This book was bound by
Badminton Press
18 Half Croft, Syston, Leicester, LE7 8LD
Telephone: Leicester (0533) 602918.

EXPERIMENTS IN REDUCTION TECHNIQUES

FOR

LINEAR AND INTEGER PROGRAMMING

by

A. N. Ahmed

A doctoral thesis submitted in partial

fulfilment of the requirement for the award of

the degree of Ph.D. of the Loughborough University of Technology

September 1986

TABLE OF CONTENTS

DECLARATION

The work of this thesis follows on from work of other authors on reduction techniques. The applications and extensions of these works are claimed as original and all other parts of the text except where otherwise noted and referenced.

The author also certifies that neither the thesis nor the original work contained herein has been submitted to any other institution for a degree.

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to Dr J M Wilson for his most valuable advice and guidance. His friendship provided an ideal environment for my dissertation research.

Also, grateful acknowledgement to Basrah University, Iraq for financial support.

Finally, I wish to thank my wife Suhair for her patience and understanding over the past three years.

ABSTRACT

This study consisted of evaluating the relative performance to a selection of the most promising size-reduction techniques. Experiments and comparisons were made among these techniques on a series of tested problems to determine their relative efficiency, efficiency versus time etc. Three main new methods were developed by modifiying and extending the previous ones. These methods were also tested and their results are compared with the earlier methods.

CHAPTER I

Introduction

Redundancy in mathematical programming is defined as a characteristic associated with a part of a system which permits deleting that part without any consequence for the system as a whole. After eliminating the redundant characteristics, the system may reduce to a simpler one having the same properties.

Over the past twenty years, investigations of redundancy in linear and integer programs have been made by various authors. In this thesis we have selected the most promising size-reduction techniques and conducted experiments with these on a series of problems obtained from different sources. Secondly, we have extended and improved some of the more efficient methods and have compared them with the earlier methods.

In this chapter, we consider the concept of redundancy, define the forms it may take, and discuss its causes as well as its consequences and its applications. Finally, we present a survey of the literature and the proposed areas of our research.

1.1 REDUNDANCY

A linear programming problem generally consists of an objective function which is to be maximised or minimised subject to a set of constraints. The constraints as well as the objective function are constructed by using a set of variables and appropriate coefficients. Consider the following LPP:

$$\begin{aligned} \text{Max} \quad & Z = CX \\ \text{S.t.} \quad & AX \leq b \\ & X \geq 0 \end{aligned} \qquad \ldots\ (1.1.1.)$$

in which $A \in R^{m \times n}$, $b \in R^m$, $C \in R^n$ and $X \in R^n$. Based on the definition presented in the next chapter, we may refer to constraints and/or variables as being redundant. For example, in the following problem:

$$\begin{aligned} \text{Max} \quad & x_1 + 2x_2 - x_3 \\ \text{S.t.} \quad & 3x_1 + 2x_2 + 2x_3 \leq 20 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

x_3 turns out to be redundant.

We divide redundancy into two general categories. The first type, called absolute redundancy is associated with constraints and/or variables which may be dropped without changing the problem structure in any way. The second type, called relative redundancy is associated with constraints and/or variables which may be dropped without changing certain aspects of the problem, for example the optimal solution.

Redundancy often occurs in practice (already noted in Hoffman (1955)) at various steps in modelling and solving the (programming) problem. In the modelling process of an LP problem, a certain amount of abstraction from the real system is necessary. It is this process which may cause redundancy. "How far should the abstraction go?", "Which aspects should be included and which not?". and so on, naturally, have to be considered and the decision policy used in dealing with these concepts directly affects the inclusion of redundant information in the model. This problem is especially evident as the size of the problem becomes so large that the formulator loses sight of the entire problem. Faced with such a problem, the formulator often includes aspects of the problem which may prove redundant.

Another reason for the occurence of redundancy is the ease of formulation in the modelling process. An example of this is the use of definitional equalities (eg. summing the quantities of raw material that go into a final product).

It is useful in the problem formulation stage to keep in mind the method that will be used in solving the problem as well as the purpose of formulating and solving the problem, since sometimes there is a distinction between problem formulators and problem solvers. Some techniques require the specification of extra information, which may cause redundancy. These techniques including all cutting plane methods so far linear (Dantzig-Wolfe decomposition, dual form, Dantzig and Wolfe (1960)), integer (Gomory (1958)), mixed integer (Benders (1962)) and convex nonlinear programming (Kelly (1963)) and all Branch-and-Bound methods (eg. Garfinkel and Nemhauser (1972)). In parametric programming (eg. Gal (1979)) redundant constraints may become nonredundant and vice-versa (see Gal (1975)). Further details are included in Karwan _et al_ (1983).

A direct consequence of redundancy in LP programs is the increase in size. The larger size has two major disadvantages. First, the problem may be so large that conventional computer programs may not be able to solve the problem. Secondly, the solution process may be more difficult and more expensive. The higher cost is associated with computational effort on redundant information which could otherwise be unnecessary. Regarding the size of the problem, more storage space will be required which may be critical if the problem cannot be solved by an in-core code.

Regardless of the size of the problem, redundant constraints may cause degeneracy. This degeneracy in turn may result in degenerate pivot steps (ie. steps in which the objective function value does not improve). Such occurance for a number of consecutive pivots is called "near cycling" (see Thompson et al (1963)). Although the relation between redundancy and cycling is not yet fully understood, Zionts (1965) and Telgen (1980) conjecture that cycling is possible only by virtue of redundancy.

In addition to the computational difficulties caused, redundancy tends to conceal certain information and possibilities, namely knowing that something is redundant might lead to a different decision. For example, in a production planning problem, if a capacity constraint is redundant, it generally indicates excess capacity which might be used in some other way.

The consequences of redundancy are not all disadvantageous. The best example of this is transforming an LP problem by adding constraints and variables to a transportation problem (see Charnes and Cooper (1961). As is well known, the latter problem is much more easily solved than the general LP problem. Other examples of the advantages of redundancy are included in Karwan et al (1983). However, it is the author's conviction that the disadvantages of redundancy generally outnumber its advantages.

Now, once a problem is formulated, a question will arise, whether it is worthwhile to implement the size reduction techniques or not. Actually, certain factors such as the costs in implementing such techniques and the derived benefits should be determined. However, there is always a positive result from identifying redundancy, but there are cases in integer programming problems where the presence of redundant constraints can accelerate the solution process. The identification of redundancy in a problem is just as difficult as solving the linear programming problem itself, where it is "easy" in linear constraints, but it is "hard" if we have to take into account integrality constraints.

Size-reduction techniques have other desirable properties when used to solve certain linear programming problems. For example, in Zionts (1965) certain problems are solved for which an ordinary simplex method computer code did not produce correct results (even with repeated runs) because of the accumulation of round-off error. In addition to that, size-reduction techniques can provide a means for altering (possibly improving) particular mathematical programming solution methods.

The application of size-reduction techniques to mathematical programming problems in general depends on the specific goal of the techniques and the type of problem. For example, a Branch-and-Bound procedure for solving integer linear programming problems may require the LP relaxation to be solved many times. Thus, identifying and removing a redundant constraint from the original integer linear programming may result in a significant decrease in the overall solution time.

Another example is an LP problem in which one set of constraints is changed regularly and the other set remains the same (eg. Generalised Upper Bound (GUB) constraints). Then, it may prove economical to determine whether any of the fixed constraints are redundant. This has two advantages. One is that the removal of such a redundant constraint has a multiple effect in reducing the computation time. Secondly, the modeller may want to replace the redundant constraint with other constraints which were left out due to the large size of the problem.

In addition to reducing the size of the problems, the removal of redundant constraints may remove the computational complexities associated with certain problems. For example, removing the redundant constraints may prevent a problem from cycling (see Zionts (1965) and Telgen (1979) for more details).

Other applications include obtaining the lower and upper bounds on variables from the problem structure (eg. Williams' method (1983)). These bounds may be of major interest to the problem formulator.

1.2 A SURVEY OF THE LITERATURE

A number of interesting results were derived for solvability and the geometric properties of a system of linear constraints without considering the constraints individually.

Fourier (1926) and Motzkin (1936) , presented an elimination method which solves the LP problem directly. Except for solving very small problems or problems of a special structure, the method is rather cumbersome.

Unlike the elimination method, Charnes et al (1953), presented the ratio-analysis method, which has been used only for problems which possess certain structures.

Wolfe (1955) describes a method to reduce a problem to a "simplest problem in standard form".

Dantzig (1955) suggests using a prior knowledge of linear programming problem to predict the solution. Some constraints can be anticipated to be non-binding and (equivalently), certain activities are anticipated to be in the optimum solution. The slacks of the non-binding constraints and these essential variables can be brought into the basis. The constraints in which they are basic, together with the variables, can then be dropped from the problem. When the optimum solution is found, these assumptions can be checked, and, if they are violated, the constraints reintroduced and more iterations taken. If the number of errors in anticipating nonbinding constraints is relatively small, great savings are achieved. If the variables are known to be present in the optimum solution, then no additional iterations need to be made. A similar approach is due to Thompson and Sethi (1983) (presented in this thesis). Their technique uses mathematical information to make a prediction about the solution by defining a candidate constraint and checking this prediction at every step, incorporating a modification of the simplex method in which only the current candidate constraints are updated. Thompson and Sethi (1984), also presented another way to take advantage of the fact that most constraints are never candidates. They begin by solving a relaxed linear program consisting of the constraints of the original problem which are initially candidates. Also they introduce the idea of a probe, that is, a line segment joining two vectors for the primal problem, using it to identify a most-violated constraint, which is added to the relaxed problem which is solved again. Their computational experiments indicate that time saving of 50% - 80% over the simplex method could be obtained by this method, which they call PAPA, the Pivot and Probe Algorithm.

From the early 1960's systems were studied from the redundancy point of view, since it is hardly disputed that redundancy exists in practical mathematical problems. Before proceeding, we note that the redundancy discussed by some authors used the terms "trivial" (Boot (1962)), "superfluous" (Thompson et al (1966)), "irrelevent" (Matthesis (1973)), "inessential" (Zeleny (1974)), essentially all mean "redundant".

Balas (1962), identifies nonbinding constraints and extraneous variables on the basis of "dominance" relationships among rows and columns. Balinski (1961), gives an algorithm to determine all extreme points of the polyhydron to identify redundant constraints. Since that path* is quite large depending on the order of introduction of hyperplanes that generate the path, and the number of extreme points grows exponentially with the size of the problem, and so this approach is very cumbersome for large problems. The same basic approach was followed by Shefi (1969) (see also Luenberger (1973)), who developed another algorithm for determining all extreme points. He also proposed certain minimality properties for systems

* Convex path solution.

of linear constraints. However, Telgen (1981), later developed a minimal representation theory in which Shefi's proposals could be considered as special cases.

Mattheiss ((1973) and (1983)) implements a vertex finding algorithm to enumerate the vertices associated with a system of linear inequalities. At each vertex, the active constraints are nonredundant (assuming there is no degeneracy). Therefore, when the enumeration process is completed the unidentified constraints are labelled as redundant. The number of vertices was shown to be significantly less than the number of vertices of the original space (see Mattheiss and Schmidt (1980)). The vertices are enumerated by a variant of the simplex method noting active constraints, which are nonredundant. This method was not efficient in practice, because a large number of vertices had to be processed, each vertex corresponding to a basic feasible solution for which the usual simplex tableau had to be constructed, the process having to be repeated until no new unlabelled vertex was found.

Greenberg (1975), develops a method for determining redundant inequlities and all solutions to convex polyhedra. In his algorithm, he is seeking to eliminate the extraneous solutions obtained when using the Motzkin method (Motzkin (1936) and Motzkin et al (1953)) for solving homogeneous solutions, which are possible to obtain in some situations, where the condition in one his theorems is necessary but not sufficient, as was pointed out in an example by Shermain (1977) Later it was corrected in Dyer and Proll (1980). A computational comparison by Dyer and Proll (1977) showed that Mattheiss' method generally outperformed Greenberg's method.

Boot (1962), was the first published paper related entirely to redundancy. His method provides algebraic tests on the solution space which makes it possible to determine whether or not a variable is extraneous or a constraint is redundant. It is based on checking the feasibility of the LP problem obtained when one of the constraints is violated by a small amount. If a feasible solution to the peturbed problem can be found, then the violated constraint is nonredundant. Otherwise, the constraint is redundant. The major disadvantage of this approach is that systems of linear constraints have to be checked for feasibility in order to check a constraint for redundancy. Therefore, the computations are much too laborious, and although the method is interesting, it is too cumbersome to be of any general use. Zionts (1965) and Thompson et al (1966) gave a simplified version of Boot's technique, that instead of violating a constraint and eliminating a variable, only sets the slack variable to $-\epsilon$ and checks for a feasible solution. But, since there is no known simple way of checking a constraint set to determine feasibility, this simplified version still faces the same difficulty.

Dale O.Cooper (1962), presents four methods for initially reducing the size of linear programming problems. One of them determines certain variables that will be strictly positive in an optimal solution. The reamaining three methods are heuristic in nature, and require making intelligent guesses as to which variables are likely to be basic or nonbasic in an optimal solution. These guesses are subsequently revised if they are false.

Zionts (1965) developed two methods. The first method is called the

Geometric Definition method which is of major importance to the concept of size-reduction in LP problems. The basic feature of this method is the establishment of situations where several simple sign tests on any row or column of the simplex tableau show that redundancy can be recognized immediately without any further computations. The method may be employed at the beginning of a linear porogramming solution porcedure, or it may also be employed during the course of solving a linear programming problem.

The second method is the heuristic method (or convex path method) based on a theoretical development for which certain sufficiency conditions cannot always be assumed to hold. The heuristic assumes that these conditions do hold. It then fixes certain variables (ie. it avoids removing them from the solution basis) on the supposition that they will form part of an optimal solution. In a similar way, certain other variables are forced to remain out of the solution basis. In either case, whether variables are fixed or whether they are forced to remain out, both types of variables are completely ignored in subsequent iterations. Once an apparently final solution to the problem (either optimal, infeasible or unbounded) has been found, the ignored variables are restored. Checks are then performed for optimality and feasibility and if these are not satisfied, then further iterations are taken if necessary. Obviously, if the required sufficiency conditions could be guaranteed to hold, the method would not be a heuristic, and the further iterations would never be needed.

The results of the Geometric Definition method were implemented by many researchers. Lisy (1971) used these simple sign tests to identify all redundant constraints in an LP problem. Zionts (1972), also extended some concepts of redundancy to integer programming. Rubin (1973) extended some of the results of Thompson et al (1966), to integer programming by modifying theorems and their proofs. Gal ((1975) and (1978)) elaborated on this approach by adding new rules for identifying nonredundant constraints as well. Telgen (1981) extended the approach by considering degenerate cases including redundant constraints which pass through an extreme point. Also, Rubin (1983) developed another version of Telgen's method to identify all redundant constraints. Zionts and Wallenius (1980), presented a new version based on the same concepts of Zionts (1965), to identify all redundant constraints. Karwan <u>et al</u> (1983) presented full details about the above four methods and their comparison in experimental tests, and mentions them as Sign Tests methods.

A number of other researchers have addressed the possibility of redundancy by virtue of a structural constraint and nonnegativity constraints on all variables. Llewellyn (1964), presented rules (see also Zeleny (1974)) to recognise this situation. These rules were generalized by Eckhardt (1971). However, Telgen (1979) showed that the rules are valid only for positive coefficients and other very special cases.

