding to the partial solutions

$$\{S, \ell_1\}, \{S, \ell_2\}, \ldots, \{S, \ell_t\}$$

and having associated costs:

$$\sum_{j \in S} c_j + \max [p, c_{\ell_1}], \sum_{j \in S} c_j + \max [p, c_{\ell_2}]$$

$$\ldots, \sum_{j \in S} c_j + \max [p, c_{\ell_t}]$$

Add these t nodes to the list. Go to step 1.

5.3.3. An example:

Consider the example of section 3.5.1.:

$$\min \quad z = 2x_1 + 4x_2 + 3x_3 + x_4 + 2x_5 + x_6$$

s.t.

$$4x_1 + 3x_2 + 2x_3 + 2x_4 + 2x_5 - x_6 \ge 7$$

$$7x_1 + 2x_2 - 3x_3 - x_4 - x_5 - x_6 \ge 4$$

$$3x_1 - 3x_2 + 3x_3 - 2x_4 - x_5 + 3x_6 \ge 3$$

$$x_1 + 2x_2 + 3x_3 + 3x_4 - x_5 + 3x_6 \ge 3$$

The sets R_i are as follows:

$R_1 = \{1, 2\}$, $R_2 = \{1\}$, $R_3 = \{1, 3, 6\}$ $R_4 = \{3, 4, 6, 2\}$

$Rmin = R_2 = \{1\}$

So x_1 is fixed at 1 and we have

$b'_1 = 7 - 4 = 3$, $b'_2 = -3$, $b'_3 = 0$, $b'_4 = 2$

The new sets R_i are:

$R_1 = \{2, 3, 4\}$, $R_2 = \{2, 3, 4, 5, 6\}$ $R_3 = \{3, 6\}$

$R_4 = \{2, 3, 4, 6\}$ $\therefore$ $R_{min} = R_3 = \{3, 6\}$

$p = \max\limits_{1 \leq i \leq m} \min\limits_{j \in R_i} c_j = 1$

We have, thus, the nodes 2 and 3 of costs:

$c_1 + \max [p, c_3] = c_1 + c_3 = 5$ and $c_1 + \max [p, c_6] = c_1 + c_6 = 3$

(see Fig. 1)

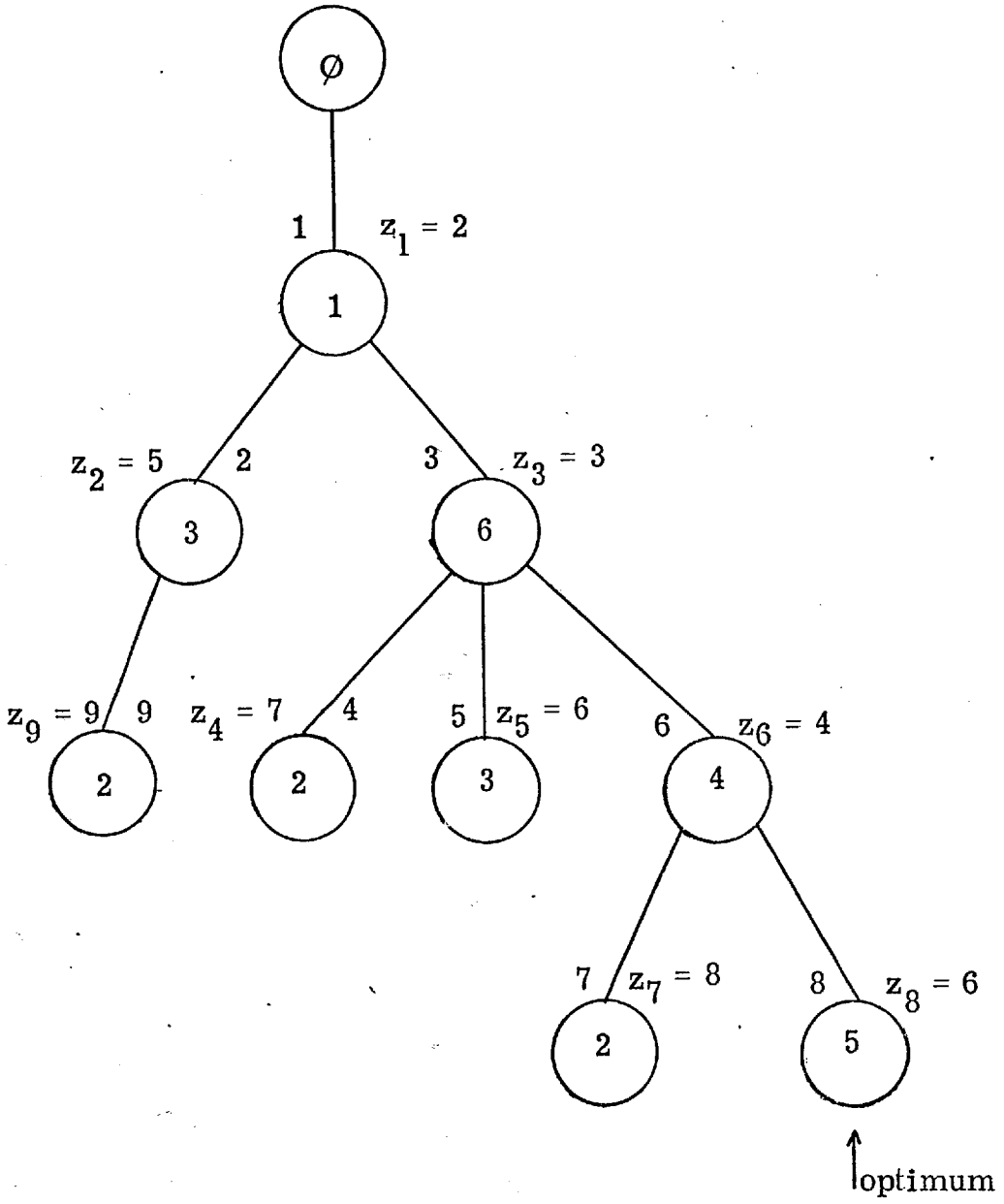
The minimum cost node is node 3 with $z_3 = 3$.

Setting x_1 and x_6 equal to 1 we have:

$b'_1 = 4$, $b'_2 = -2$ $b'_3 = -3$, $b'_4 = -1$

and $R_1 = \{2, 3, 4\}$ So, the set R_1 will create three nodes (see Fig. 1) of costs $3 + c_2 = 7$, $3 + c_3 = 6$, $3 + c_4 = 4$. Now, the node associated with the minimum cost is node 6 and by setting x_1, x_6, x_4 at 1 we have: $b'_1 = 2$, $b'_2 = -1$, $b'_3 = -1$, $b'_4 = -4$ and $R_1 = \{2, 3, 5\}$, $R_2 = \{2, 5\}$, $R_3 = \{3, 5\}$, $R_4 = \{2, 3, 5\}$ and $R_{min} = R_2 = \{2, 5\}$. Thus the nodes 7 and 8 are created. The next node for branching is node 2 of cost $z_2 = 5$. Setting x_1 and x_3 at 1, we have $R_{min} = R_2 = \{2\}$ and node 9 is created of cost $z_9 = 9$. Now, node 8 is the minimum cost node ($z_8 = 6$).

Setting x_1, x_4, x_5, x_6 at 1 we have:



The numbers adjacent to the nodes represent the sequence according to which the nodes were generated. The encircled numbers represent the indices of the variables that are fixed.

Figure 1

$b'_1 = 0,\ b'_2 = 0,\ b'_3 = 0,\ b'_4 = -3$

The optimum has been found as all right hand sides are non-positive.

The optimal solution is

$x_1 = x_4 = x_5 = x_6 = 1 \quad x_2 = x_3 = 0 \quad$ and $\quad z^* = 6$

5.3.4. Improvements to the above algorithm

Similar to the definition of the set R_i another set L_i can be defined for every constraint i, which contains those variables that should take the value 0 in an optimal solution. L_i then, contains those variables that correspond to the coefficients a_{ik} for which $a_{ik} < b_i - \sum a^+_{ij}$ where $\sum a^+_{ij}$ denotes the sum of the positive coefficients of constraint i.

In the present algorithm, the number of the nodes of the tree depends on the size of R_{min} and R_{min} can be further reduced if the sets L_i are introduced: Every set R_i can be intersected with each set L_k $k = 1, \ldots, m$ and produce a set R'_i,

$$R'_i = \left\{ R_i - \left\{ R_i \cap L_1,\ R_i \cap L_2,\ \ldots, R_i \cap L_m \right\} \right\}$$

so that, R'_{min} is that set among R'_i containing the minimum number of elements.

E.g. consider the constraints:

$$2x_1 - 3x_2 + x_3 + 3x_4 - 4x_5 \geq 3$$

$$3x_1 + 2x_2 - x_3 - 6x_4 + 2x_5 \geq 2$$

Then: $R_1 = \{4, 1\} \qquad L_1 = \{5\}$

$R_2 = \{1, 2, 5\} \quad L_2 = \{4\}$

$$R'_1 = R_1 - \{R_1 \cap L_2\} = \{1\}$$

$$R'_2 = R_2 - \{R_2 \cap L_1\} = \{1, 2\}$$

and $R'_{min} = \{1\}$ whereas without the introduction of the sets L_i, $R_{min} = \{4, 1\}$.

5.3.5. Computational aspects

The basic storage requirements of the algorithm are:

1. An one-dimensional array of size N for keeping the costs of the live nodes where N is an upper bound on the number of live nodes at any stage of the algorithm.
2. An one-dimensional array for keeping the partial solutions that correspond to the live nodes. The storage is done here as in the method of Chapter 3 i.e. a computer word that corresponds to an element of this array, is assumed to store up to d indices where the word length is d bits. If $d < n$ then $\lfloor \frac{n}{d} \rfloor$ arrays are required for storing the solutions S.

The method presented here has been tested on problems found in the literature and on a few random problems. The results are as shown in Table 1.

Table 1

Computational Results

Problem No.	Number of variables n	Number of constraints m	Present method		
			Iterations**	Time (sec) CDC 6600	
1	6	10	6	0.22	Petersen's [44] problems
2	10	10	8	0.23	
3	15	10	56	0.73	
4	20	10	366	4.63	
5	28	10	481	4.16	
6	39	5	802	11.7	
7	50	5	870	12.6	
8	17	15	374	2.85	Haldi's [46] problems
9	15	35	715	6.95	
10	10	1	6	0.09	Trauth's [17] allocation problems
11	10	1	21	0.32	
12	10	1	17	0.30	
13	10	1	14	0.27	
14	10	1	14	0.25	
15	10	1	16	0.24	
16	10	1	15	0.23	
17	10	1	7	0.11	
18	10	1	4	0.08	
19	12	6	5	0.4	Random problems*
20	20	10	35	0.92	
21	30	15	76	1.14	
22	35	15	88	1.32	
23	50	15	1112	47.5	

* The density of these problems is 40% and $-5 \leq a_{ij} \leq 10$, $5 \leq b_i \leq 30$

** By "iteration" here is meant the finding of the m sets R_i

5.4. A BINARY TREE SEARCH METHOD

This method solves problem P (as defined in section 5.2.1.) and it is described in terms of a depth-first approach.

The algorithm consists of two distinct parts, the first part is directed at finding a feasible solution (of cost z^0) as early as possible and the second part is the search for the optimal solution of cost $z^* \leq z^0$ and uses a different branching rule than the first part. Both parts at every branching step, calculate the m reduced sets R_i (as defined in section 5.2.1.) for every constraint i, i=1, ..., m. At every stage of the algorithm there is a set S of variables that are fixed at either 1 or 0, the remaining variables being free . The set S of fixed variables is expressed in the usual way, where an index with a positive sign signifies that the variable of that index is fixed at 1 while a negative sign signifies that the corresponding variable is fixed at 0. E.g. $S = \{2, 5, -6, 7\}$ means that $x_2 = 1$, $x_5 = 1$ $x_6 = 0$ and $x_7 = 1$.

5.4.1. Finding an initial feasible solution to problem P

The reduced sets R_i i = 1, ...,m are found at every step of this initial procedure. Let s_j, $1 \leq s_j \leq m$, be the number of the sets R_i that contain the variable x_j (the index j) e.g. if m = 3, and $R_1 = \{2, 4, 5\}$, $R_2 = \{1, 2\}$ and $R_3 = \{3, 4\}$ we have $s_1 = 1$, $s_2 = 2$, $s_3 = 1$, $s_4 = 2$, $s_5 = 1$ all other $s_j = 0$. Initially, $S = \emptyset$ and let $b'_i = b_i$ The initial feasible solution is then found as follows:

Step 1 Find $b'_i = b_i - \sum_{j \in S} a_{ij} x_j$ for all $i = 1, \ldots, m$

If $b'_i \leq 0$ for all $i = 1, \ldots, m$ go to step 6

If $b'_i > 0$ for some i, go to step 2.

Step 2 Find the sets R_i for all $i = 1, \ldots, m$, when the right hand sides of the constraints are b'_i and only the free variables x_j $j \notin S$ are considered.

If for any i, $R_i = \emptyset$ go to step 4.

If for all i, $R_i \neq \emptyset$ go to step 3.

Step 3 (Branching) Find that variable x_j for which $s_j = \max_{1 \leq t \leq n} s_t$

(In case two or mor s'_ts have the same value, $\max_{1 \leq t \leq n} s_t$, choose that variable j for which $c_j = \min \left\{ c_t / t : \max_{1 \leq t \leq n} s_t \right\}$)

Fix the variable x_j at 1 and add its index j to the set of fixed variables: Make $S \longrightarrow \{S, +j\}$

(This variable x_j is set to 1 because it has the greatest chance of leading towards feasibility quicker.)

Go to step 1.

Step 4 (Backtracking) Find the rightmost element j_r of the set S, which has a positive sign i.e. find that variable last fixed at 1; if no such element exists go to step 5. Otherwise, reverse the positive sign of j_r to minus i.e. fix x_{j_r} at 0. Discard any indices located to the right of j_r (which, obviously, have a minus sign).

Go to step 1.

Step 5 The problem has no feasible solution.

Step 6 A feasible solution has been found:

Its value is $z^0 = \sum_{j \in S} c_j x_j$ and $x_j = 1$ if $+j \in S$,

$x_j = 0$ if $-j \in S$, $x_j = 0$ if $j \notin S$.

An example

This example is specially constructed to show all the steps of the technique of finding an initial feasible solution.