A totally different approach was developed by Boneh and Golan (1979). The method is based on determining the constraints having the closest distance from an interior point in a randomly chosen direction. Such constraints are clearly nonredundant. Then, after a large number of trials all constraints which have not been hit are declared to be redundant. The latter results are not necessarily correct (ie. a nonredundant constraint may not be hit within the given number of trials). Telgen (1981) suggested the use of co-ordinate directions instead of randomly chosen directions. We will present Boneh and Golan's method in this thesis.

Lotfi (1981), presented three methods, the first of which is called the "Extended Sign Test", which is an improved version of the earlier sign test methods. The second method is called "Hybrid" which is combined with the Extended Sign Test method and Co-ordinate Direction method (the improvement of Boneh and Golan's method using Telgen's suggestion). The third method is called "Reduce" and applies the Extended Sign test method to both the primal and dual problem while solving the problem. All three methods are presented in detail in this thesis.

Brearley et al (1975) described the REDUCE option of many commercial mathematical programming packages, which is essentially an extension of the "Geometric Definition Method" of Zionts (1965), which was developed independantly. The extended geometric method is based on a collection of theorems which make it possible to compute bounds on primal and dual variables from the problem structure. Then, given these bounds, extraneous variables and nonbinding constraints are identified. The process is repeated until no further reduction is possible. More details given by Williams (1983) are presented in this thesis.

Klein and Holm (1975) suggest a similar approach utilizing the complementary slackness theorem of linear programming in combination with bounds on the primal and dual variables to identify extraneous variables and nonbinding constraints. In the absence of these bounds, a method is proposed for calculating them. The problem however, must have a special structure. All coefficients of the matrix must be nonnegative, and all inequalities must be less than or equal to ($\leq$). The details of the method are presented in this thesis.

A number of papers discussed redundancy in large scale problems.

Bradley, Brown and Graves' (1983) discussed automatic detection and exploitation of structural redundancy in large scale linear programming (as well as mixed integer programming) problmes, where such redundancy represents an embedded special structure which can give significant insight to the model proponent as well as greatly reduce solution effort. Various identification techniques for economic application to large problems were developed and tested. The details of these techniques are presented in this thesis.

Finally, some other papers relate only to the class of (0-1) linear programs. Wilson (1983), developed a procedure to reduce the set of (0-1) linear inequalities to a smaller set by examining pairs of inequalities and then deriving an implicit inequality, based on the fact that, any explicity (0-1) linear inequality may be expressed as a set of $k(k>1)$ implicit inequalities with unit coefficients in the A matrix.

Crowder et al (1983) presented a method which included problem preprocessing and constraint generation, to get the optimal solution of sparse large-scale (0-1) linear programming problems. In problem preprocessing, variables could be fixed at either 0 or 1, and inactive constraints could be determined. The constraint generation is performed by generating cutting-planes which are satisfied by (0-1) solutions of the problems. The details of the method are presented in this thesis.

1.3 PROPOSED RESEARCH

The objective of this thesis is to ascertain how successfully Size-reduction techniques could be implemented in Commercial Mathematical Programming Packages. By studying the most promising techniques, and improving some of them, new ones are developed which are more practically efficient and economical in their implementation.

The thesis consists of seven chapters. The present chapter provides an introduction to the concept of redundancy, its applications and a survey of the literature.

Chapter II intends to present the definitions, notation, and some common theorems which are frequently used by the methods presented in the thesis. Nine selected size-reduction techniques to be studied are presented in detail in chapter III.

New improvements to most of these selected methods are presented. Chapter IV contains two improvements in methods for general linear programming problems. Chapter V contains an improved method to reduce general integer problems and its implication to the "Dynamic-Presolve" procedure, which is a feature of the SCICONIC package. Then, a procedure to reduce subproblems having Special Order Sets (SOS) is presented.

Chapter VI presents the programming aspects of some of the methods presented in chapter III and our improvements to methods. A discussion and comparison based on the experimental results of our improvements methods and the earlier method follows.

Finally, conclusions and recommendations for future research are discussed in chapter VII.

CHAPTER II

In this chapter we present definitions and notation that will be used throughout this thesis, as well as some common theorems which are frequently used by the methods to be discussed.

2.1 MATHEMATICAL FOUNDATION AND NOTATION

We consider the follwing linear programming problem:

$$\text{Max} \quad Z = CX \qquad \ldots (2.1.1)$$

$$\text{S.t.} \quad AX \leq b$$

$$X \geq 0 \qquad \ldots (2.1.2)$$

in which $A \epsilon R^{mxn}$, $b \epsilon R^m$, $X \epsilon R^n$ and $C \epsilon R^n$.

We denote $S = (S_1, \ldots, S_m, S_{m+1}, \ldots, S_{m+n})$, where the set $(S_1, \ldots, S_m)$ contains the slack variables of the structural constraints, and the set $(S_{m+1}, \ldots, S_{m+n})$ contains the slack variables of the nonnegativity constraints.

Adding the slack variables of structural constraints, pre-multiplying by the inverse of an appropriate basis, we partition $(A : I)$ into $(B : N)$ and redefine the variables (both slacks and structural variables) as x_j^N or x_j^B according to their status (N for nonbasic and B for basic), yielding the equivalent system

$$\left[B^{-1}N \quad I\right] \begin{bmatrix} x^N \\ x^B \end{bmatrix} = B^{-1}b$$

with x^N, $x^B \geq 0$

The matrix $B^{-1}N$ is usually referred to as a contracted simplex tableau (Dantzig (1963)). We refer to the elements of $B^{-1}N$ as a_{ij} and denote the "updated right-hand side" elements by b_i.

The feasible region corresponding to the system of linear constraints (2.1.2) is defined as:

$$F_L = \{X \epsilon R^n | AX \leq b\} \qquad \ldots (2.1.3)$$

and thorughout it is assumed that the feasible region exists, ie. $F_L \neq \emptyset$.

Also we define the set:

$$F_L(k) = \{X \epsilon R^n / A_i x \leq b_i, \ i \neq k \text{ and } x \geq 0 \} \qquad \ldots (2.1.4)$$

where A_i denotes to the ith row of A.

Analogously, we define F_I and $F_I(k)$ with the additional restriction that x be integral.

Definition 2.1

The constraint $A_k x \leq b_k$ is redundant in LP(IP) if

$$F_L = F_L(k) \ (F_I = F_I(k)).$$

The above definition may be utilised for the nonoegativity constraint $x_j \geq 0$ as well. Note that $F_L(k) = F_L$ if and only if $A_k x \leq b_k$ for all $x \epsilon F_L(k)$; hence an equivalent definition in which

$$s_k(x) = b_k - A_k x \qquad \ldots (2.1.5)$$

makes it easy to see that $A_k x \leq b_k$ is redundant in the system of linear constraints (2.2), if and only if

$$\hat{s}_k = \min \{s_k(x) \mid X \in S_k\} \geq 0 \qquad \ldots (2.1.6)$$

This definition is especially useful because we may consider every variable as a slack (the structural variables are the slacks of their nonnegativity constraints).

Now, if $\hat{s}_k = 0$, then the constraint is termed <u>weakly redundant</u>, if $\hat{s}_k > 0$ it is termed <u>strongly redundant</u>.

Throughout, we will use the term redundant referring to both strong and weak redundancy and will refer to each type explicitly when the need arises. The following example clarifies the concepts of strong and weak redundancy. Consider

$$\begin{array}{rcrl}
x_1 + x_2 & \leq & 4 & \quad (1) \\
2x_1 + x_2 & \leq & 6 & \quad (2) \\
x_1 - x_2 & \leq & 3 & \quad (3) \\
x_1 & \leq & 2 & \quad (4) \\
- x_2 & \leq & -1 & \quad (5) \\
x_1 & \geq & 0 & \quad (6) \\
x_2 & \geq & 0 & \quad (7)
\end{array} \qquad \ldots (2.1.7)$$

which is presented in Figure 2.1. In the above system of inequalities, constraint (3) and the nonnegativity constraint (7) are strongly redundant, constraint (2) is weakly redundant.

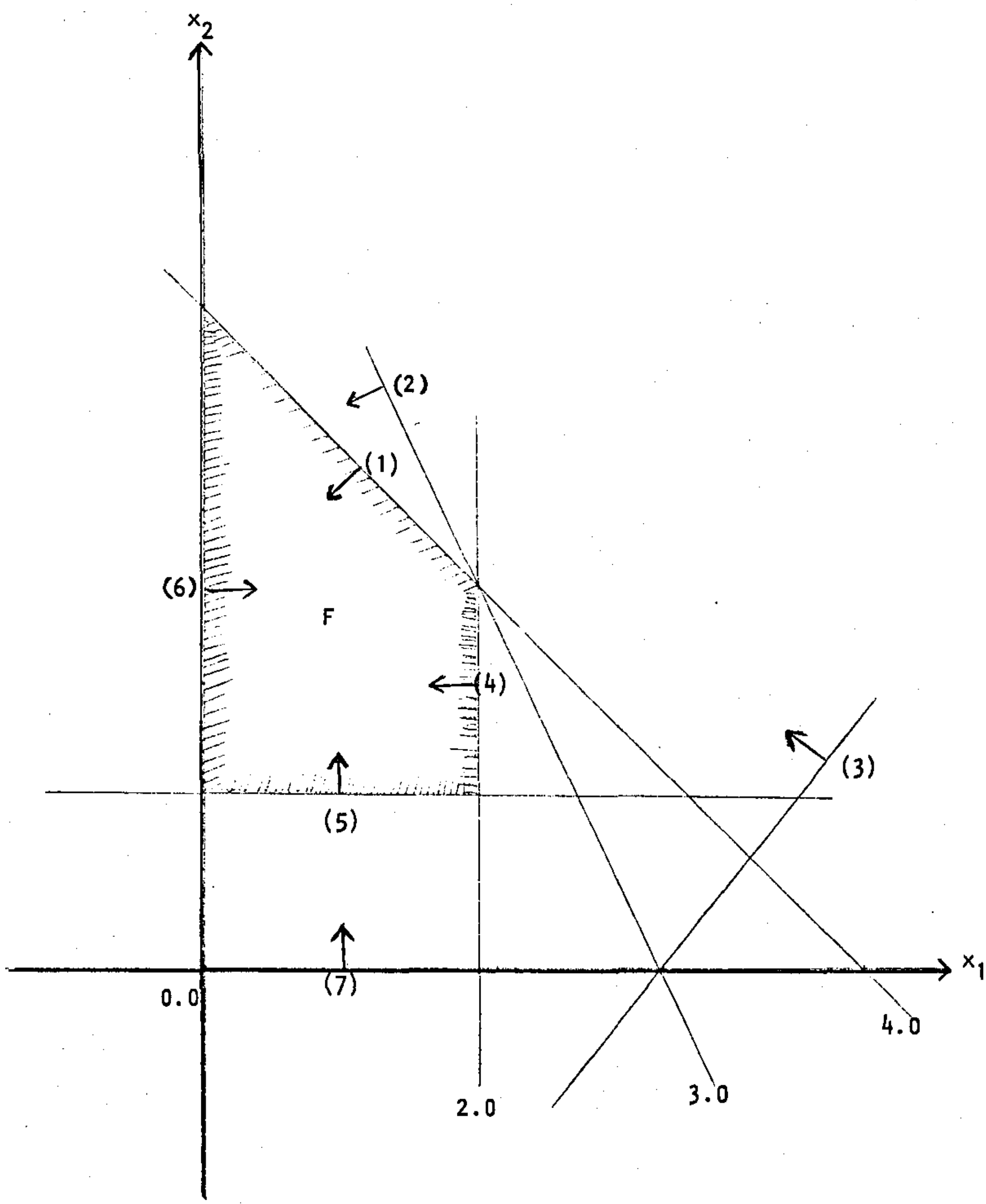

Figure 2.1 Feasible Region for system (2.7)

Until now, we have considered mainly the system of linear constraints (2.2). There are other kinds of constraints which are called "non-redundant" constraints and we subdivide these into two groups of "non-binding" and "binding" constraints, for which we need to introduce the objective function (2.1.1) into the system (2.1.2.)

Definition 2.2

A constraint is nonbinding if and only if it is nonredundant and its associated slack variable is positive in every optimal solution.

Definition 2.3

A constraint is binding if and only if it is neither redundant nor nonbinding.

A "binding" constraint is termed "strongly" if its associated slack variable is zero at every optimal solution; if it is zero in some but not all optimal solutions, the constraint is termed "weakly binding".

For example, suppose the objective in Figure 2.1 is parallel to constraint (4) and an increasing factor of x_1. Then, constraint (4) is strongly binding, constraints (1) and (5) are weakly binding, while the only non-binding constraint is the nonnegativity constraint (6).

It should be noted that dropping the redundant constraints does not change the feasible region and of course the set of optimal solutions remains the same. Dropping the nonbinding constraints increases the feasible solution region but not the set of optimal solutions.

Looking at the results of redundancy from the duality view point, one could see that in any solution to the linear programming problem (and thus optimal solutions too) a redundant constraint in the primal problem which implies by the complementary slackness theorem (see eg. Dantzig (1963)) that the corresponding dual variable equals zero in the optimal solution and we can delete such a variable but the feasible region of the dual problem will not be increased. We refer to such a variable as extraneous. In order to define the extraneous variables mathematically, let us present the following notation:

$$F_j^* = \{X^* \in R^n | AX^* \leq b, \; x_j \notin X^* \text{ and } X^* \subseteq X\} \qquad \ldots (2.1.8)$$

Definition 2.4

A variable x_j is extraneous in LP(IP) if and only if

$$F_j^* = F_L \; (F_j^* = F_I)$$

If x_j is zero in every optimal solution, then x_j is strongly extraneous. If it is zero in some but not all optimal solutions, then it is weakly extraneous. Note that the status of a redundant constraint is not changed for a different choice of the objective function. However, a different choice of the right-hand side may change the status of the extraneity of the variable.

As with nonredundant constraints, we refer to variables which are not extraneous as nonextraneous, and these may be divided further into free, inessential and essential variables. Karwan et al. (1983) gives further details.

2.2 SOME COMMON THEOREMS

The following theorems are frequently used by most methods presented in this thesis, to identify the redundancy status of constraints (and variables if applied to the dual problem). Therefore, to avoid repetition, we present them in this section. For associated theorems (if any), these will be discussed as part of a method itself. Also, throughout this thesis we will refer to the application of each theorem as a "Test" with its corresponding number (eg. by test one we mean the application of theorem one).

Theorem 2.1 Gal (1975)

A constraint is redundant if and only if its associated slack variable s_k has the property:
$s_k = x_r^B$ in a basic feasible solution in which $a_{rj} \leq 0$ for all $j = 1, \ldots, n$.

Theorem 2.2 Zionts (1965), Thompson *et al*. (1966)

A constraint is redundant if its associated slack variable S_k has the property:
$S_k = x_p^N$ in a basic solution in which for some i, $b_i \leq 0$, $a_{ij} \geq 0$ for all $j = 1, \ldots, n$, $j \neq p$ and $a_{ip} < 0$.

Theorem 2.3 Telgen (1979), Zionts and Wallenius (1980)

A constraint is not redundant if its associated slack variable S_k has the property:
$S_k = x_p^N$ in a basic feasible solution in which $a_{ip} \geq 0$ for all i with $b_i = 0$.

Theorem 2.4 Rubin (1972), Mattheiss (1973) and Gal (1975)

A constraint is non redundant if its associated slack variable is nonbasic in a nondegenerate basic feasible solution.