Consider the problem:

$$\min z = x_1 + 2x_2 + 3x_3 + x_4 + 2x_5 + 2x_6$$

$$\text{s.t.} \quad 2x_1 - 3x_2 + 3x_3 + x_4 - x_5 \geq 3$$

$$3x_1 + x_2 - 2x_3 - 4x_4 - x_5 - x_6 \geq 3$$

$b'_1 = 3$, $b'_2 = 3$ and $R_1 = \{1, 3\}$, $R_2 = \{1\}$ $s_1 = \max_{1 \leq t \leq 6} s_t = 2$ and variable x_1 is fixed at 1: $S = 1$. The new rhs are:

$b'_1 = 1$, $b'_2 = 0$, and $R_1 = \{3, 4\}$ $R_2 = \{2\}$. We have $\max_{1 \leq t \leq 6} s_t = 1$ and variable x_4 is fixed at 1 as it has the smallest cost coefficient among the variables having the same value $\max_{1 \leq t \leq 6} s_t$. $\therefore$ $S = \{1, 4\}$ and $b'_1 = 0$, $b'_2 = 4$, $R_1 = \{3\}$, $R_2 = \emptyset$.

Setting the rightmost positive element of S at 0, we have:

$S = \{1, -4\}$ and $b'_1 = 1$, $b'_2 = 0$, $R_1 = \{3\}$, $R_2 = \{2\}$.

Fixing variable x_2 at 1, we have:

$S = \{1, -4, 2\}$, $b'_1 = 4$, $b'_2 = -1$, $R_1 = \emptyset$. Thus, the variable x_2 has to be fixed at 0 and the set S becomes: $S = \{1, -4, -2\}$ and $b'_1 = 1$, $b'_2 = 0$, $R_1 = \{3\}$, $R_2 = \emptyset$. Thus S becomes $S = \{-1\}$ which leads to $b'_1 = 3$, $b'_2 = 3$ and $R_1 = \{3\}$, $R_2 = \emptyset$.

The set S contains no positive element, thus the problem is infeasible.

5.4.2. Finding the optimal solution

This part of the algorithm finds the optimal solution to problem P by either establishing the optimality of the already found feasible solution or by finding other better feasible solutions of cost $z < z^0$. Here, use is made of LP to facilitate the fathoming of nodes.

Branching:

At every step, the sets R_i are found for all $i = 1, \ldots, m$, and the set R_{min} having the minimum number of elements is considered. That variable j_b is chosen for branching which is contained in R_{min}, and $c_b = \min_{j \in R_{min}} \{c_j\}$. This branching rule aims at finding a feasible solution of cost $< z^0$.

Fathoming of a node.

Further branching from a node is discontinued when one of the following conditions is true:

1a. If $\sum_{\pm j \in S} c_j x_j + \lfloor w \rfloor \geq z^0$

where w is the LP optimum of the remaining subproblem:

$$\left.\begin{array}{ll} \min & w = \sum_{j \notin S} c_j x_j \\ \text{s.t.} & \sum_{j \notin S} a_{ij} x_j \geq b_i' \text{ for those i for which } b_i' > 0 \end{array}\right\} \; T(S)$$

1b. If Problem T(S) is infeasible.

2. If any $R_i = \emptyset$ for any $1 \leq i \leq m$, branching is discontinued from that particular node.

3. After having chosen the variable to branch, j_b, if $c_b + \sum_{j \in S} c_j x_j > z^o$ branching stops from that node.

5.4.3. The algorithm

Thus, the algorithm proceeds as follows:

Step 1 Find an initial feasible solution in the way described in section 5.4.1. If no such solution can be found the problem is infeasible. If such a solution exists, let it be expressed by the set S^o which contains the indices of the variables fixed at either 1 or 0, and where the remaining variables not contained in the set S^o have the value 0. Let the cost of this solution be z^o. Let $S = S^o$.

Step 2 (Backtracking) Find the rightmost element of S, j_r, which has a positive sign. Make $j_r \longrightarrow -j_r$. If no element of S has a positive sign go to step 9. Otherwise, discard the elements of S which have a minus sign and are located to the right of $-j_r$. Go to step 3.

Step 3 Find the new right hand sides b'_i. If all $b'_i \leq 0$, another feasible solution has been found of cost $\sum_{\pm j \in S} c_j x_j$. Go to step 8. If some $b'_i > 0$ go to step 4.

Step 4 Find the sets R_i for $i = 1, \ldots, m$

If $R_i = \emptyset$ for any i, go to step 2. Otherwise go to step 5.

Step 5 Solve the following LP problem:

$$\min \; w = \sum_{j \notin S} c_j x_j$$

s.t. $\sum_{j \notin S} a_{ij} x_j \geq b'_i$ and for those i for which $b'_i > 0$. If this problem is infeasible or if $\sum_{\pm j \in S} c_j x_j + \lfloor w \rfloor \geq z^o$, go to step 2. Otherwise; go to step 6.

Step 6 Find the set R_{min} from the newest generated sets R_i. Find that variable j_b, $j_b \in R_{min}$ and $c_b = \min_{j \in R_{min}} \{ c_j \}$

If $\sum_{\pm j \in S} c_j x_j + c_b \geq z^o$ go to step 2. Otherwise, go to step 7.

Step 7 (Branching) Add the index j_b to the set S. (The consequences of setting x_{j_b} at 1 are examined first). Go to step 3.

Step 8 If $\sum_{\pm j \in S} c_j x_j \geq z^o$, go to step 2. If $\sum_{\pm j \in S} c_j x_j < z^o$, the feasible solution just found, has a lower cost, and z^o is replaced by $\sum_{\pm j \in S} c_j x_j$ and the solution S is stored in S^o. Go to step 2.

Step 9 The search has finished and the optimal solution z^* has been found: $z^* = z^o$ and $x_j = 1$ if $+j \in S^o$, $x_j = 0$ if $-j \in S^o$ and $x_j = 0$ if $j \notin S^o$

5.4.4. An example

Consider the problem of chapter 3 whose initial feasible solution is found to be: $x_1 = x_2 = x_3 = 1$, $x_4 = x_5 = x_6 = 0$, $z^o = 9$.

The initial solution is: $S = \{1, 3, 2\}$

Applying step 2 of the algorithm we have: $S = \{1, 3, -2\}$ The sets R_i when $S = \{1, 3, -2\}$ are: $R_1 = \{4, 5\}$, $R_2 = \emptyset$, $R_3 = \{4, 5, 6\}$, $R_4 = \{4, 5, 6\}$. Therefore, $S = \{1, -3\}$, which leads to $R_{min} = R_3 = \{6\}$. Applying step 7 we have: $S = \{1, -3, 6\}$ and the following LP problem has to be solved:

$$\min \; w = 4x_2 + x_4 + 2x_5$$

$$\text{s.t.} \quad 3x_2 + 2x_4 + 2x_5 \geq 4$$

giving $\lfloor w \rfloor = 2$. $\sum_{j \in S} c_j x_j + \lfloor w \rfloor = 3 + 2 = 5 < 9$

$R_{min} = R_1 = \{2, 4\}$ $S = \{1, -3, 6, 4\}$ $b'_1 = 2$, $b'_2 = -1$, $b'_3 = -1$ $b'_4 = -4$ and $R_{min} = R_3 = \{5\}$.

The solution $S = \{1, -3, 6, 4, 5\}$ of cost $z = 6$ is feasible to the problem and therefore $z^o = 6$. We then start backtracking by considering $S = \{1, -3, 6, 4, -5\}$ which gives $b'_1 = 2$, $b'_2 = -1$, $b'_3 = -1$, $b'_4 = -4$ and $R_3 = \emptyset$. $S = \{1, -3, 6, -4\}$ gives $b'_1 = 4$, $b'_2 = -2$, $b'_3 = -3$, $b'_4 = -1$ and $R_{min} = R_1 = \{2\}$

The problem

$$\min \; w = 4x_2 + 2x_5$$

$$\text{s.t.} \quad 3x_2 + 2x_5 \geq 4$$

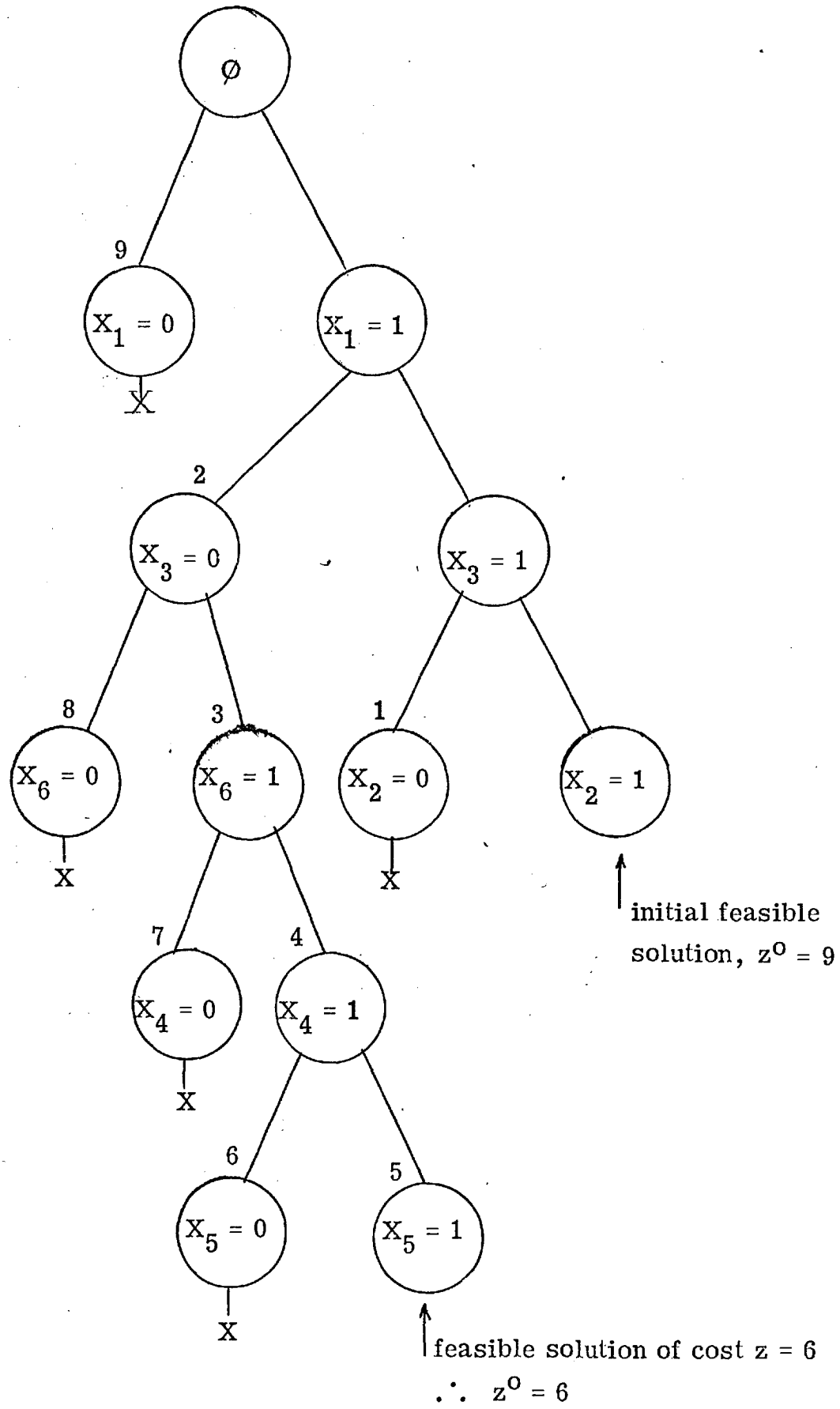
gives $w = 4$ and $c_1 + c_6 + w = 3 + 4 = 7 > z^o = 6$ and the branch is discontinued. $S = \{1, -3, -6\}$ gives $b'_1 = 3$, $b'_2 = -3$, $b'_3 = 0$, $b'_4 = 2$ and $R_3 = \emptyset$. Finally $S = \{-1\}$ gives $R_2 = \emptyset$ and the process termin-

ates. The whole procedure is shown in Figure 2.

5.4.5. Computational results

The above algorithm has been tested on a number of random problems and the results are as shown in Table 2. It was noted from these experiments that the cost of the initial feasible solution did not differ greatly from the optimum and usually the optimal solution had a value 20% - 30% less than the cost of the initial solution.

The tree for the example of section 5.4.4.



The symbol X under a node signifies that branching from that node is discontinued.

Figure 2

Table 2

Computational results on random problems

Problem No.	n	m	Iterations	Time (secs) CDC 6600
1	40	10	238	21.9
2	40	15	886	110.3
3	45	15	273	34.6
4	45	10	357	33.7
5	45	10	312	32.5
6	50	10	346	42.7
7	50	15	774	131.3
8	50	15	510	90.2
9	50	10	320	44.3
10	60	15	968	178.3
11	60	20	713	195.
12	60	10	276	41.1
13	60	10	192	32.1
14	65	10	467	69.3

Cont./.....

Problem No.	n	m	Iterations	CDC 6600 secs.
15	70	10	211	36.1
16*	70	10	461	82.
17	70	10	720	121.1
18*	70	15	928	231.
19	75	10	121	20.9
20*	75	10	326	55.2
21	75	10	193	30.5
22*	80	10	769	120.9
23	80	10	920	182.2
24*	80	10	520	101.3
25	85	10	823	135.6
26*	85	15		> 300
27	90	10	412	80.8
28	90	10	389	74.3
29	90	10	641	141.3
30*	90	10	891	221.9
31	95	10	980	215.4
32*	95	15		> 300
33	95	15	1007	242.1
34	100	10	864	166.9
35	100	10	615	114.2

* These problems have $-3 \leq a_{ij} \leq 9 \quad 0 \leq b_j \leq 20 \quad 0 \leq c_j \leq 5$

The rest of the problems have a_{ij}'s lying in the range $-3 \leq a_{ij} \leq 15$

The density of all the problems is 35%.

C H A P T E R 6

CONCLUSIONS

Conclusions

In this thesis, three new methods are presented for solving the 0 - 1 linear programming problem. The method presented in Chapter 3 is the most promising. Its computational results suggest that the method is most powerful for dense problems having small coefficients. Many improvements can be incorporated in the basic algorithm to make it even more powerful. For example, the use of LP and cutting planes can further enhance its power.

The two methods presented in Chapter 5 place the emplasis on the logical structure of the problem and their deviation from related schemes is in the branching process and the way that subproblems are fathomed. The computational results of these two methods are acceptable and they can become more effective if stronger bounds are introduced.

REFERENCES

1. BALAS E. "An Additive Algorithm for Solving the Linear Programs with 0 - 1 variables". Oper. Res. Vol 13, 1965, p.p. 517 - 546.

2. GEOFFRION A.M. "Integer Programming by Implicit Enumeration and Balas' Method". SIAM review Vol. 9, 1967, p.p. 178 - 190.

3. GLOVER F. "A Multiphase Dual Algorithm for the 0 - 1 Integer Programming Problem". Oper. Res. Vol. 13, 1965, p.p. 879 - 919.