Theorem 2.5 Telgen (1977)

A constraint is not redundant if its associated slack variable S_k has the property:

$S_k = x_r^B$ in some basic feasible solution which

$b_r/a_{rs} = \min \{b_i/a_{is} | a_{is} > 0\}$ is unique for some s.

Proofs of these theorems are contained in the appropriate references.

CHAPTER III

In this chapter we will present the details of the most promising size-reduction techniques. These methods are classified according to their main objectives. Namely, Boneh and Golan's, Lotif's (Extended Sign Tests, and Hybrid) methods are categorised as one group which attempts to identify redundant (or equivalently non-redundant constraints). The second group consists of Klein and Holm's, Williams' and Lotfi's (Reduce) methods, which attempts to identify redundant and nonbinding constraints as well as extraneous variables. The third group consists of Thompson and Sethi's method which uses a variation of the simplex method. Finally, the fourth group consists of the methods of Bradley et al. and Crowder et al. which attempt to discuss redundancy in large-scale problems.

3.1 GROUP ONE METHODS

3.1.1 Boneh and Golan's method

Boneh (1983), describes a probabilistic method, developed by Boneh and Golan which attempts to identify nonredundant constraints. Then, after sufficiently many iterations, the remaining unidentified constraints are declared as redundant (possibly erroneously). The method is based on the premise that for a given non-empty polyhedral set, the closest constraints to an interior point are non-redundant. In order to identify such constraints, first an interior point is determined. Then, a random direction is generated and the distance between the interior point and each constraint (along the random direction) is computed. The constraints with smallest positive distance and the largest negative distance are closest constraints to the interior point (one on each side). Hence, these constraints are labelled as non-redundant. For the next iteration the interior point is moved uniformly along the random direction (within the feasible region) and a new random direction is generated. This process is repeated until a certain stopping criterion (eg. certain number of iterations) is satisfied. If so, the non-redundant constraints identified (accurately) are output along with the remaining constraints labelled as redundant (possibly erroneously).

The algorithm requires two initial steps. In the first step, all the constraints of Type "$\leq$" are changed to "$\geq$", and the problem becomes the general form:

$$A_i X \geq b_i \qquad i = 1, \ldots, m \qquad \ldots (3.1.1.1)$$

The second initial step, is to determine an interior feasible point for the system (3.1.1), either by generating some arbitrary point X^o and check for feasibility, or generating a random direction and move X^o along this direction to a point which satisfies more constraints.

The basic approach is to evaluate and (if necessary) sort the intersection points of a specified straight line in n-dimensional space with each and every one of the constraints. Therefore, if $X^o \in R^n$, $d \in R^n$ are the interior point and the direction, respectively, the scalar $t \in R^1$ is the parameter of the straight line passing through the point X^o in the direction d, then t_i is evaluated by the following equation:

$$t_i = \frac{b_i - A_i X^o}{A_i d} \qquad (i=1, \ldots, m+n) \qquad \ldots (3.1.1.2)$$

The algorithm has two options for generating straight lines, randomly generated and co-ordinate direction as suggested by Telgen (1981). In the co-ordinate direction the above computation in (3.1.1.2) could be reduced more, and the equation (3.1.1.2) reduces to:

$$t_i = \frac{b_i - A_i X^0}{a_{ij}} \qquad (i=1, \ldots, m+n) \qquad \ldots (3.1.1.3)$$

In both options, the algorithm generates a new interior point X^1 as follows:

$$X^1 = X^0 + [t_\ell + \mu(t_k - t_\ell)]\, d \qquad \ldots (3.1.1.4)$$

where t_ℓ, t_k are the distances associated with the closest constraints to X^0 (one on each side) and μ is a random uniform deviate in the unit interval. Clearly, when d is a co-ordinate direction, the equation (3.1.1.4) may be updated at each successive iteration, that is,

$$X^1 = X^0 + [t_\ell + \mu\, (t_k - t_\ell)] \qquad \ldots (3.1.1.5)$$

Now, we present the main steps in Boneh and Golan's method (note that initially all of the constraints are labelled as redundant).

Step 1: Generate a random direction $d \in R^n$ with $d_j \sim N(0,1)$

Step 2: Compute

$$t_i = \frac{b_i - A_i X^0}{A_i d} \qquad (i=1, \ldots, m+n)$$

Step 3: Determine $t_k = \min_i \{t_i | t_i > 0\}$ and $t_\ell = \max_i \{t_i | t_i < 0\}$

(note that $b_i \neq 0 \ \forall i$ since x^0 is not allowed to be a boundary point), label constraints k and ℓ as non-redundant. If all constraints have been identified as non-redundant, stop, otherwise go to step 4.

Step 4: Generate a random multiplier $\mu \in (0,1)$ and compute:

$$X^1 = X^0 + [\, t_\ell + \mu\,(t_k - t_\ell)]\, d$$

(note that X^0 is moving along the line $X^0 + td$), relabel X^1 as X^0 and go to step 5.

Step 5: Stop if one or both of the following conditions are met:

(a) Total number has exceeded 10(mxn) log (m+n)

(b) The number of consecutive unsuccessful iterations (iterations in which no new constraints are identified) is more than 2(m+n). Otherwise go to step 1.

Now we present a numerical example to illustrate the use of Boneh and Golan's method. Consider the following system:

$$\begin{array}{rcll} -x_1 + x_2 & \leq & 1 & (1) \\ x_1 + x_2 & \leq & 3 & (2) \\ x_1 & \leq & 2 & (3) \\ 4x_1 + 3x_2 & \leq & 12 & (4) \\ x_1 & \geq & 0 & (5) \\ x_2 & \geq & 0 & (6) \end{array} \qquad \ldots (3.1.1.6)$$

which is shown in Fig. (3.1)

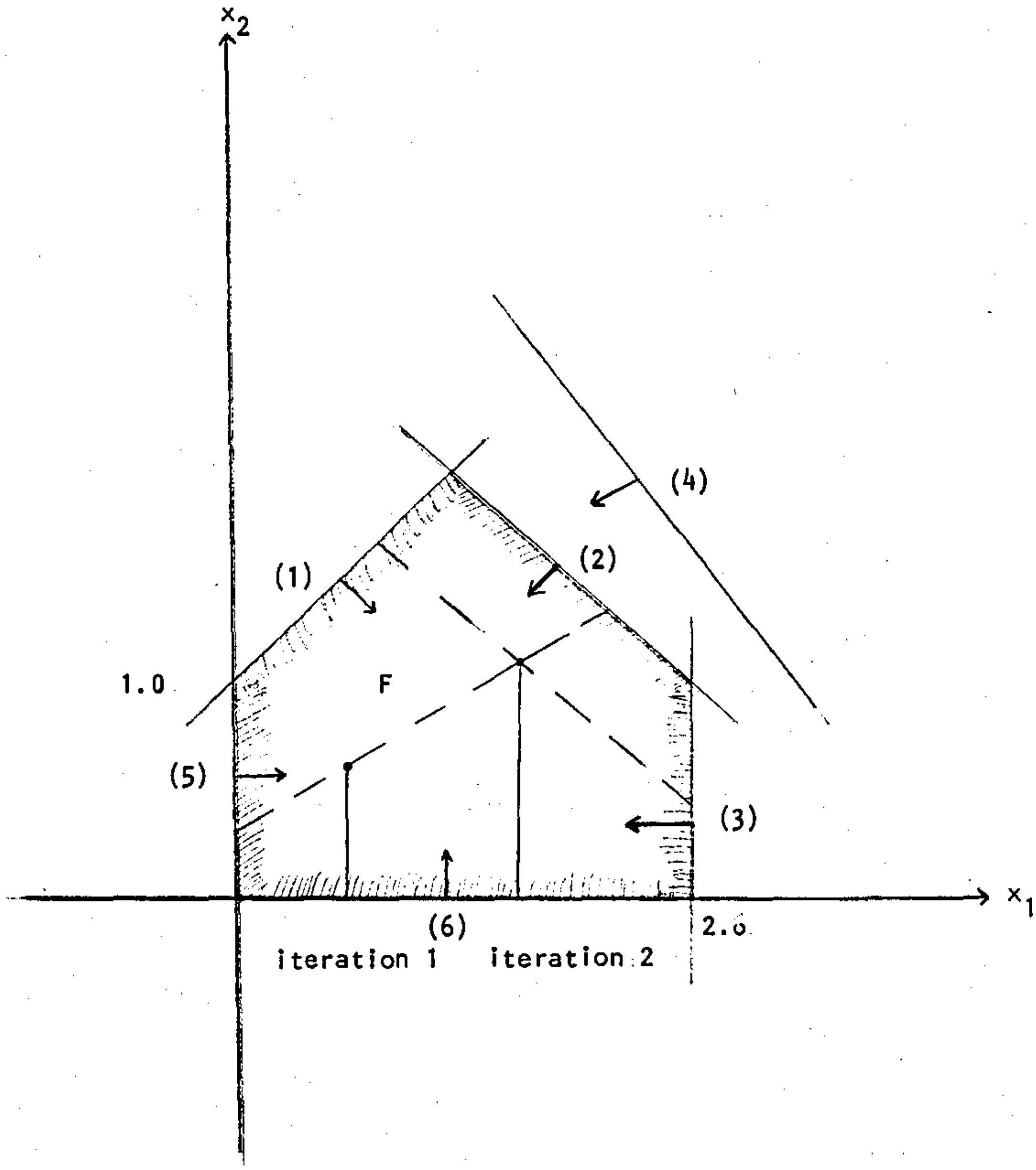

Figure 3.1 Feasible Region for System (3.1.1.6)

Initial Step (1): Changing the direction of the inequalities (1) through (4) and adding the non-negativity constraints, we have:

$$x_1 - x_2 \geq -1 \qquad (1)$$
$$-x_1 - x_2 \geq -3 \qquad (2)$$
$$-x_1 \geq -2 \qquad (3)$$
$$-4x_1 - 3x_1 \geq -12 \qquad (4)$$
$$x_1 \geq 0 \qquad (5)$$
$$x_2 \geq 0 \qquad (6)$$

Initial Step (2): Let $X^0 = (0.5, 0.5)$ be an interior feasible point. The following are two representative iterations of the main steps:

Step (1): Let $d = (o.2, 0.1)$

Step (2): $t = (-10, 6.7, 7.5, 7.7, -2.5, -5)$

Step (3): $t_k = 6.7 \quad k = 2, \quad t_\ell = -2.5, \quad \ell = 5$:

constraints 2 and 5 are non-redundant.

Step (4): Let $\mu = 0.7$, $X^1 = (1.3, 0.9)$, $X^0 = (1.3, 0.9)$

Step (1): Let $d = (0.3, 0.2)$

Step (2): $t = (2.8, -8.0, -2.3, -6.8, 4.3, -4.5)$

Step (3): $t_k = 2.8 \quad k = 1, \; t_\ell = -2.3, \quad \ell = 3$:

constraints 1 and 3 are non-redundant

Step (4): Let $\mu = 0.2$, $X^1 = (1.7, 0.6)$, $X^0 = (1.7, 0.6)$

The above steps are repeated until a stopping criterion is satisfied in which case the remaining unidentified constraints are declared as redundant.

3.1.2 Lotfi's Methods

Lotfi (1981) presented two improvement methods within this group.

Extended Sign Test Method

This method is an improved version of the earlier sign test method. The method is developed from some modifications (some tests are eliminated during the course of testing process) to the earlier sign methods. Since there is no new mathematical theory involved, he utilised the theorems presented in chapter II.

Now, we present the details of the various steps:

Initial Step: Determine a basic solution and let $H = \{i | i = 1, \ldots, m+n\}$. H is a set containing the indices of all variables. The first m elements correspond to the original constraints and the next n elements, the non-negativity constraints.

Step (1): Check all the basic variables $x_i^B = S_k$ $k \in H$ for the property $a_{ij} \leq 0$, $j = 1, \ldots, n$. If this holds, then constraint k is redundant, (Theorem 2.1); remove k from H and drop row i.

Step (2): Determine the set $Q = \{i | x_i^B = S_k$ and $b_i = 0\}$. If $Q = \phi$, then all non-basic variables $x_j^N = S_k$ are slacks of non-redundant constraints (Theorem 2.4); remove these k from H and go to step (5). Otherwise continue with step (3).

Step (3): Check all the basic variables $x_i^B = S_k$, $i \in Q$ for the property $a_{ij} \geq 0$, $j = 1, \ldots, m$, $j \neq p$ and $a_{ip} < 0$. If this holds, then $S_q = X_p^N$ is a slack of a non-redundant constraint (Theorem 2.2); remove q from H.

Step (4): For every non-basic varaible $x_p^N = S_k$ $k \in H$. Check the property $a_{ip} \geq 0$ for all $i \in Q$. If this holds, then constraint k is non-redundant (Theorem 2.3); remove k from H.

Step (5): If $H = \emptyset$, stop. Otherwise find the row with the lowest index k, such that $x_k^B = S_r$ and $r \in H$. If no such row is found continues with step 7. In row k find the column p with $a_{kp} = \max_j a_{kj}$. Determine the minimum quotient $b_t/a_{kp} = \min_i \{b_i/a_{ip} | a_{ip} > 0\}$.

If this quotient is unique, then, $S_q = x_i^B$ is the slack of a non-redundant constraint (Theorem 2.5); remove q from H. Further, if $q = r$ (ie. the unique quotient is in the current objective row), then repeat step 5. Otherwise continue with step (6).

Step (6): Perform a simplex pivot on a_{tp} and drop row t if the non-basic variable in column p was a slack of a redundant constraint. Go to step (1).

Step (7): Introduce a non-basic variable $x_j^N = S_k$ with $k \in H$ into the basis and then go to step (1).

Now, we present the following numerical example:

The problem is as follows:

$$x_1 - x_2 \le 2 \quad (1)$$

$$2x_1 - x_2 \le 7 \quad (2)$$

$$x_1 \le 2 \quad (3)$$

$$-x_1 + 2x_2 \le 4 \quad (4)$$

$$2x_2 \le 5 \quad (5)$$

$$x_1 + x_2 \le 4 \quad (6)$$

$$x_1 \ge 0 \quad (7)$$

$$x_2 \ge 0 \quad (8)$$

Initial Step: A basic feasible solution is given by $(S_7, S_8) = (0,0)$ and the corresponding contracted tableau T_0 is:

$T_0 =$

	S_7	S_8	RHS
S_1	1	-1	2
S_2	2	1	7
S_3	1	0	2
S_4	-1	2	4
S_5	0	2	5
S_6	1	1	4

with index set H = (1, 2, 3, 4, 5, 6, 7, 8).

Step (2): $Q = \emptyset$, S_7 and S_8 are slacks of non-redundant constraints, $H = (1, 2, 3, 4, 5, 6)$;

Step (5): Select S_1 as the slack of the objective function. In column 1, there is a tie for the minimum quotient;

Step (6): Pivoting on a_{31} we get tableau T_1:

T_1 =

	S_3	S_8	RHS
S_1	-1	-1	0
S_2	-2	1	3
S_7	1	0	2
S_4	1	2	6
S_5	0	2	5
S_6	-1	1	2

Step (1): S_1 is a slack of redundant constraint, drop row 1, $H = (2,3,4,5,6)$;

Step (2): $Q = \emptyset$, S_3 is a slack of a non-redundant constraint, $H = (2,4,5,6)$;

Step (5): Select S_2 as the slack of the objective function. In column 2, the pivot element $a_{62} = 1$ is unique; S_6 is a slack of a non-redundant constraint, $H = (2,4,5)$.

Step (6): Pivot on a_{62} to obtain tableau T_2:

$T_2 =$

	S_3	S_6	RHS
S_2	-1	-1	1
S_7	1	0	2
S_4	3	-2	2
S_5	2	-2	1
S_8	-1	1	2

Step (1): S_2 is a slack of a redundant constraint, drop row 1, $H = (4,5)$;

Step (5): Select S_4 as the slack of the objective function. In column 1, the pivot element $a_{41} = 2$ is unique, S_5 is a slack of a non-redundant constraint, $H = (4)$;

Step (6): Pivoting on a_{41}, we get tableau T_3:

$T_3 =$

	S_5	S_6	RHS
S_7	-0.5	1	1.5
S_4	-1.5	1	0.5
S_3	0.5	-1	0.5
S_8	0.5	0	2.5

Step (5): S_4 is still the slack of the objective function. In column 2, the minimum quotient is unique and is in the row containing S_4. Hence, S_4 is a slack of a non-redundant constraint, $H = \phi$; stop.

Hybrid Method

Considering the major difficiences for the co-ordinate direction method, there is no guarantee that the remaining unidentified constraints are actually redundant, and the extended sign test method results in more extreme points to be determined in order to identify non-redundant constraints. Therefore, a Hybrid method (Lotfi (1981)) was developed which consists of two parts. In the first part, the co-ordinate direction method, is used to identify some of the non-redundant constraints. In the second part, the extended sign test method identifies the remaining constraints.

Each part requires a different initial solution. The co-ordinate direction method requires an interior point, whereas the extended sign test method needs a basic feasible solution. Therefore, one solution is obtained from another by using sensitivity analysis to overcome this difficulty.

Once a basic feasible solution for the system

$$\begin{aligned} AX &\leq b \\ X &\geq 0 \end{aligned} \qquad \ldots (3.1.2.1)$$

has been found, perturb the above system by two vectors (E_1, E_2) containing small positive values, Then an interior feasible solution is obtained by letting

$$X^0 = \hat{S} + E_2 \qquad \ldots (3.1.2.2)$$

where $\hat{S}$ denotes the values of the slacks of the non-negativity constraints in a basic feasible solution to (3.1.2.1).

Compute the change in the right-hand side $\Delta\hat{S}$ as follows:

$$\Delta b = E_1 + AE_2 \qquad \ldots (3.1.2.3)$$

and

$$\Delta\hat{S} = B^{-1}\,\Delta\bar{b} \qquad \ldots (3.1.2.4)$$

Then a basic feasible solution to (3.1.2.1) is simply:

$$X^B = \hat{S} + \Delta\hat{S} \qquad \ldots (3.1.2.5)$$

Now, we present the details of the steps for the Hybrid method as follows:

Initial Step: let $H = \{i \mid i = 1, \ldots, m+n\}$, where H is the set of indices for all variables. Store $AX \leq b$, and compute $\Delta\bar{b}$ and store it. Find X^0 and go to step 1.

Step 1: Retrieve $AX \leq b$, put it in proper form;

Step 2: Using X^0 as the starting interior feasible solution, perform the co-ordinate direction method for a pre-specified number of iterations. Remove the indices of identified constraints from H. If $H = \emptyset$, stop, all constraints are non-redundant. Otherwise continue with step (3).

Step 3: Retrieve the tableau and Δb, update the right-hand side and go to step (4).

Step 4: Apply the extended sign test method to classify the constraints starting with the above tableau. Continue until $H = \emptyset$. Then, stop and output the status of all constraints.

The first part of the above algorithm requires a stopping criterion as in the co-ordinate direction method. It is suggested that one co-ordinate direction iteration seems to be a reasonable upper limit to the number of such iterations.

Now, to illustrate the use of the Hybrid method, consider the same numerical example presented for Boneh and Golan's method.

As before, H = (1,2,3,4,5,6). Adding the slacks, the initial contracted tableau is:

T_0:

	S_5	S_6	RHS
S_1	-1	1	1
S_2	1	1	3
S_3	1	0	2
S_4	4	3	12

with $E_1 = (.01, .01, .01, .01)^T$ and $E_2 = (.01, .01)^T$,

$$\Delta b = (.01, .03, .02, .08)^T.$$

The perturbed problem is tableau T_1 which is feasible.

		s_5	s_6	RHS
	s_1	-1	1	0.99
	s_2	1	1	2.97
T_1:	s_3	1	0	1.98
	s_4	4	3	11.92

Store Δb and the above tableau for later use. $X^0 = (.01, .01)^T$ since slacks of non-negativity constraints are zero. Now begin with part one of the algorithm.

Step (1):

$$\begin{array}{rcll}
x_1 - x_2 & \geq & -1 & (1) \\
-x_1 - x_2 & \geq & -3 & (2) \\
-x_1 & \geq & -2 & (3) \\
-4x_1 - 3x_2 & \geq & -12 & (4) \\
x_1 & \geq & 0 & (5) \\
x_2 & \geq & 0 & (6)
\end{array}$$

Step (2): Using one iteration of the co-ordinate direction method, constraints one, three, five and six are identified as non-redundant. $H = (2,4)$.

Step (3): Retrieve T_1 and Δb and update T_1 by adding $B^{-1}\,\Delta b = \Delta b$ to the right-hand sides (in this instance B^{-1} is the identity matrix. The updates tableau is T_0.

The contracted tableau is T_2:

T_2:

	S_5	S_6	RHS
S_1	-1	1	1
S_2	1	1	3
S_3	1	0	2
S_4	4	3	12

Taking S_2 as the slack of the objective function and pivoting on $a_{31} = 1$, obtaining T_3.

T_3:

	S_3	S_6	RHS
S_1	1	1	3
S_2	-1	1	1
S_3	1	0	2
S_4	-4	3	4

Select the second column for pivoting. In this column, there is a unique pivot in the row containing S_2. Thus, S_2 is a slack of a non-redundant constraint, H = (4). So select S_4 as the slack of the objective function and pivot on $a_{22} = 1$ to get T_4 which implies S_4 is a slack of a redundant constraint. Then $H = \emptyset$, so the algorithm stops.

T_4:

	s_3	s_2	RHS
s_1	2	-1	2
s_6	-1	1	1
s_5	1	0	2
s_4	-1	-3	1

3.2 Group Two Methods

As mentioned earlier, the objective of the methods in group two is to identify extraneous variables and non-binding constraints. Before presenting the details of these methods we restate our (primal) linear programming problem as:

$$\begin{aligned} &\max \quad CX \\ &\text{S.t.} \quad AX \leq b \\ &\qquad\quad X \geq 0 \end{aligned} \qquad \ldots (3.2.1)$$

Then, the dual problem associated with system (3.2.1) is

$$\begin{aligned} &\min \quad Wb \\ &\text{S.t.} \quad WA' \geqslant C \\ &\qquad\quad W \geqslant 0 \end{aligned} \qquad \ldots (3.2.2)$$

where A' is an (nxm) matrix transposed from the original matrix A. C and X are n vectors, b and W are m vectors.

3.2.1 Klein and Holm's Method

Klein and Holm's method utilises the complementary slackness theorem (CST) of linear programming (see for example, Jarvis and Bazaraa (1977)) in combination with bounds on the primal and duals variables. Such bounds are directly available in problems with bounded variables and some problems with special structure, ie. problems with positive coefficients and problems with Leontief structure (for details see Klein and Holm (1975)).

In order to present the mathematical theory used in this method , we define the following notation. Let pos(.) and neg(.) denote two operators which select the positive and negative elements of a matrix or vector. For example, if v is a vector then pos(v) is a vector which contains the positive elements of v and zeros for non-positive elements of v, ie. v = pos(v) + neg(v). Let A(i.) and A(.j) denote the ith row and jth column of the matrix A, respectively. Finally, let x^{ℓ}, x^{u} and w^{ℓ}, w^{u} be lower and upper bounds on the optimal solutions X* and W* of (3.2.1) and (3.2.2) respectively.

The following two theorems and associated corollaries establish sufficient conditions for identifying extraneous variables and non-binding constraints. the reader may refer to the reference for the proofs.

Theorem (3.1)

If there exists column index sets R and T, and vectors P>0 and q>0 such that

$$C_R P - C_T q > w^u \mathrm{pos}\ (A_R P - A_T q) + w^{\ell}\ \mathrm{neg}\ (A_R P - A_T q) \qquad \ldots (3.2.1.1)$$

then there exists a column index $t \in T$ such that X_t is extraneous (ie. it has a value of zero for every optimal solution of (3.2.1).

Corollary (3.1)

If there exist column indices r and t such that

$$C_r - C_t > w^u \mathrm{pos}\ (A(.r) - A(.t)) + w^{\ell} \mathrm{neg}\ (A.r) - A(.t) \qquad \ldots (3.2.1.2)$$

then x_t is extraneous.

Note that when $w^{\ell} = 0$ then (3.2.1.2) reduces to:

$$c_r - c_t > w^u \, pos(A(.r) - A(.t)) \qquad \ldots (3.2.1.3)$$

<u>Theorem (3.2)</u>

If there exist row index sets K and L and vectors P>0 and q>0 such that

$$Pb_k - qb_L > pos\,(PA_k - qA_L)x^u + neg(PA_k - qA_L)x^{\ell} \qquad \ldots (3.2.1.4)$$

then there exists a row index $k \in K$ such that constraint k is non-binding.

<u>Corollary (3.2)</u>

If there exist row indices r and t such that

$$b_r - b_t > pos(A(r.) - A(t.))x^u + neg(A(r.) - A(t.))x^{\ell} \qquad \ldots (3.2.1.5)$$

Note that in the system (3.2.1) $x^{\ell} = 0$ resulting in (3.2.1.5) reduces to

$$b_r - b_t > pos(A(r.) - A(t.))x^u \qquad \ldots (3.2.1.6)$$

Klein and Holm's algorithm searches by making pairwise comparisons through rows and columns of system (3.2.1) to find row and column indices satisfying conditions (3.2.1.3) and (3.2.1.6). Clearly, these conditions are sufficient, but not necessary for identifying extraneous variables and non-binding constraints. The effectiveness of the approach depends greatly on the tightness of the required bounds on variables in systems (3.2.1) and (3.2.2).

Theorem (3.3)

If A>0, b>0 and c>0 then

(a) $$x_j^u = \min_{i:a_{ij}>0} \{b_i/a_{ij}\} \qquad j = 1, \ldots, n \qquad \ldots (3.2.1.7)$$

is an upper bound on the optimal value of the structural variables in system (3.2.1).

(b) i) $$w_i^u = \max_{j:a_{ij}>0} \{c_j/a_{ij}\} \qquad i=1, \ldots, m \qquad \ldots (3.2.1.8)$$

ii) $$w_i^u = (1/b_i) \sum_{j \in k} c_j x_j^u \qquad i=1, \ldots, m \qquad \ldots (3.2.1.9)$$

where K is the set of column indices corresponding to the K largest values of $c_j x_j^u$ and K = min (m,n)

iii) $$w_i^u = (1/b_i) \min_{k \in M} \{b_k \max_{j:a_{kj}} (c_j/a_{ij}) \qquad i=1, \ldots, m \qquad \ldots (3.2.1.10)$$

where M is the set of row indices which correspond to strictly positive rows, ie.

$$M = \{i | a_{ij}>0, \quad j = 1, \ldots, n\}$$

The following steps represent the details of Klein and Holm's algorithm:

Initial Step: Determine the upper bounds for both primal and dual variables.

Step (1): Let j = 1, and set the logical variable IRD = 0.

Step (2): Let t be the index of the smallest element of C;

Step (3): Let r be the index of j-th largest element of C;

Step (4): If condition (3.2.1.3) is satisfied go to step (6);

Step (5): If there are more columns to be compared with C_t set $j = j + 1$ and go to step (3) otherwise continue with step (8).

Step (6): Delete column t from the problem, set IRD = 1 and go to step (8).

Step (7): Remove C_t temporarily, if no more columns are left, go to step (8), otherwise continue with step (1).

Step (8): Let $i = 1$ and set the logical variable IRD = 0.

Step (9): Let t be the index of smallest element of b;

Step (10): Let r be the index of the i-th largest element of b;

Step (11): If condition (3.2.1.5) holds go to step (13).

Step (12): If there are more rows to be compared with b_t set $i=i+1$ and go to step (10); otherwise continue with step (14).

Step (13): Delete row r from the problem, set IRD=1 and go to step (1).

Step (14): Remove b_t temporarily, if no more rows left go to step (15). Otherwise continue with step (8).

Step (15): If IRD=0 stop, no more reduction is possible. Otherwise continue with Step (10).

Now, we present a numerical example (taken from Klein and Holm (1975)) to illustrate the above algorithm. Consider the following system:

$$\max. \quad 23x_1 + 23x_2 + 22x_3 + 18x_4 + x_5$$

S.t.

$$\begin{array}{lr} 22x_1 + 18x_2 + x_3 + 23x_1 \le 6 & (1) \\ 17x_2 + 22x_3 + 11x_5 \le 6 & (2). \\ 15x_1 + 21x_5 \le 13 & (3) \\ 23x_1 + 14x_2 + 14x_5 \le 14 & (4) \\ 3x_4 \le 18 & (5) \end{array}$$

$$x_j \ge 0 \quad j=1, \ldots, 5$$

Initial Step: Clearly, the lower bounds on both primal and dual variables are zero.

$$X^u = (0.27, 0.33, 0.27, 0.26, 0.55)$$

$$W^u = (4.16, 1.35, 1.53, 1.64, 1.40)$$

Steps 2 - 4: $t = 5$, $r = 3$: condition (3.2.1.6) holds, row 5 is elimimated.

Steps 9 - 11: $t = 1$, $r = 5$: condition (3.2.1.6) holds, row 5 is eliminated.

Steps 9 - 11: $t = 1$, $r = 4$: condition (3.2.1.6) holds, row 4 is eliminated

Steps 9 - 11: $t = 1$, $r = 3$: condition (3.2.1.6) holds, row 3 is eliminated.

Steps 2 - 4: $t = 4$, $r = 1$: condition (3.2.1.2) holds

column 4 is eliminated.

No further reduction is possible, the problem reduces to:

$$\max \quad 23x_1 + 23x_2 + 22x_3$$

S.t.

$$22x_1 + 18x_2 + x_3 \leq 6 \qquad (1)$$

$$17x_3 + 22x_3 \leq 6 \qquad (2)$$

$$x_j \geq 0 \quad j = 1,2,3$$

As Klein and Holm point out, further reductions may be achieved if the bounds are updated after each reduction. For instance, in the above example the lower bound and the previous upper bounds, column 2 can be eliminated (condition (3.2.1.2)). Computational results are reported in Klein and Holm (1975) and (1976)) for LPPs with positive coefficients.

3.2.2 Williams' Method

The second technique in this group is proposed by Williams (1983). Williams' method is similar to an earlier algorithm developed by Zionts (1965) called "The Extended Geometric Method". The extended geometric method is based on a collection of theorems which makes it possible to compute bounds on primal and dual variables from the structure of the problem. Then, according to these bounds extraneous variables and non-binding constraints are identified and dropped. The tightening of the bounds on all remaining variables continues until no further reduction is possible in which case a simplex pivot is performed. The above process continues until optimality is achieved.