4. GLOVER F. & ZIONTS S. "A note on the additive algorithm of Balas". Oper. Res. Vol. 13, 1965, p.p. 546 - 549.

5. HAMMER P. & RUDENAU S. "Boolean Methods in Operational Research". Springer-Verlag, Berlin 1968.

6. LEMKE C. & SPIELBERG K. "Direct Search Algorithms for 0 - 1 and Mixed Interger Programming". Oper. Res. Vol. 15, 1967, p.p. 892 - 914.

7. HAMMER P. & NGUYEN S. "A Partial Order in the Solution Space of Bivalent Programs". Report No. CRM - 163, 1972, University of Montreal.

8. LAWLER E. & BELL M.D. "A Method for Solving Discrete Optimization Problems". Oper. Res. Vol. 14, 1966, p.p. 1098 - 1112.

9. FLEISCHMANN B. "Computational Experience with the Algorithm of Balas". Oper. Res. Vol. 15, 1967, p.p. 153 - 155.

10. PIERCE J.F. "Application of Combinatorial Programming to a class of All 0 - 1 Interger Programming problems". Man Sci. Vol. 15, 1968, p.p. 191 - 209.

11. GARFINKEL R.S. & NEMHAUSER G.L. "Integer Programming" John Wiley, 1972.

12. MARTIN G.T. "Solving the Travelling Salesman Problem by Integer Programming". Control Data Corporation, New York, May 1966.

13. MARTIN G.T. "Integer Programming: Gomory Plus Ten". 38th National ORSA meeting, Miami, November 1969.

14. SRINIVASAN A.V. "An Investigation of Some Computational aspects of Integer Programming". JACM Vol. 12, 1965, p.p. 525 - 535.

15. GOMORY R. "All-Integer Integer Programming Algorithm" in Muth & Thompson "Industrial Scheduling" Prentice Hall, 1963, p.p. 193 - 206.

16. LAND A.H. & DOIG A.G. "An Automatic Method for Solving Discrete Programming Problems". Econometrica Vol. 28, 1960, p.p. 497 - 520.

17. TRAUTH C.A. & WOOLSEY R.E. "Integer Linear Programming: A Study in Computational Efficiency". Man. Sci., Vol. 11, 1969, p.p. 481 - 493.

18. GOMORY R.E. "An Algorithm for Integer Solutions to Linear Programs" in Graves & Wolfe "Recent advances in Math Progr". McGraw Hill 1963, p.p. 269 - 302. (Originally appeared in 1958).

19. GLOVER F. "Generalized Cuts in Diaphantine Programming" Man. Sci. Vol. 13, 1966, p.p. 254 - 268.

20. BOWMAN V.J. & NEMHAUSER G.L. "Deep cuts in Integer Programming". Oper. Res. Vol. 8, 1971, p.p. 89 - 111.

21. BALAS E. "Intersection Cuts - A New Type of Cutting Planes for Integer Programming". Oper. Res. Vol. 19, 1971, p.p. 19 - 39.

22. BURDET C. "A Class of Cuts & Related Algorithms in Integer Programming" Man. Sci. Res. Report No. 220, Carnegie Mellon University, 1970.

23. BALAS E., BOWMAN V., GLOVER F., SOMMER D., "An Intersection Cut from the Dual of the Unit Hypercube". Oper. Res. Vol. 19, 1971, p.p. 40 - 44.

24. BURDET C. "On Cutting Planes". Man. Sci. Report No. 319, Carnegie Mellon University, 1973.

25. BURDET C. & BREU R. "Numerical Experimentation in Large Scale Integer Programming". W.P. 90 - 72 - 3 Carnegie Mellon University, 1973.

26. YOUNG R.D. "Hypercylindrically Deduced Cuts". Oper. Res. Vol. 19, 1971, p.p. 1393 - 1405.

27. BALAS E. "Integer Programming & Convex Analysis, Intersection Cuts from Outer Polars". Math. Progr. Vol. 2, 1972, p.p. 330 - 383.

28. BURDET C. "Enumerative Cuts". Oper. Res. Vol. 21, 1973, p.p. 61 - 89.

29. GLOVER F. "Cut search Methods in Integer Programming". Math. Progr. Vol. 3, 1972, p.p. 86 - 100.

30. DAKIN R.J. "A Tree Search Algorithm for Mixed Integer Programming Problems". Computer Journal Vol. 8, 1965, p.p. 250 - 255.

31. GEOFFRION A.M. & MARSTEN R.E. "Integer Programming Algorithms: A Framework and State of the Art Survey". Man. Sci. Vol. 18, 1972, p.p. 465 - 491.

32. DRIEBEEK N.J. "An Algorithm for the Solution of Mixed Integer Programming Problems". Man. Sci. Vol. 12, 1966, p.p. 576 - 587.

33. TOMLIN J.A. "Branch & Bound Methods for Integer and Non-Convex Programming" in Abadie's "Integer & Non-Linear Programming," 1970, p.p. 437 - 450.

34. TOMLIN J.A. "An Improved Branch & Bound method for Integer Programming": Oper. Res. 19, 1971, p.p. 1070-1074.

35. BENICHOU M.J. et al. "Experiments in Mixed Integer Programming". Math. Progr. 1, 1971, p.p. 76 - 94.

36. LAWLER E.L. & WOOD D.E. "Branch & Bound Methods: A Survey". Oper. Res. 14, 1966, p.p. 699 - 719.

37. BREU R. & BURDET C.A. "Branch & Bound Experiments in 0 - 1 Programming". Man. Sci. Report, October 1973, Carnegie Mellon University.

38. ZIONTS S. "Linear and Integer Programming". Prentice Hall, 1974.

39. BRADLEY G.H. "Transformation of Integer Programs to Knapsack Problems" Discrete Math. 1, 1971, p.p. 29 - 45.

40. PADBERG M. "Equivalent Knapsack-type Formulations of Bounded Integer Linear Programs". Man. Sci. Report No. 227, 1970, Carnegie Mellon University.

41. EDMONDS J. & JOHNSON E.L. "Matching: A Well Solved Class of Integer Linear Programs". Proceedings of the Calgary International Conference (1969) on Combinatorial Structures and their applications Pub. by Gordon & Beach, 1970.

42. GEOFFRION A.M. "An Improved Implicit Enumeration Approach for Integer Programming". Oper. Res. Vol. 17, 1969, p.p. 437 - 454.

43. BALAS E. "Discrete Programming by the Filter Method". Oper. Res. Vol. 15, 1967, p.p. 915 - 957.

44. PETERSEN C. "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R & D Projects". Man. Sci. Vol. 13, 1967, p.p. 736 - 750.

45. BRADLEY G.H., HAMMER P.L., WOLSEY L. "Coefficient Reduction for Inequalities in 0 - 1 variables". Math. Progr. Vol. 7, 1974, p.p. 263 - 282.

46. HALDI J. "Integer Programming Test Problems". Working paper No. 43, Dec. 1964, Stanford University.

47. CHRISTOFIDES N. "Zero-One Programming Using Non-Binary Tree Search". Computer Journal Vol. 14, 1971, p.p. 418 - 421.

48. BERGE C. "Balanced matrices". Math. Progr. Vol. 2, 1972, p.p. 19 - 31.

49. KARP R.M. "Reducibility among combinatorial problems". in "Complexity of Computer Computations" Edited by R.E. Miller & J.W. Thatcher, Plenum Press, New York-London 1972.