Williams' modification to the above alogrithm consists of eliminating the simplex pivot step and adding other steps which remove singleton columns and rows (defined as columns or rows with exactly one non-zero entry excluding the cost coefficients and right-hand sides). In order to present the mathematical theory used in Williams' method we will utilise the teminology implemented in the previous section. To reiterate, consider the system (3.2.1) and denote the lower and upper bounds on x_j, by x_j^ℓ and x_j^u, j=1, ..., n. respectively. Similarly, denote the lower and upper bounds on the dual variables w_i (system (3.2.2)) by w_i^ℓ and w_i^u, i=1, ...,m. We will frequently refer to the w_i's as shadow prices, and refer to their associated bounds as shadow price bounds. Also, for each of the primal constraints we introduce the concept of "activity level". For each row, the lower activity (L_i) and upper activity (U_i) are given as

$$L_i = \sum_{j:a_{ij}>0} a_{ij}\, x_j^{\ell} + \sum_{j:a_{ij}<0} a_{ij}\, x_j^{u} \qquad i=1, \ldots, m. \qquad \ldots (3.2.2.1)$$

$$U_i = \sum_{j:a_{ij}>0} a_{ij}\, x_j^{u} + \sum_{j:a_{ij}<0} a_{ij}\, x_j^{\ell} \qquad i=1, \ldots, m \qquad \ldots (3.2.2.2)$$

Similarly, for each column j we define the "imputed cost" and denote its lower and upper values by P_j and Q_j respectively. The lower and upper imputed costs are given as:

$$P_j = \sum_{i:a_{ij}>0} a_{ij} w_j^{\ell} + \sum_{i:\ a_{ij}<0} a_{ij} w_j^{u} \qquad j=1, \ldots, n \qquad \ldots (3.2.2.3)$$

$$Q_j = \sum_{i:a_{ij}>0} a_{ij} w_j^{u} + \sum_{i:a_{ij}<0} a_{ij} w_j^{\ell} \qquad j=1, \ldots, n \qquad \ldots (3.2.2.4)$$

Now, we present the mathematical theory implemented in Williams' method. Initially, for all of the variables (primal as well as the dual) the lower bounds are set to zero (because of the non-negativity constraints) and the upper bounds at a sufficiently large real number M. Since all of the tests in this method have their dual counterparts we will describe the tests in pairs with the primal test followed by the dual test:

Primal Test One (P1): A singleton row may be replaced by a simple bound. According to the nature of a_{ij} a new simple bound of $\bar{x}_j^{\ell}$ or $\bar{x}_j^{u}$ is given to x_j as follows:

$$x_j^{\ell} = \bar{x}_j^{\ell} \quad \text{if } a_{ij} < 0 \text{ and } \bar{x}_j^{\ell} = \frac{b_i}{a_{ij}} > x_j^{\ell} \qquad \ldots (3.2.2.5)$$

$$x_j^u = \bar{x}_j^u \quad \text{if} \quad a_{ij} > 0 \quad \text{and} \quad \bar{x}_j^u = \frac{b_i}{a_{ij}} < x_j^u \qquad \ldots (3.2.2.6)$$

Also, the singleton row must have the original shadow price bounds (0,M). The reason is that, tighter shadow price bounds indicate that singleton columns may have been removed temporarily. It should be noted that if the new bound obtained by test P1 is less strict than the existing value, the row will be found redundant according to test P2 below:

Dual Test One (D1): A singleton column may be replaced by a shadow price according to the nature of a_{ij} a new shadow price bound $\bar{w}_i^\ell$ or $\bar{w}_i^u$ is given to w_i as follows:

$$w_i^\ell = \bar{w}_i^\ell \quad \text{if } a_{ij} > 0 \quad \text{and } \bar{w}_i^\ell = \frac{c_j}{a_{ij}} > w_i^\ell \qquad \ldots (3.2.2.7)$$

$$w_i^u \equiv \bar{w}_i^u \quad \text{if } a_{ij} < 0 \text{ and } \bar{w}_i^u = \frac{c_j}{a_{ij}} < w_i^u \qquad \ldots (3.2.2.8)$$

Similarly, the singleton column must have the original primal bounds (0,M). The reason is that tighter primal bounds indicate that singleton rows may be removed remporarily. As with the primal bounds, when the new shadow price bound is less strict than the existing value, x_j will be set to one of the bounds according to test D2 (below) and the above test not applied.

Primal Test Two (P2): A constraint taken in conjunction with primal bounds may demonstrate a "redundant" or infeasible constraint. According to the values of lower activity (L_i) and upper activity (U_i), the following actions are taken:

$L_i > b_i$ and $w_i^u = M$ constraint i(and hence model) infeasible

$L_i > b_i$ and $w_i^u < M$ subtract w_i^u times constraint i from objective and remove constraint i

$U_i < b_i$ and $w_i^\ell = 0$ constraint i is redundant, remove constraint i.

$U_i < b_i$ and $w_i^\ell > 0$ subtract w_i^ℓ times constraint i from objective and remove constraint i.

Dual Test Two (D2): A column taken in conjunction with shadow price bounds may demonstrate that the corresponding variable can be set at one of its bounds. By comparing P_j and Q_j with C_j, the following actions are taken:

$P_j > C_j$ and $x_j^\ell = 0$ x_j is extraneous, remove column j.

$P_j > C_j$ and $x_i^\ell > 0$ set x_j to its lower bound and substitute out.

$Q_j < C_j$ and $x_j^u = M$ variable x_j (and hence model) unbounded.

$Q_j < C_j$ and $x_j^u < M$ set x_j to its upper bound and substitute out.

Primal Test Three (P3): A constraint together with primal bounds on some of the variables may imply bounds on other variables. The new bounds are readily computed by using the lower and upper activities.

$$\bar{x}_j^\ell = x_j^u + (b_i - L_i)/a_{ij} \qquad \text{if } a_{ij}<0 \qquad \ldots (3.2.2.9)$$

$$\bar{x}_j^u = x_j^\ell + (b_i - L_i)/a_{ij} \qquad \text{if } a_{ij}>0 \qquad \ldots (3.2.2.10)$$

It should be noted that the new bounds $\bar{x}_j^{\ell}$ and $\bar{x}_j^{u}$ may be less strict than the existing value in which case they are ignored. Moreover, the new bounds may result in the following actions to be taken: If $\bar{x}_j^{\ell} = x_j^u$ or $\bar{x}_j^u$, or $\bar{x}_j^u = x_j^{\ell}$ or $\bar{x}_j^{\ell}$ set variable x_j at the common value and substitute for it.

Dual Test Three(D3): A column together with bounds on some of the shadow price bounds. The new bounds are readily computed by using the lower and upper imputed costs,

$$\bar{w}_i^u = w_i^{\ell} + (C_j - Q_j)/a_{ij} \qquad \text{if } a_{ij}<0 \qquad \ldots (3.2.2.11)$$

$$\bar{w}_i^{\ell} = w_i^u + (C_j - Q_j)/a_{ij} \qquad \text{if } a_{ij}>0 \qquad \ldots (3.2.2.12)$$

Similarly, these new bounds may be less strict than the existing values in which case they are ignored. Also, the new bounds may result in the following action to be taken: If $\bar{w}_i^{\ell} = w_i^u$ or $\bar{w}_i^u$, or $\bar{w}_i^u = w_i^{\ell}$ or $\bar{w}_i^{\ell}$ set w_i to this common value and use as a multiple of the constraint to subtract from the objective function.

The above six tests may be implemented for reducing the size of the problem by making successive passes over the model. On each pass the columns of the model are examined sequentially. For each column: Tests P3 (except for first pass), D2, D1, D3 are applied in this order. At the end of each pass, Tests P2, P1 are applied in this order. However, performing these tests without any systematic approach may prove disadvantageous. The reason is, in a loose sense tightening the bounds on primal variables and dual variables simultaneously have opposite effects on the model. In order to resolve the dilemma over whether to relax or tighten the bounds a two phase procedure is suggested. In the first phase, primal bounds are tightened and shadow price bounds are relaxed. In the second phase, primal bounds are relaxed and shadow price bounds are tightened. A phase of the procedure terminates when two successive passes yield no simplification. Furthermore, when singleton columns replaced by shadow price bounds or constraints with non-zero shadow prices removed by subtracting from the objective, it is ultimately necessary to restore them. This is to ensure that the variables are at their optimal values, and the model will not reduce any further. The whole procedure is repeated in part two, however, singleton columns are not replaced by shadow price bounds and constraints with non-zero shadow prices are not subtracted from the objective function.

In order to illustrate the use of Williams' method, we present a numerical example (taken from Williams (1983)). Consider

$$
\begin{array}{llllllllll}
\max & 2x_1 & + 3x_2 & - x_3 & - x_4 & & & & & \\
\text{S.t.} & & & & & & & & & \\
R_1: & x_1 & + x_2 & + x_3 & - x_4 & \leq & 4 & & w_i^{\ell} & w_i^{u} \\
R_2: & -x_1 & - x_2 & + x_3 & - x_4 & \leq & 1 & & 0 & M \\
 & & & & & & & & 0 & M \\
R_3: & x_1 & & & + x_4 & \leq & 3 & & & \\
 & & & & & & & & 0 & M \\
x_j^{\ell} & 0 & 0 & 0 & 0 & & & & 0 & M \\
x_j^{u} & M & M & M & M & & & & &
\end{array}
$$

Part 1:

Phase 1: Pass 1: $P_3 > C_3$, x_3 is extraneous; remove x_3

$U_2 < b_2$, R_2 is redundant; remove R_2

Pass 2: x_1^u is tightened to 3.

singleton column x_2 replaced by $w_1^\ell = 3$

x_4^u tightened to 3.

Pass 3: $P_1 > C_1$, x_1 is extraneous; remove x_1.

$U_1 < b_i$, multiply R_1 by $w_1^\ell = 3$ and subtract from the objective, remove R_1.

The model is now

		w_i^ℓ	w_u^u
max	$5x_4 + 12$		
R_3	$x_4 \leq 3$	0	M
x_j^ℓ	0		
x_j^u	3		

$U_3 = b_3$, remove R_3.

Pass 4: $Q_4 < C_4$, $x_4 = x_4^u = 3$, and substituted.

The model is now: max 27

S.t. nothing

Clearly the remaining two passes and Phase two will not have any changes. Then, the algorithm enters the second part. The singleton column x_2 and constraint R_1 (which was removed with non-zero shadow price) are restored. Now the model is:

		w_i^ℓ	w_i^u
max	$3x_2 - 3$		
R_1	$x_2 \leq 10$	0	M
x_j^ℓ	0		
x_j^u	M		

Part 2:

Phase 2; Pass 1: Singleton row R_1, replaced by $x_2^u = 10$.

Pass 2: $Q_2 < C_2$, $x_2 = x_2^u = 10$, and substituted.

Other passes and phases are completed with no action.

The solution: $x_1 = 10$, $x_2 = 10$, $x_3 = 0$, $x_4 = 3$, objective $= 27$.

3.2.3 Reduce Method

The third method in this group is proposed by Lotfi (1981), which identified non-binding and/or non-redundant constraints by applying tests one and two to the primal problem. Then, the dual counterparts of these theorm are used to identify extraneous variables. The use of the tests one and two was illustrated in previous method. To present the application of these tests to the dual problem, given a basic feasible solution, the non-basic variable x_j^N is extraneous if

$$a_{ij} \geq 0 \qquad (i=1, \ldots, m) \text{ and } z_j - c_j \geq 0 \qquad \ldots (3.2.3.1)$$

where $z_j - c_j$ is the reduced cost. The correctness of the above test may be illustrated by noting that the j-th dual constraint is redundant.

The dual counterpart of test two, however, is somewhat different. Recall that test two would identify a redundant constraint one pivot away from test one. In fact, the simplex method works towards attaining dual feasibility. Therefore, a violated dual constraint may satisfy the condition as well. That is, in a basic feasible solution with

$$a_{ij} \leq 0, \; i \neq r, \; a_{rj} > 0 \text{ and } z_j - c_j \leq 0 \qquad \ldots (3.2.3.2)$$

the basic variable x_r^B is extraneous. The proof of the above is the same as that of test two, pivoting on a_{rj} will give the condition proposed in (3.2.3.1).

In addition to the above two tests for identifying the extraneous variables one may identify such variables in a special-type implicit equality, having non-negative entries and a zero right-hand side. Then, a variable with a positive entry in this row is extraneous. That is, if

$$a_{ij} \geq 0 \quad j=1, \ldots, n \qquad \text{with } b_i = 0 \qquad \ldots (3.2.3.3)$$

then $x_j^N = 0$ for $a_{ij} > 0$

The proof of this test is rather simple:

There are many algorithms to implement the above tests, and in each algorithm, more than one course of action may be implemented at certain tests. For example, suppose that condition (3.2.3.2) is satisfied at certain tableau, then the course of action which is adopted in this method, is to mark the variables appropriatly when they were identified and drop the row (column) when the variable entered (left) the basis.

Now, we present the details of the Reduce method in algorithmic form:

Initial Step: Determine a basic feasible solution.
Let $H = \{k | S_k = x_i^B\}$ and $G = \{r | S_r = x_j^N\}$ where H and G are the sets of indices of slack variables in rows and columns still remaining in the problem.

Step (1): If the current solution is optimal go to step (8). Otherwise continue with step (2).

Step (2): For every row i with $x_i^B = S_k$ and $k \notin H$, check the property

$$a_{ij} \geq 0 \text{ for all } j \text{ and } b_i = 0$$

If this holds remove all r with $S_r = x_j^N$ and $a_{ij} > 0$ from G drop all such columns.

Drop row i and remove k from H.

Step (3): For every row i with $x_i^B = S_k$ and $k \in H$, check the property:

$$a_{ij} \leq 0 \text{ for all } j.$$

If this holds, then drop all these rows and remove the indices of their slacks from H.

Step (4): For every row i with $x_i^B = S_k$ and $k \notin H$, check the property

$$a_{ij} \geq 0 \quad j \neq p \quad \text{and } a_{ip}<0 \text{ with } b_i = 0$$

If this holds, then mark x_p^N as the slack of a non-binding constraint.

Step (5): For every column j with $x_j^N = S_r$ and $r \notin G$, check the property

$$a_{ij} \geq 0 \quad \text{for all } i \quad \text{and } z_j - c_j \geq 0$$

If this holds, then drop column j and remove r from G.

Step (6): For every column j with $x_j^N = S_k$ and $k \notin G$, check the property

$$a_{ij} \leq 0 \quad i \neq r \quad a_{rj}>0 \quad \text{and } z_j - c_j \leq 0.$$

If this holds, then mark $S_q = x_r^B$ as extraneous.

Step (7): Determine the non-basic variable $x_j^N = S_k$, $k \notin G$ with the most negative reduced cost $z_j - c_j$. If no such variable exists go to step (8). Otherwise compute:

$$a_{rj} = \min_i \{b_i/a_{ij} | a_{ij}>0\}$$

and perform a simplex pivot on a_{rj} updating the rows and columns still remaining in the problem. Then, drop the row and/or the column if the respective variables have been marked and remove their indices from G and H. Update G and H for the indices.

Step (8): If no rows or columns have been removed, stop. Otherwise update the right-hand sides for the rows and $z_j - c_j$ for the columns which were dropped, then stop.

It should be noted that steps 2 - 6 may be repeated until no further changes are made. In chapter VI we will present the results of this method on the tested problems.

Now, we will illustrate the use of the above algorithm by the following numerical example. Consider

$$\max \quad x_1 - 2x_2 + x_3 + 5x_4 - 4x_5$$

$$\text{S.t.}$$

$$\begin{array}{lr} x_1 + x_2 + x_3 \leq 10 & (1) \\ x_2 - x_3 + x_4 + x_5 \leq 12 & (2) \\ x_1 + x_4 \leq 3 & (3) \end{array}$$

adding slack variables, the tableau for the initial basic feasible solution is:

	s_4	s_5	s_6	s_7	s_8	
Z	-1	2	-1	-5	4	0
s_1	1	1	1	0	0	10
s_2	0	1	-1	1	1	12
s_3	1	0	0	1	0	3

with $H = \{1,2,3\}$, $G = \{4,5,6,7,8\}$.

Step (5): S_5 and S_8 are extraneous variables, drop columns 2 and 5: $G = \{4,6,7\}$;

Step (6): Mark S_1 as extraneous (denoted by (*));

Step (7): The pivot element is $a_{34} = 1$. The updated reduced tableau is:

	S_4	S_6	S_3	RHS
Z	4	-1	5	15
$*S_1$	1	1	0	10
S_2	-1	-1	-1	9
S_7	1	0	1	3

with $H = \{1,2,7\}$ and $G = \{4,6,3\}$

Step (3): Row 2 is non-binding: drop row 2, $H = \{1,7\}$;

Step (5): S_4 and S_3 are extraneous, drop column 1 and 4 and $H = G = \{6\}$;

Step (7): Pivoting on $a_{13} = 1$, getting the optimal solution as

$$S_6 = 10, \quad S_7 = 3, \quad S_2 = 19, \quad \text{with } Z = 25.$$

3.3 Group Three Methods

The method in this group is presented by Thompson and Sethi (1983) which is unlike other methods. They attempt to solve LPP's by defining certain constraints called "non-candidate constraints" as those which never contain a potential pivot element during the course of solving a linear program. Keeping these constraints in updated form is of no value. A "Candidate Constraint" is one that, for at least one pivot step, contains a potential point.

The method is merely a modification of the standard simplex method in which only constraints which currently are candidates are updated, taking advantage of the fact that only some of the candidate constraints will be binding at the optimum solution. Therefore, no new theoretical results are needed to establish the correctness of the approach. Hence, in order to present the method, we restate the linear programming problem as:

$$\begin{aligned} \text{Max} \quad & CX \\ \text{S.t.} \quad & AX \leq b \\ & X \geq 0 \end{aligned} \qquad \ldots (3.3.1)$$

without loss of generality, assume that $b \geq 0$. Adding slack variables to $AX \leq b$ and using matrix notation below:

$$\begin{bmatrix} 1 & -C & 0 \\ 0 & A & I \end{bmatrix} \begin{bmatrix} Z \\ X \\ S \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix} \qquad \ldots (3.3.2)$$

Any instance of the above problem may be obtained by choosing a proper basis B and multiplying the right-hand side vector b by B^{-1}.

That is, $x^B = B^{-1}b$, hence $Z = C_B B^{-1}b$ which may be written as:

$$\begin{bmatrix} Z \\ x^B \end{bmatrix} = \begin{bmatrix} 1 & C_B B^{-1} \\ 0 & B^{-1} \end{bmatrix} \begin{bmatrix} 0 \\ b \end{bmatrix} \qquad \text{... (3.3.3)}$$

Therefore multiplying the left side of equation (3.3.2) by this same matrix or

$$\begin{bmatrix} 1 & C_B B^{-1} \\ 0 & B^{-1} \end{bmatrix} \begin{bmatrix} 1 & -C & 0 \\ 0 & A & 1 \end{bmatrix} = \begin{bmatrix} 1 & C_B B^{-1}A-C & C_B B^{-1} \\ 0 & B^{-1}A & B^{-1} \end{bmatrix} \qquad \text{... (3.3.4)}$$

which gives the desired matrix form of the system (3.3.2) after any iterations as

$$\begin{bmatrix} 1 & C_B B^{-1}A-C & C_B B^{-1} \\ 0 & B^{-1}A & B^{-1} \end{bmatrix} \begin{bmatrix} Z \\ X \\ S \end{bmatrix} = \begin{bmatrix} C_B B^{-1}b \\ B^{-1}b \end{bmatrix} \qquad \text{... (3.3.5)}$$

Note that the system (3.3.5) is a full tableau of the simplex method which is required by Thompson and Sethi's method. For the purpose of simplicity, redefine the above system. Let $x_{n+i} = S_i \quad i=1, \ldots, m.$

Let $Z-C = \begin{bmatrix} C_B B^{-1}A-C & C_B B^{-1} \end{bmatrix}$ and $y = \begin{bmatrix} B^{-1}A & B^{-1} \end{bmatrix}$ where $Z - C$ and y are of proper dimension. Also, Let $y_0 = B^{-1}b$, then we may rewrite the system (3.3.1) as

$$\max \quad Z + \sum_{j=1}^{m+n} (Z_j - C_j)x_j = C_B B^{-1} b$$

$$\text{S.t.} \quad \sum_{j=1}^{m+n} y_{ij} x_j = y_{10} \qquad i=1, \ldots, m \qquad \ldots (3.3.6)$$

Associated with system (3.3.6), the superscript (k) will denote the k-th iteration of the problem (eg. $x^{(k)}$ denotes the solution at k-th iteration).

Because this method utilises the maximum objective rule, the pivot element in every column with a non-negative reduced cost must be identified. The set of variables with a negative reduced cost is represented by:

$$J^{(k)} = \{j | Z_j - C_j < 0, \ j=1, \ldots, m+n\} \qquad \ldots (3.3.7)$$

Clearly, if $j^{(k)} = \emptyset$ the optimal solution has been found. The set of leaving basic variables is found by the usual minimum quotient rule, ie.

$$R^{(k)} = \{i | y_{i0}^{(k)}/y_{ij}^{(k)} = \min_{y_{rj}^{(k)}>0} y_{ro}^{(k)}/y_{rj}^{(k)}, \quad i=1, \ldots, m\} \qquad \ldots (3.3.8)$$

Now, we may define the set of "Candidate constraints" at iteration k as

$$S^{(k)} = \{i | i \in R_j^{(k)} \text{ for some } j \in J^{(k)}\} \qquad \ldots (3.3.9)$$

Then, the set of non-candidate constraints at iteration k is

$$\bar{S}^{(k)} = \{i | i \notin S^{(k)}, \ i=1, \ldots, m\} \qquad \ldots (3.3.10)$$

To determine the pivot element when $J^{(k)} \neq 0$, the following computation must be performed

$$\delta^{(k)} = \max_{j \in J^{(k)}} \quad y_{i0}^{(k)}/y_{ij}^{(k)} \qquad (\bar{c}_j - z_j) \qquad \ldots (3.3.11)$$

which increases the objective function by $\delta^{(k)}$.

As mentioned before, a permanent non-candidate constraint need not be updated at all during the course of the solution. At each iteration the set of non-candidate constraints $\bar{S}^{(k)}$ is not updated with the hope that they will never become violated. Obviously because the choice of pivot row i is by the minimum ratio rule (3.3.8) and (3.3.11), no non-candidate constraint at step k is ever violated at step k+1. However, such constraints may be violated in subsequent iterations. All that needs to be done to prevent such infeasibilities from occuring is to update the right-hand side vector (call this partial pivoting) for a given pivot element y_{ij}. In other words, $y_0^{(k+1)}$ may be computed from $y_0^{(k)}$ and a constraint i is violated if $y_{io}^{(k)} < 0$. In this case, the pivot step is not performed, instead the i-th constraint would be violated, a new pivot element is identified and the above process is repeated. When no constraint is violated for a given choice of a pivot element, a simplex pivot is performed, but the non-candidate constraints are not updated. This procedure is repeated until $J^{(k)} = 0$, which implies that optimality is achieved.

It should be noted that in constructing the set $R_j^{(k)}$, one may face unboundedness (ie. $y_{rj} \leq 0$, $r \varepsilon S^{(k-1)}$). However, this unboundedness may be false since a non-candidate constraint, say ℓ could contain a positive entry in column j if updated. Therefore, when the above condition occurs, the non-candidate constraints are updated one at a time until either a pivot element if found or there is no such constraint left, and the problem is indeed unbounded.

Now, we present the details of the method in algorithmic form:

Step (1): Find $J^{(0)}$, if $J^{(0)} = \emptyset$, stop, the solution is optimal. Otherwise find $S^{(0)}$, let $k = 0$ and go to step (2).

Step (2): Find (i, j) the row and column of the pivot element obtained from (3.3.11).

Step (3): Pivot on y_{ij} in the tableau restricted to the rows in $S^{(k)}$.

Step (4): If the solution is optimal update the right-hand side y_{io}, $i \in S^{(k)}$ and stop. Otherwise continue with step (5).

Step (5): Identify the non-candidate constraints in the updated tableau, remove them from $S^{(k)}$ to get $S^{(k+1)}$.

Step (6): Find (i,j), the row and column of the pivot element by (3.3.11) in the tableau restricted to $S^{(k+1)}$.

Step (7): Do a partial pivot on $y_0^{(k)}$ restricted to $S^{(k+1)}$ to get $y_0^{(k+1)}$ and $x^{(k+1)}$.

Step (8): Use $x^{(k+1)}$ to see if any constraint $i \notin S^{(k)}$ is violated. If not, replace k by k+1 and go to step (3). Otherwise, continue with step (9).

Step (9): Update the violated constraint and put in the current tableau. Add its index to $S^{(k)}$ and go to step (5).

To illustrate the use of the above algorithm, we will present a numerical example. Consider the problem:

$$
\begin{array}{lrcl}
\max & 2x_1 + 2x_2 + 3x_3 & & \\
\text{S.t.} & & & \\
& 2x_1 + x_2 + x_3 & \leq & 9 \\
& x_2 + 2x_3 & \leq & 6 \\
& -x_1 + 2x_2 - x_3 & \leq & 5 \\
& -x_1 + 3x_2 + x_3 & \leq & 12 \\
& x_1 , x_2 , x_3 & \geq & 0
\end{array}
$$

after adding the slack variables, the following initial tableau:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
Z	-2	-5	-3	0	0	0	0	0
x_4	2	1	1	1	0	0	0	9
x_5	0	1	2	0	1	0	0	6
x_6	-1	2	-1	0	0	1	0	5
$*x_7$	-1	3	1	0	0	0	1	12

The potential candidates for entering into the basis are $J^{(0)} = \{1,2,3\}$ with the candidate constraints. $S^{(0)} = \{1,3,2\}$.

From equation (3.57) the pivot element is $y_{32} = 2$. We pivot on y_{32} updating constraints in $S^{(0)}$ to get (non-candidate constraints are denoted by a (*)):

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
Z	-4.5	0	-5.5	0	0	2.5	0	12.5
x_4	2.5	0	1.5	1	0	-0.5	0	6.5
x_5	0.5	0	2.5	0	1	-0.5	0	3.5
$*x_2$	-0.5	1	-0.5	0	0	0.5	0	2.5
$*x_1$	-1	3	1	0	0	0	1	12.0

and the incoming variables for this tableau are $J^{(1)} = \{1,3\}$, with candidate constraints $S^{(1)} = \{1,2\}$.

The pivot element with the maximum objective function change is $y_{23} = 2.5$. So, we perform a partial pivot in the right-hand side to check for any violations.

$$X_3^{(1)} = 1.4, \qquad x_4^{(1)} = 4.4$$

Since no constraint will be violated we perform a pivot only on $S^{(1)}$ and the **z**-row.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
Z	-3.4	0	0	0	2.2	1.4	0	20.2
x_4	2.2	0	0	1	-0.6	-0.2	0	4.4
$*x_3$	0.2	0	1	0	0.4	0.2	0	1.4
$*x_7$	-1	3	-1	0	0	0	1	12.0

Now there is only one incoming variable x_1 and one candidate constraint. The pivot element is $y_{11} = 2.2$, so we do a partial pivot, $x_1^{(2)} = 2.0$.

Since no constraint will be violated we perform a pivot on y_{11}, updating only the first row.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
Z	0	0	0	1.5	1.3	1.1	0	27.0
x_1	1	0	0	0.45	-0.27	-0.09	0	2.0
$*x_3$	-2	0	1	0	0.4	0.5	0	1.4
$*x_2$	-0.5	1	-0.5	0	0	0.5	0	2.5
$*x_7$	-1	3	1	0	0	0	1	12.0

The above solution is optimal so we perform the final update on the right hand side

$x_3 = 1.0$, $x_2 = 4$, $x_7 = 1.0$

3.4 Group Four Methods

As mentioned earlier, the objective of the methods in group four is to consider redundancy in larger-scale mathematical programming problems.

3.4.1 Bradley *et al.* Method

Bradley *et al.* (1983) discussed an automated method for the exploitation of structural redundancy in a large-scale mathematical programming models. Their work deals primarily with row factorisation methods (eg. McBride (1973) and Graves and McBride (1976)) to identify the best embedded structure in any particular model. These structures are considered in increasing order of maximum row identification complexity. The efficient polynomial algorithms are operationally defined here as low-order polynomial in terms of intrinsic problem dimension (eg. number of rows, columns and non-zero elements), and not in terms of the total volume of model information. (eg. total number of bits in all coefficients). The efforts of Bradley *et al.* are devoted to two issues: analysis of the LP, and solving it efficiently. The analysis is confined to reductions that do not change the feasible region. The analysis can also be called "Orthogonal" in that the reduction tests are made on the current problem with no pivotal transformations actually performed.

The analysis is applied to a fully ranged and bounded linear program.

$$\begin{aligned} &\text{Min } \sum c_j x_j \\ &\text{S.t.} \quad r_i \le \sum a_{ij} x_j \le r_i \qquad \forall_i \text{ (ranged constraints)} \\ &\qquad\quad x_j^{\ell} \le x_j \le x_j^{u} \qquad \forall j \text{ (simple upper bounds)} \end{aligned} \qquad \ldots (3.4.1.1)$$

Some ranges and bounds may be missing (that is $+\infty$ or $-\infty$).

Bradley et al. presented a number of reduction analyses. Simple reduction tests are applied on the LP model. The same reduction tests have been reported by Brearley, Mitra and Williams (1975).

The elimination of an equation and column with a non-zero coefficient in the equation is discussed in the transformation reduction analysis. In particular, transformation reduction can generate a "reduced, equivalent LP" which is actually denser, and not necessarily as well-scaled as its progenitor.

Determining the set of Generalised Upper Bound (the set of rows for which each column has at most one non-zero coefficient restricted to the rows) have been dicsussed. An effective method to find maximal GUB sets was developed by Brearley et al. (1975). Also, Brown and Thomen (1980) have developed bounds on the size of the maximum GUB set which are sharp and easily computed.

Heuristic identification methods are presented, where an extension of GUB can be used to achieve NET ("Pure Network Rows" are a set of rows for which each column has at most two non-zero coefficients (restricted to those rows) are +1 and -1) factorisations. First GUB set is determined (Brearley et al. (1975), Brown and Thomen (1980)). Then second GUB set is found from an eligible subset of remaining rows, such that its row members must process non-zero coefficients of opposite sign in each column for which the prior GUB set has a non-zero coefficient.

Brown and Wright (1980) developed a method for direct NET factorisation of implicit network rows. With the same procedure by simple screening of admissible candidate rows, can be identified pure NET rows.

This heuristic is designed to perform network facotrisation of a signed matrix (0,1 entries only). It is a deletion heuristic which is, feasibility seeking. The measure of infeasibility at any point is a matrix penalty computed as the sum of individual row penalties. The algorithm is two-phased, one pass and non-backtracking. The first phase yields a feasible set of rows, while the second phase attempts to improve the set by reincluding rows previously excluded. Each iteration in Phase 1 either deletes a row or reflects it (multiplies it by -1) and guarantees that the matrix penalty will be reduced. Thus, the number of iterations in phase 1 is bounded by the initial value of the matrix penalty, which is polynomially bounded. The details of the method are included in Bradley _et al_. (1983).