50. PADBERG M.W. "Perfect zero-one matrices". Math. Progr. Vol. 6, 1974, p.p. 180 - 196.

51. GARFINKEL R.S. & NEMHAUSER G.L. "Optimal set covering: A survey" 1972, in "Perspectives on Optimization: A collection of Expository articles" Geoffrion editor 1972. Addison - Wesley.

52. DAY R.H. "On Optimal Extracting From a Multiple File Data Storage System: An Application of Integer Programming" Oper. Res. Vol. 13, 1965, p.p. 482 - 494.

53. BELLMORE M., GREENBERG H.J., JARVIS J.J. "Multi-Commodity Disconnecting Sets". Man.Sci. Vol. 16, 1970, B427 - B433.

54. BALINSKI M.L., & QUANDT R.E. "On an Integer Program For a Delivery Problem". Oper. Res. Vol. 12, 1964, p.p. 300 - 304.

55. WAGNER W.H. "An application of Integer Programming to Legislative Redistricting" presented at the 34th Nat. Meeting of ORSA.

56. GARFINKEL R.S. & NEMHAUSER G.L. "Optimal Political Districting by Implicit Enumeration Techniques". Man. Sci. Vol. 16, B495 - B508.

57. BESSIERE F. "Sur la Recherche du Nombre Chromatique d'un Graphe par un programme Lineaire en nombres entiers". Rev. Franc. Recherche Operationelle, 1965, Vol. 9. p.p. 143 - 148.

58. KOLNER T.N. "Some Highlights of a scheduling matrix generator system" United Airlines, 1966.

59. ARABEYRE J.P. et al "The airline crew scheduling problem: A Survey" Trans. Sci. Vol. 3, 1969, p.p. 140 - 163.

60. COBHAM A., FRIDSHAL R., & NORTH J.H. "An application of Linear Programming to the minimization of Boolean functions" Research Report RC - 472 IBM, 1961.

61. COBHAM A., FRIDSHAL R., & NORTH J.H. "A statistical study of the minimization of Boolean functions using Integer Programming" Research Report RC - 756, IBM, 1962.

62. BALINSKI M.L. "Fixed cost transportation problems" Nav. Res. Log. Quart. Vol. 11, 1961, p.p. 41 - 54.

63. BALINSKI M.L. & SPIELBERG K. "Methods for Integer Programming: Algebraic, Combinatorial and Enumerative" Chapter 7 in "Progress in Operational Research" Volume III edited by J.S. Aronofsky, John Wiley & Sons, 1969.

64. LEMKE C.E., SALKIN H.M., SPIELBERG K. "Set Covering by Single Branch Enumeration with Linear Programming subproblems" Oper. Res. Vol. 19, p.p. 998 - 1022.

65. CHRISTOFIDES N. & KORMAN S. "A computational survey of methods for the set covering problem" Imperial College, Dept. of Manag. Science, 1973, Report 73/2.

66. PIERCE J.F. & LASKY J.S. "Improved Combinatorial Programming algorithms for a Class of all 0 - 1 Integer Programming problems". Man. Sci. Vol. 19, 1973, p.p. 528 - 543.

67. MANNE A.S. "Plant Location under Economies of Scale-Decentralization and Computation". Man. Sci. Vol. 11, 1964, p.p. 213 - 235.

68. SPIELBERG K. "Algorithms for the simple plant-location problem with some side conditions" Oper. Res. Vol. 17, 1969, p.p. 85 - 111.

69. DAVIS P.S. & RAY T.L. "A Branch & Bound Algorithm for the Capacitated Facilities location problem" Nav. Res. Log. Quart. Vol. 16, 1969, p.p. 331 - 344.

70. CHRISTOFIDES N. "The optimum traversal of a graph" Imperial College, Dept. of Manag. Sci. Report 71/16 November 1971.

71. BELLMORE M. & NEMHAUSER G.L. "The travelling salesman problem: A Survey" Oper. Res. Vol. 16, 1968, p.p. 538 - 558.

72. BEALE E.M.L. "Mathematical Programming in practice" Pitman, 1968.

73. CONWAY R.W., MAXWELL W.L., MILLER L.W. 'Theory of Scheduling' Addison-Wesley, 1967

74. BOWMAN E.H. "The Schedule-Sequencing Problem" Oper. Res. Vol. 7, 1959, p.p. 621 - 624.

75. MANNE A.S. "On the Job Shop Scheduling Problem" in 'Industrial Scheduling' Ed. J. Muth & G.L. Thompson, Prentice Hall, 1963.

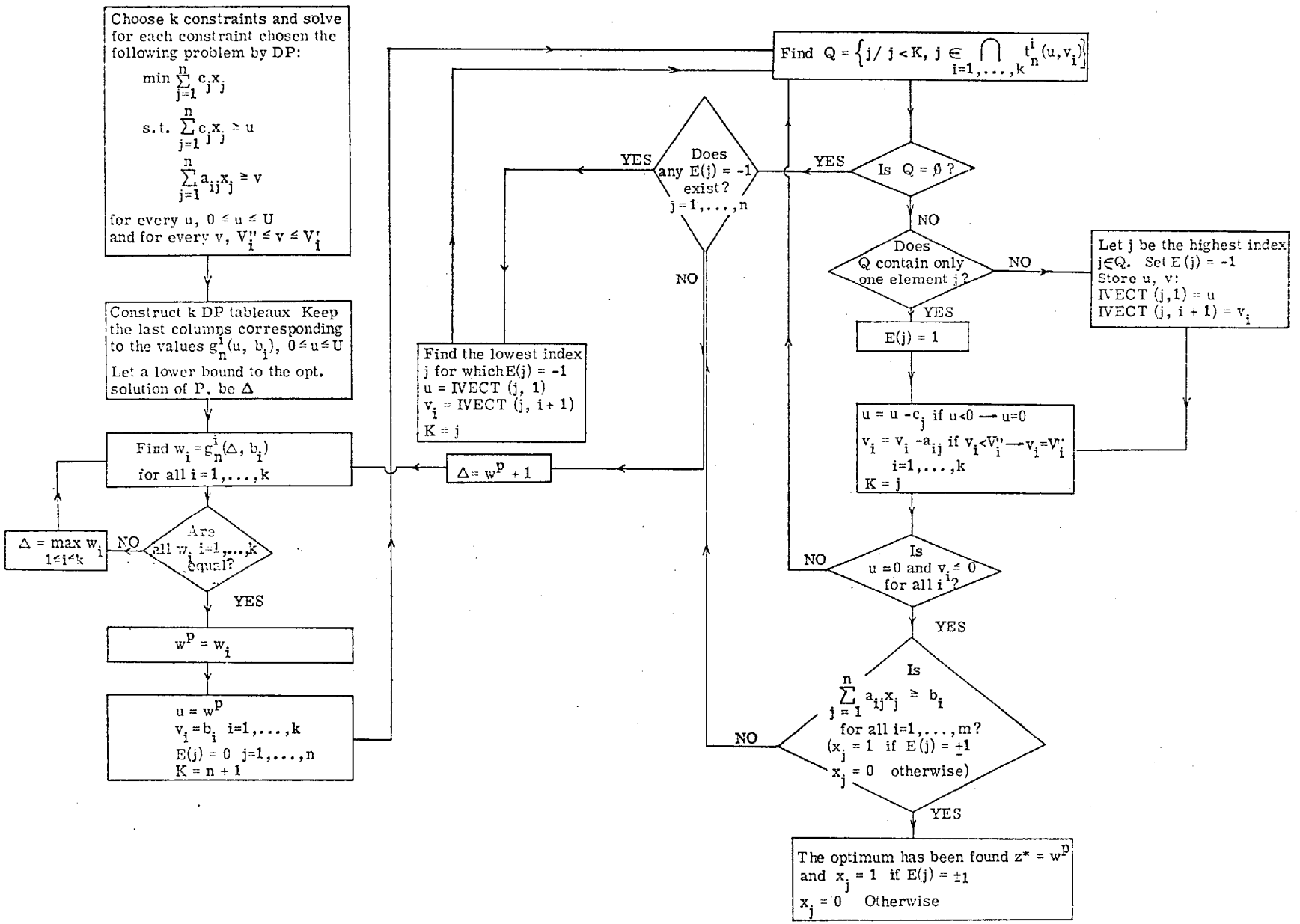
76. WAGNER H.M. "An Integer Linear Programming Model for Machine Scheduling" Nav. Res. Log. Quart. Vol. 6, 1959, p.p. 131 - 140.