3.4.2 Crowder et al. Method

Crowder et al. (1983) presented a method incorporated in PIPX (an experimental software package that they designed to solve pure (0-1) programming problems.), which includes automatic problem preprocessing and constraint generation. Problem pre-processing inspects the user-supplied formulation of a (0-1) linear program and improves on the associated linear programming formulation by "tightening" the constraint set, "spotting" variables that can be fixed at either 0 or 1, and "determining" constraints of the problem that are rendered inactive. Constraint generation essentially generates cutting-planes that are satisfied by (0-1) solutions of the problem and that chop off part of the feasible set of the linear programming relaxation and utilises the Branch-and-Bound strategy to find good integer solutions quickly. This procedure is used repeatedly and utilises information contained in the reduced costs associated with the optimal solution of the linear programming relaxation to fix variables to 0 or 1.

Crowder et al. attempted to establish the usefulness of these methodoligical advances - when combined with clever Branch-and Bound strategies for automatic solution of sparse large-scale (0-1) linear programming problems.

The following problem has been considered

$$\begin{aligned} \min \quad & CX \\ \text{S.t.} \quad & AX \leq b \qquad \qquad .. (3.4.2.1) \\ & x_j = 0 \text{ or } 1 \text{ for } j=1, \ldots, n \end{aligned}$$

where $A = (a_{ij})$ is mxn matrix, with $a_{ij} = 0, \pm 1, \forall i,j$, b and c are vectors of length m and n, respectively.

Problem Preprocessing

(I) Constraint Classification:

The inequalities of the problem (3.4.2.1) are classified into two types: type (1) constraints are special ordered set constraints, ie constraints of the type

$$\sum_{j\in L} x_j - \sum_{j\in H} x_j \le 1 - |H| \qquad \ldots (3.4.2.2)$$

where L and H are disjoint index sets and $|H|$ denotes the cardinality of the set H. Clearly $x_j = 1$ for some j, implies $x_k = 0$ for all $k \in L$, $k \neq j$, and $x_k = 1$ for all $k \in H$, while $x_j = 0$ for some $j \in H$ implies $x_k = 1$ for all $k \in H$, $k \neq j$, and $x_k = 0$ for all $k \in L$. Type (2) constraints are all other constraints of problem (3.4.2.1).

(II) Variable Fixing and Blatant Infeasibility check:

Suppose, for notational simplicity, that type (2) constraint of (3.4.2.1) is written as:

$$\sum_{j\in P} a_j x_j + \sum_{j\in N} a_j x_j \le b \qquad \ldots (3.4.2.3)$$

where P and N are the index sets of coefficients with positive and negative values respectively. If

$$\sum_{j\in N} a_j > b \qquad \ldots (3.4.2.4)$$

holds, then constraint (3.4.2.3) does not have a feasible solution and the overall problem (3.4.2.1), of which (3.4.2.3) is but one constraint, is blatantly infeasible. On the other hand if

$$\sum_{j \in P} a_j < b \qquad \ldots (3.4.2.5)$$

holds, the constraint (3.4.2.3) is inactive because every possible (0-1) vector x satisfies it. Such an inequaltiy can be dropped from the constraint set of (3.4.2.1) because it does not exclude any (0-1) solution. Let $j \in P$ and suppose that

$$a_j > b - \sum_{k \in N} a_k \qquad \ldots (3.2.4.6)$$

holds, then $x_j = 0$ in every feasible (0-1) solution to (3.4.2.1) and we can fix variable x_j at the value 0 and drop it from the problem (3.4.2.1). Likewise, if for some $j \in N$ we have

$$-a_j > b - \sum_{k \in N} a_k \qquad \ldots (3.2.4.7)$$

then $x_j = 1$ holds in every feasible (0-1) solution to (3.4.2.3). We can fix variable x_j at value 1, adjust the right-hand side vector b of (3.4.2.1) and drop the variable x_j from the problem (3.4.2.1). If a variable that is fixed at value 1 also appears in a type (1) constraint with a positive coefficient, the remaining variables in this special ordered set are fixed as discussed in the previous section; a similar argument holds if a variable that is fixed at value 0 appears also in a type (1) constraint with a negative coefficient . All type (2) constraints of problem (3.4.2.1) are examined one at a time in the order in which they appear in the formulation.

3.4.2.1.3 Coefficient Reduction

Consider an arbitrary linear inequality in the form

$$\sum_{j=1}^{r} a_j x_j \geq b \qquad \text{... (3.4.2.8)}$$

where all a_j for $j=1, \ldots, r$ are positive. If we have $a_k > b$ for some $k \in \{1, \ldots, r\}$, then we can replace a_k by b and the inequality

$$b x_k + \sum_{j=1,\ j \neq k}^{r} a_j x_j \geq b \qquad \text{... (3.4.2.9)}$$

has the same solution set in terms of (0-1) solutions as (3.4.2.8) but fewer real solutions in the unit-hypercube. Thus (3.4.2.9) is a "tighter" inequality that (3.4.2.8) for the associated linear programming relaxation. Of course, the constraints of (3.4.2.1) are not always of the form (3.4.2.8), but using the substitution $x_j' = 1 - x_j$ where necessary, we can bring every constraint of (3.4.2.1) into this form, apply this reasoning and check each coefficient of each type (2) constraint for a possible coefficient reduction.

Constraint Generation

The constraint generation procedure is the second computational phase of PIPX, to produce and solve a linear programming problem with a better optimal continuous objective function value. The real measure of the effectiveness of the constraint generation procedure is determined by how much it closes the "gap" between the optimal linear program relaxation objective function value and the optimal (0-1) objective function value.

In a large-scale (0-1) programming problem with a sparse matrix A and with no apparant special structure, it is reasonable to expect that the intersection of the m knapsack polytopes P_I^i $(^*CONV\{X\in R^n | a_i x \leq b_i,$ $x_j = 0$ or 1 for $j=1, \ldots, n\})$ provide a fairly good approximation to the (0-1) polytope P_I $(\underline{c} \bigcap_{i=1}^{m} P_I^i)$ over which to minimise a linear objective function. On the other hand, if the matrix is dense, then the different rows of A interact and cutting planes from individual rows of A, while certainly valid and in some instances useful, cannot be expected to produce the same impressive results that would come from sparse large-scale (0-1) problems with no apparent special structure. This is the first difference between Crowder et al. method and the traditional cutting-planes described in the text books on integer programs. The second difference, is the inequalities that Crowder et al. generate preserve the sparsity of the constraint matrix; on the other hand, the traditional cutting planes are typically rather dense and as integer programming folklore has it - lead to explosive storage requirement.

Crowder et at. modified the standard Branch-and Bound algorithm to facilitate the search, by computing the upper bound on the optimal solution and measuring the gap between the continuous optimal solution and the optimal (0-1) objective value to provide a good way of guiding to Mathematical Integer Programming Software to find integer solutions, and finally using the continuous reduced cost implication to fix the variables in the current Branch-and-Bound tree.

Finally, Crowder et al. mentioned that there are some computational difficulties in their constraint identification procedure because of the computer storage requirements. The other difficulty is the design and implementation of an effective and efficient interface between the computational procedure and the mathematical software for solving linear and integer programming problems.

(*CONV: The convexified solution).

IMPROVEMENTS

AND

EXTENSIONS

CHAPTER IV

In the previous chapter we presented the most promising size-reduction techniques. While the results of some of these techniques will be presented in chapter VI, we suggest here some changes which result in improving the performance of these techniques.

In this chapter, we present the details of two extended methods which have evolved from the previous ones. The first method called "Extended Reduce" is an improved version of the earlier Reduce method, in order to identify extraneous variables as well as redundant constraints. The second one is called "Extended-Williams' Procedure" for linear and integer programs, which is an extended version of Williams' procedure.

Before we proceed with the details of each method, and to avoid any repetition in the terminology and notations, we restate our (primal) linear programming problem as:

$$\begin{array}{ll} \max & Z = CX \\ \text{S.t} & AX \leq b \\ & X \geq b \end{array} \qquad \ldots (4.1)$$

and the dual problem associated with the above system is:

$$\begin{array}{ll} \min & Y = Wb \\ \text{s.t.} & WA' \geq c \\ & W \geq 0 \end{array} \qquad \ldots (4.2)$$

where A is an mxn matrix, A' is the transpose of A, C and X are n vectors, b and W are m vectors.

4.1 Extended Reduce Method

As mentioned before, the Extended-Reduce method is an improved version of the earlier Reduce method presented in chapter III. The method is to identify extraneous variables and redundant constraints. Also, redundant constraints are identified by implementing a modified version of the co-ordinate direction method at certain steps if necessary. Based on the following modifications involving more efficient tests from some theorems presented in chapter II on both primal and dual, together with a modified version of the coordinate direction method, the Extended Reduce method is developed.

We utilise the same notation developed in chapter II and in Boneh and Golan's method presented in chapter III. Namely, we use the constructed tableau A(mxn) and denote its elements by a_{ij}. The updated right-hand side vector is denoted by b(mx1) and its associated elements by b_i. The reduced cost vector is denoted by Z - C (1xn) and its associated elements by $Z_j - C_j$. Also, the vector of basic variables is x_j^B and that of non-basic variables is x_j^N.

The results from experiments on Extended Sign Tests, Hybrid and Reduce methods, presented in chapter VI, show that test two and its dual test are unhelpful and expensive (in terms of computation times), hence they are not considered here. On the other hand, test one and its dual test as well as step two of Reduce method (ie. a constraint having non-negative entries and a zero right hand side, then a variable with a positive entry in this row is extraneous) are found most useful. Test five is found most efficient when it is used as part of the simplex step.

We especially attempt to make use of this test to identify redundant constraints, by implementing the modified version of the co-ordinate direction method with it.

The results of the co-ordinate direction method from experiments on the Hybrid method seems very efficient (in terms of computations time). However, identifying non-negativity constraints as redundant tells us very little about their variables, since their values may turn out equal to zero or not. Also, the existence of extraneous variables in the problem may affect the results by classifying some redundant constraints as non-redundant and this occurs because of perturbing the problem where extraneous variables could have small positive values in an interior feasible point. Secondly, when the direction from the interior feasible point to all constraints is along one of the extraneous variables, difficulties can also arise. To explain this, let us consider the following example:

$$\begin{array}{lllr}
\max & x_1 - x_2 + 2x_3 & & \\
\text{s.t.} & & & \\
 & x_1 + x_2 & \leq 2 & \ldots\ldots \text{R1} \\
 & -x_1 + 3x_2 + x_3 & \leq 2.5 & \ldots\ldots \text{R2} \\
 & x_1 + x_2 + x_3 & \leq 2 & \ldots\ldots \text{R3}
\end{array}$$

by perturbation of the problem, the interior feasible point is $(0.01, 0.01, 0.01)$. Clearly x_2 is extraneous, but if the direction from the interior feasible point to all constraints moves along x_2, R_2 is classified as non-redundant, which it is in fact redundant.

As a result of the above difficulties, we modify the co-ordinate direction method to be used with test five and only when the pivot ratio is not unique, in order to identify redundant constraints before we perform a simplex iteration. First we consider only the structural constraints having the same pivot ratio value, and positive coefficient corresponding to the variable, which has been taken as a current direction. Second, in order to be sure that the direction is not along any of the extraneous variables, perform the test along the next pivot column, which is easy to identify by simply updating the objective function. Third, in order to be sure that none of the extraneous variables could have any positive number, we start with the boundary point instead of the interior point, and we perturb only the slacks of non-negativity constraints which are in the basis, and all other variables must have zero value.

Given a boundary or interior feasible point X^0, the distance t_i between any constraint and X^0 along the j-th direction is given as follows:

$$t_i = \frac{b_i - A_i X^0}{a_{ij}} , \quad a_{ij} > 0 \qquad \ldots (4.1.1)$$

where i is the constraint index having the same pivot ratio value; of course A_i is the i-th constraint of the original problem (4.1).

Therefore, if the i-th constraint has a minimum value t_i, then the other constraints classify as redundant. Moreover, the i-th constraint becomes a pivot row for the simplex iteration.

As a result of tests, such modification is computationally beneficial, since it is less expensive (in terms of computations time) to identify redundant constraints, where great saving in time and storage space have been achieved, since the total number of arithmetic operations to compute (4.1.1) reduces from (m+n)(2n+2) for computing equation (3.1.1.3) to at most m(2n+2). Furthermore, there is no need to convert the original matrix problem into the form of "$\geq$". Finally, the number of simplex iterations to reach the optimum solution could be reduced, and that is due to the right choice of pivot constraint (when the pivot ratio is not unique).

Now, we present the Extended-Reduce method in algorithmic steps:

Initial Step: Let $H = \{k \mid S_k = x_i^B\}$ and $G = \{r \mid S_r = x_j^N\}$ where H and G are the set of indices of the slack variables in rows and columns still remaining in the problem.
Store $AX \leq b$, find a basic feasible solution to the system (4.1).

Step (1): If all $z_j - c_j \geq 0$, stop. Otherwise continue with step 2.

Step (2): For every column j with $x_j^N = S_r$ and $r \in G$, check the property:

$$a_{ij} \geq 0 \text{ for all } i \text{ and } z_j - c_j \geq 0$$

If this holds, drop column j and remove r from G.

Step (3): For every row i with $x_i^B = S_k$ and $k \in H$, check the property:

$$a_{ij} \leq 0 \text{ for all } j$$

If this holds, drop row i and remove k from H.

Step (4): For every row i with $x_i^B = S_k$ and $k \in H$, check the property:

$$a_{ij} \geq 0 \text{ for all } j \text{ and } b_i = 0$$

If this holds, drop all columns with $a_{ij}>0$ and remove all their indices from G. Then drop row i and remove k from H.

Step (5): Determine the non-basic variable $S_r = x_j^N$ with the most negative reduced cost $Z_p - C_p$. Compute:

$$b_t/a_{tp} = \min_i \{b_i/a_{ip} | a_{ip}>0\}$$

If the above ratio is unique then $S_k = x_t^B$, $k \in H$ is a slack of a non-redundant constraint, and go to step 7. Otherwise continue with step 6.

Step (6): Determine the latest boundary or interior feasible point, and the next pivot column j. Among only constraints having the same ratio value, determine the constraint with minimum t_i. Drop the other constraints from the problem, and their indices from H.

Step (7): Perform a simplex pivot iteration, and update the table. If no rows or columns have been removed, stop. Otherwise go to step 1.

Now, to illustrate the use of our extended reduce method, we consider the following numerical example:

$$\begin{array}{ll} \max & 2x_1 + x_2 - x_3 + 3x_4 - 3x_5 \\ \text{s.t.} & \\ & x_1 + x_2 + x_5 \leq 1 \\ & 2x_3 + 2x_4 + x_5 \leq 4 \\ & -x_1 + x_3 + x_4 \leq 4 \end{array}$$

$$
\begin{aligned}
x_1 \qquad\qquad\quad + 4x_4 - x_5 \quad &\leq 8 \\
x_1 + x_2 + x_3 + 3x_4 \qquad\quad &\leq 6 \\
x_j \quad &\geq 0 \quad \text{for all } j.
\end{aligned}
$$

Initial Step: In what follows, we label the slack variables as s_1 through s_5 and x_1 through x_5 as s_6 through s_{10} respectively.