77. EDMONDS J. "Maximum matching and a polyhedron with 0 - 1 vertices" in Journal of Research Section B, National Bureau of Standards Vol. 69, 1965, p.p. 125 - 130.

78. WEINGARTNER H.M. "Mathematical Programming and the Analysis of Capital Budgeting Problems", 1963 Prentice Hall.

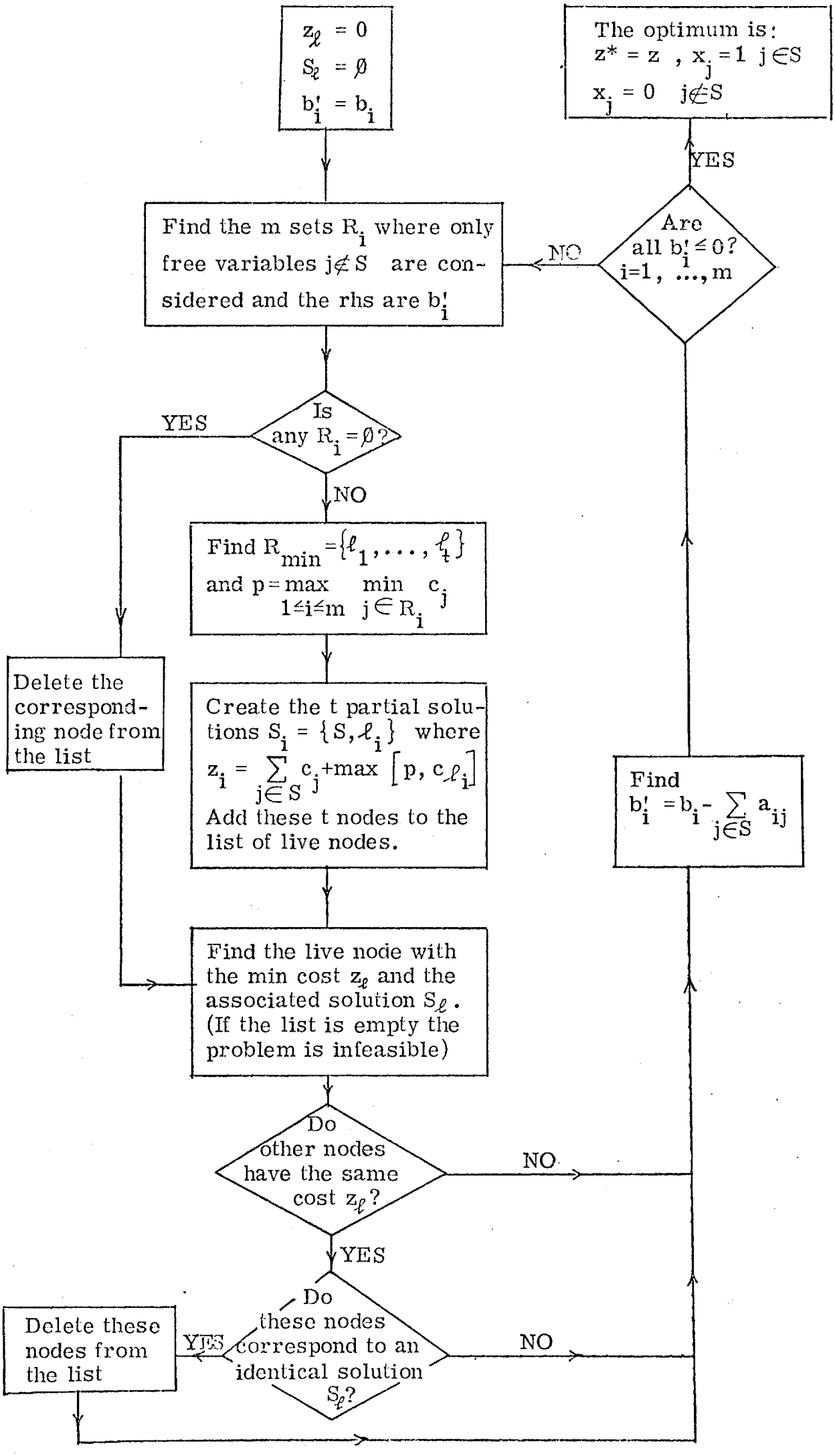
A P P E N D I X I

THE FLOW CHART FOR THE METHOD OF CHAPTER 3


Choose k constraints and solve for each constraint chosen the following problem by DP:
min $\sum_{j=1}^{n} c_j x_j$
s.t. $\sum_{j=1}^{n} c_j x_j \geq u$
$\sum_{j=1}^{n} a_{ij} x_j \geq v$
for every u, $0 \leq u \leq U$
and for every v, $V_i'' \leq v \leq V_i'$
Construct k DP tableaux Keep the last columns corresponding to the values $g_n^i(u, b_i)$, $0 \leq u \leq U$
Let a lower bound to the opt. solution of P, be Δ
Find $w_i = g_n^i(\Delta, b_i)$ for all i = 1,...,k
Are all w_i i=1,...,k equal?
NO
$\Delta = \max_{1 \leq i \leq k} w_i$
YES
$w^p = w_i$
u = w^p
$v_i = b_i$ i=1,...,k
E(j) = 0 j=1,...,n
K = n + 1
Find $Q = \{j / j < K, j \in \bigcap_{i=1,...,k} t_n^i(u, v_i)\}$
Is Q = ∅ ?
YES
NO
Does any E(j) = -1 exist? j=1,...,n
YES
NO
Find the lowest index j for which E(j) = -1
u = IVECT (j, 1)
v_i = IVECT (j, i + 1)
K = j
$\Delta = w^p + 1$
Does Q contain only one element j?
NO
YES
Let j be the highest index j∈Q. Set E(j) = -1
Store u, v:
IVECT (j,1) = u
IVECT (j, i + 1) = v_i
E(j) = 1
$u = u - c_j$ if $u < 0 \rightarrow u = 0$
$v_i = v_i - a_{ij}$ if $v_i < V_i'' \rightarrow v_i = V_i'$
i=1,...,k
K = j
Is u = 0 and $v_i \leq 0$ for all i?
NO
YES
Is $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ for all i=1,...,m?
($x_j = 1$ if E(j) = ±1
$x_j = 0$ otherwise)
NO
YES
The optimum has been found $z^* = w^p$
and $x_j = 1$ if E(j) = ±1
$x_j = 0$ Otherwise

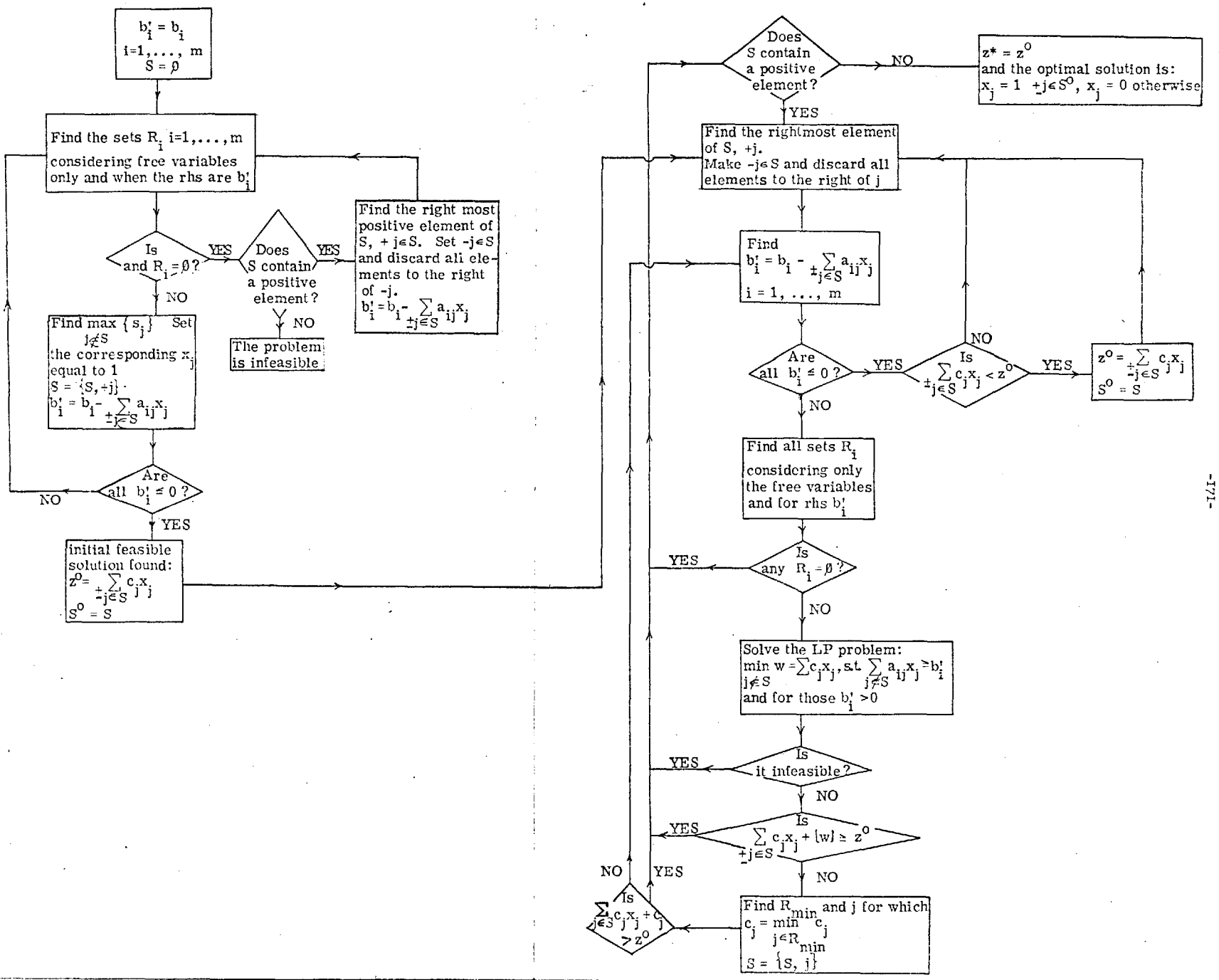
A P P E N D I X II

THE FLOW CHART OF THE NON-BINARY TREE SEARCH METHOD PRESENTED IN CHAPTER 5 (SECTION 5.3.)


$z_\ell = 0$
$S_\ell = \emptyset$
$b'_i = b_i$
Find the m sets R_i where only free variables $j \notin S$ are considered and the rhs are b'_i
Are all $b'_i \leq 0$? $i=1, \ldots, m$
NO
YES
The optimum is: $z^* = z$, $x_j = 1$ $j \in S$ $x_j = 0$ $j \notin S$
Is any $R_i = \emptyset$?
YES
NO
Find $R_{min} = \{\ell_1, \ldots, \ell_t\}$ and $p = \max_{1 \leq i \leq m} \min_{j \in R_i} c_j$
Delete the corresponding node from the list
Create the t partial solutions $S_i = \{S, \ell_i\}$ where $z_i = \sum_{j \in S} c_j + \max [p, c_{\ell_i}]$ Add these t nodes to the list of live nodes.
Find $b'_i = b_i - \sum_{j \in S} a_{ij}$
Find the live node with the min cost z_ℓ and the associated solution S_ℓ. (If the list is empty the problem is infeasible)
Do other nodes have the same cost z_ℓ?
NO
YES
Do these nodes correspond to an identical solution S_ℓ?
YES
NO
Delete these nodes from the list

A P P E N D I X III

THE FLOW CHART OF THE BINARY TREE SEARCH METHOD PRESENTED IN CHAPTER 5 (SECTION 5.4.)


$b_i' = b_i$ $i=1,\ldots,m$ $S=\emptyset$
Find the sets R_i $i=1,\ldots,m$ considering free variables only and when the rhs are b_i'
Is $R_i=\emptyset$ and $R_i \neq \emptyset$?
YES
NO
Does S contain a positive element?
YES
NO
Find the right most positive element of S, $+j \in S$. Set $-j \in S$ and discard all elements to the right of -j. $b_i' = b_i - \sum_{\pm j \in S} a_{ij}x_j$
The problem is infeasible
Find $\max_{j \notin S}\{s_j\}$ Set the corresponding x_j equal to 1 $S = \{S, +j\}$ $b_i' = b_i - \sum_{\pm j \in S} a_{ij}x_j$
Are all $b_i' \leq 0$?
NO
YES
initial feasible solution found: $z^o = \sum_{\pm j \in S} c_j x_j$ $S^o = S$
Does S contain a positive element?
NO
$z^* = z^o$ and the optimal solution is: $x_j = 1$ $\pm j \in S^o$, $x_j = 0$ otherwise
YES
Find the right(most element of S, +j. Make $-j \in S$ and discard all elements to the right of j
Find $b_i' = b_i - \sum_{\pm j \in S} a_{ij}x_j$ $i = 1, \ldots, m$
Are all $b_i' \leq 0$?
YES
NO
Is $\sum_{\pm j \in S} c_j x_j < z^o$?
YES
NO
$z^o = \sum_{\pm j \in S} c_j x_j$ $S^o = S$
Find all sets R_i considering only the free variables and for rhs b_i'
Is any $R_i = \emptyset$?
YES
NO
Solve the LP problem: $\min w = \sum c_j x_j$, s.t. $\sum_{j \notin S} a_{ij}x_j \geq b_i'$ and for those $b_i' > 0$
Is it infeasible?
YES
NO
Is $\sum_{\pm j \in S} c_j x_j + \{w\} \geq z^o$
YES
NO
Find R_{min} and j for which $c_j = \min_{j \in R_{min}} c_j$ $S = \{S, j\}$
Is $\sum_{j \in S} c_j x_j + c_j > z^o$
NO
YES