The constructed tableau is:

T_0 =

	s_6	s_7	s_8	s_9	s_{10}	b
$Z_j - C_j$	-2	-1	1	-3	3	0
s_1	1	1			1	1
s_2			2	2	1	4
s_3	-1		1	1		4
s_4	1			4	-1	8
s_5	1	1	1	$\boxed{3}$	1	6

The index sets H = {1,2,3,4,5} and G = {6,7,8,9,10}

Step (2): S_8 is extraneous, we drop column 3. G = {6,7,9,10}

Step (5): The pivot ratio is chosen for column four (with most negative $Z_4 - C_4 = -3$), and it is not unique.

Step (6): The next pivot column $j = 1$ $(Z_1 - C_1 = -1)$,

$X^0 = (0.01, 0, 0, 1.99, 0)$

$t_4 = \frac{0.03}{4}$,

$t_5 = \frac{0.02}{3}$

Therefore constraint 5 is non-redundant, constraints 2 and 4 are redundant.

Step (7): After pivoting on $a_{54} = 2$, we get the following updated table:

		S_6	S_7	S_5	S_{10}	b
	$Z_j - C_j$	-1	2	1	3	-6
	S_1	$\boxed{1}$	1		1	1
T_1:	S_3	$-\frac{4}{3}$		$-\frac{1}{3}$		2
	S_9	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	2

$H = \{1,3,9\}$ $\qquad$ $G = \{6,7,5,10\}$

Step (2): S_7 and S_{10} are extraneous, we drop columns 2 and 5.
$G = \{6,5\}$

Step (3): Row 3 redundant, we drop row 3. $H = \{1,9\}$

Step (5): The pivot ratio on column one is chosen (with most negative $Z_1 - C_1 = -1$), and it is unique.

Step (7): After pivoting on a_{11}, we get the following updated table:

	S_1	S_5	b
$Z_j - C_j$	1	1	
S_6	1		1
S_9	$\frac{-1}{3}$	$\frac{1}{3}$	$\frac{5}{3}$

and $H = \{6,9\}$, $G = \{1,5\}$

Step (1): All $z_j - c_j \geq 0$, the solution is optimum, stop.

The test results of Extended-Reduce method are presented in chapter VI.

4.2 Extended Williams Procedure

Unlike the previous improved method, the size of linear (and integer) programming problems has been reduced prior to applying the simplex method. The procedure presented here is an extended version of Williams' procedure achieved by combining another test based on theorem (3.1) presented in Holm and Klein's method in chapter III, in order to identify extraneous variables. More suggestions have been made to reduce the course of processing. Based on the above, we developed "Extended Williams Procedure".

In order to present the mathematical theory used in the extended procedure we will utilise the same terminology implemented in Williams' procedure and Holm and Klein's method presented in chapter III. Initially, for all the variables (primal and dual), the lower bounds are set to zero (because of the non-negativity constraints) and the upper bounds at a sufficiently large real number.

As a result of testing Williams' procedure presented in chapter VI, the structure of the tested problem (redundancy and degeneracy) is affected on its reduction processing, and that is due to unsuccessful tightening of the bounds of primal and dual variables. Then, the required conditions in test D2 to fix variables at their bounds are affected and not easy to hold. The results show that most of the variables having non-zero coefficients in all constraints with zero lower bounds in shadow prices are not fixed to their bounds. To demonstrate this, consider the following example

$$\max \quad Z = x_1 + \dots$$

s.t.

$$x_1 + \dots \quad \leq 5 \qquad \_\_\_\_ \text{ R1}$$

$$-x_1 + \dots \quad \leq 8 \qquad \_\_\_\_ \text{ R2}$$

Suppose that $x_1 = 0$ at the optimal solution, and $w_1^\ell = w_2^\ell = 0$. Williams' procedure is unable to fix such a variable at its lower bounds zero, and such variables may affect the whole procedure of size reduction.

Holm and Klein (1975) identify extraneous variables by pairwise comparisons between variables, based on theorem (3.1) presented in chapter III, which we may restate as a test as follows:

If there exists column indices r and j such that

$$c_r - c_j > w^u \text{ pos}(A(.r) - A(.j)) + w^\ell \text{ neg}(A(.r) - A(.j)) \qquad \dots (4.2.1)$$

then x_j is extraneous.

The basic idea of the above test is from the Complementary Slackness Theorem, ie. a variable $x_j^0 = 0$ whenever $c_j - w^0 A(.j) < 0$ where X^0 and w^0 are the optimal solution to (4.1) and (4.2), repsectively. However, as the test covers most of the situations, and the pairwise comparison needs a little more processing, we decided to combine such a test with Williams' procedure, in a way to reduce the pairwise comparisons time processing in the whole procedure, by not repeating the pairwise comparisons processing in each pass, if neither any singleton columns are replaced by shadow price bounds nor any constraints removed nor any shadow price bounds tightened.

In fact, we are using the same names for tests as in the original Williams' procedure, such as test P1, test D1, ..., etc (see Williams' procedure in chapter III) in presenting the algorithmic steps of Extended Williams procedure. As in Williams' procedure all the tests are implemented in the same systematic approach, and our procedure also has two phases, to resolve the dilemma over whether to relax or tighten the bounds on primal and dual variables. On the other hand, as a result of testing Williams' procedure, we suggest, first, a phase of the procedure is terminated when one pass yields no simplification. Second, there is no need to repeat the whole procedure processing in part two, if neither singleton columns are replaced by shadow price bounds nor constraints with non-zero lower shadow price bounds are removed by subtracting from the objective function.

We now present the details of our extended procedure in an algorithmic form. The following logical variables are used as switches for various steps

PART = F for part 1
= T for part 2

PHASE = F for phase 1
= T for phase 2

PSACT = T changes made during the current pass
= F otherwise

PRDSC = T changes made either by replacing singleton column or removing constraint with non-zero shadow price bounds
= F otherwise

Initial Step : Set all logical variables to F, all lower bounds (primal and dual) to zero and all upper bounds to a large real number M.

Step (1): Let K = 1

Step (2): Let j be the k-th index of smallest element of C;

Step (3): Compute $\bar{x}_j^\ell$ and $\bar{x}_j^u$ (equations 3.2.2.9 - 10)

Step (4): If $\bar{x}_j^\ell > x_j^u$ or $\bar{x}_j^u$, or $\bar{x}_j^u < x_j^\ell$ or $\bar{x}_j^\ell$, the model is infeasible, stop

Step (5): If $\bar{x}_j^\ell = x_j^u$ or $\bar{x}_j^u$, or $\bar{x}_j^u = x_j^\ell$ or $\bar{x}_j^\ell$, set x_j to this common value, substitute out, set PSACT = T and go to step 2.

Step (6): If PHASE if T, go to step 8.

Step (7): If the new primal bounds are more strict than existing values, update these bounds and set PSACT = T. Otherwise go to step 9.

Step (8): If the new primal bounds are more strict than existing values, restore the initial bounds to x_j.

Step (9): Compute P_j and Q_j (equations 3.2.2.3 - 4).

Step (10): Perform test D2, if changes made set PSACT = T, and go to step 2. Otherwise, continue with step 11.

Step (11): If PART is T, go to step 13. Otherwise, continue with step 12.

Step (12): Perform test D1, if changes made, set PSACT = PRDSC = T, and go to step 23. Otherwise continue with step 13.

Step (13): Compute $\bar{w}_i^\ell$ and $\bar{w}_i^u$ (equations 3.2.2.11 - 12)

Step (14): If $\bar{w}_i^\ell > w_i^u$ or $\bar{w}_i^u$, or $\bar{w}_i^u < w_i^\ell$ or $\bar{w}_i^\ell$, the model is either unbounded or infeasible, stop.

Step (15): If $\bar{w}_i^\ell = w_i^u$ or $\bar{w}_i^u$, or $\bar{w}_i^u = w_i^\ell$ or $\bar{w}_i^\ell$ multiply the constraint i by this common value, subtract from the objective function, remove the constraint i, set PSACT = PRDSC = T, then go to step 19.

Step (16): If PHASE is F, go to step 18.

Step (17): If the new shadow price bounds are more strict than existing values, update these bounds, and set PSACT = T. Otherwise fo to step 19.

Step (18): If the new shadow price bounds are more strict than existing values, restore the initial bounds to the dual variables.

Step (19): Let L=0.

Step (20): If there are no more columns to be compared with c_j, go to step 23. Otherwise, set L = L+1, and continue with step 21.

Step (21): Let r be the L-th index of largest element of C

Step (22): If condition (4.2.1) is satisfied, remove column j, set PSACT = T, and go to step 24. Otherwise go to step 20.

Step (23): Compute L_i and U_i (equations 3.2.2.1 - 2).

Step (24): If there are no more columns left (ie. K equals N), go to step 25. Otherwise set k = k+1 and go to step 2.

Step (25): Perform test P2, if changes made, set PSACT = T, moreover if removed constraints have non-zero shadow price bounds set PRDSC = T.

Step (26): Perform test P1, if change made, set PSACT = T.

Step (27): If PSACT is F, go to step 29.

Step (28): Set PSACT = F and go to step 1.

Step (29): If PHASE is F, set PHASE = T and go to step 1.

Step (30): If PART is T, stop.

Step (31): If PRDSC = T, restore all singleton columns and constraints with non-zero shadow price bounds subtracted from the objective function, set PART = T, PHASE = F and go to step 1. Otherwise, stop.

To illustrate the use of the above algorithm, let us consider the following numerical example taken from Williams (1983), after modification. Without affecting the feasibility or the optimal solution, further reductions are found, where Williams' procedure failed to reduce its size:

max	$2x_1$	$+ 3x_2$	$+ x_3$				w_i^ℓ	w_i^u
S.t.								
R1	$-x_1$	$+ x_2$	$+ x_3$	$+ x_4$	$- 2x_5$	≤ 4	0	M
R2	$-x_1$	$- x_2$	$+ x_3$	$+ x_4$	$- x_5$	≤ 1	0	M
R3	x_1			$+ x_4$	$+ x_5$	≤ 3	0	M
x_j^ℓ	0	0	0	0	0			
x_j^u	M	M	M	M	M			

Solution:

PART ONE

PHASE ONE

PASS (1)

X_4 extraneous

X_3 extraneous

constraint 2 redundant

PASS (2)

$X^u_5 = 3, \quad X^u_1 = 3.$

Lower shadow price bound on constraint 1 is 3.

PASS (3)

constraint 1 redundant

PASS (4)

X_1 extraneous

constraint 3 redundant

PASS (5)

$X_5 = 3.0$

PASS (6)

nothing

PHASE TWO

PASS (1)

PASS (2)

PART TWO

PHASE ONE

PASS (1)

$X^u_2 = 10.0$

$X_2 = 10.0$ and the problem solved.

STOP

The results of the Extended Williams' procedure are presented in chapter VI.

CHAPTER V

In the previous chapter, two reduction methods are presented, mainly for Linear programs. In this chapter extended techniques are presented mainly for integer programs.

The requirement that the variables must take integer values is a mathematical extension of Linear programming, which is known as Pure Integer Programming. There are many ways of solving such problems, however, there is only one method which purports to be applicable to all such problems and is sometimes presented as a simple extension to cope with integer variables in the LP algorithm of commercial packages - the so-called "Branch-and-Bound" algorithm.

As the problem size increases, the amount of work needed to produce an integer optimum solution may increase exponentially, where, subproblems are generated and the number of branches increases as the number of integer variables increases in the problem. In general, there are unnecessary variables and rows in a model formulation which increase the number of branches and the solution time. Therefore, reducing the size of the problems by removing unnecessary variables and rows will reduce the number of branches required in order to solve the problems, using Branch-and-Bound algorithm efficiently.

In this chapter, we present a preprocessing reduction procedure for general integer linear programming problems, and discuss its implications for Dynamic-Presolve which is a feature of the SCICONIC package. Also, reductions to subproblems having Special Order Sets (SOS) will be presented.

Before we proceed with the details, to avoid repetition we state our (primal) integer linear programming problem as follows:

$$\text{Max } Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \quad \sum a_{ij} x_j \leq b_i \qquad i=1,\ldots,m \qquad \ldots(5.1)$$

$$x_j \geq 0 \quad \text{and integers}$$

and for each variable there are finite integer lower and upper bounds

$$0 \leq x_j^{\ell} \leq x_j \leq x_j^{u} \qquad \ldots(5.2)$$

5.1 Preprocessing Reduction Procedure for ILPP's

A preprocessing technique is developed to reduce the size of general ILPP's using the primal bounds to fix variables at their bounds and identify extraneous variables and redundant constraints prior to applying the simplex and Branch-and-Bound algorithms.

In order to present the mathematical theory used in our procedure, we use the same terminologies as in "Holm and Klein's" and "Williams'" methods presented in Chapter III.

With integer variables it is generally advantageous to tighten the bounds rather than relax them since it may be possible to tighten the bound to the next appropriate integer value. The bounds have been tightened in our procedure in a fashion similar to that of Williams' techniques, that is, a constraint together with bounds on some variables may imply bounds on another variable (equations 3.2.2.9-10), and if $\bar{x}_j^u < x_j^u$, the upper bound x_j^u is replaced by $[\bar{x}^u + \epsilon]$. If $\bar{x}_j^{\ell} > x_j^{\ell}$, the lower bound x_j^{ℓ} is replaced by $[\bar{x}_j^{\ell} - \epsilon]+1$, where ϵ is a small positive number. Should $\bar{x}_j^{\ell}$ be equal to $\bar{x}_j^u$, the variable x_j may be fixed at this common value and removed from the problem by replacing b_i by $(b_i - a_{ij}\, x_j)$ for all i and adding the constant $c_j x_j$ to the objective function.

However, Brearley et al. [1975] and Williams [1983] identify constraint i in a system (5.1) as redundant if $U_i \leq b_i$ provided that it does not have a nonzero lower shadow price. Williams [1978] mentioned that, in integer problems, if a constraint has a positive slack it does not necessarily represent a "free good" (i.e., in one sense it is not worth anything) and may therefore have a positive economic value (see Williams [1978], Ch. 10).

Rubin [1972], extended the results of test one to apply to integer problems, by presenting the following theorem:

Theorem Rubin [1972]

If row i is a structural constraint having

$a_{ij} \leq 0$ for all j and $B_i \geq 0$ ...(5.1.1)

then it is redundant in IP.

Since no simplex iterations have been performed during the course of our procedure, all rows are structural constraints, therefore, we decided to use the above test to identify redundant constraints.

As a result of the above test, many redundant constraints could not be identified, because condition (5.1.1) was not satisfied. We decided to implement Holmand Klein's test, presented in chapter III, in order to identify redundant constraints by pairwise comparisons between constraints, based on theorem (3.2), (condition 3.2.1.6). However, as the pairwise comparisions need more time processing, we combined and performed this test in a way to reduce the pairwise comparisons time processing as much as we can, such as terminating the test as soon as the right-hand side of (3.2.1.6) becomes greater than or equal to the left-hand side.

In our procedure we construct formulae using only primal bounds to fix the variables at their bounds, as follows:

Case (a): If $k \in P$, and

$$a_{ik} > b_i - \left(\sum_{\substack{j\in P\\ j\neq k}} a_{ij}\, x_j^{\ell} + \sum_{j\in N} a_{ij}\, x_j^{u}\right) \qquad \ldots(5.1.2)$$

holds, then $x_k=0$ at every feasible solution to (5.1)

Case (b): If $k \in N$, and

$$-a_{ik} > b_i - \left(\sum_{j\in P} a_{ij}\, x_j^{\ell} + \sum_{\substack{j\in N\\ j\neq k}} a_{ij}\, x_j^{u}\right) \qquad \ldots(5.1.3)$$

holds, then $x_k = x_k^u$ at every feasible solution to (5.1)

where P and N are the index sets of coefficients with positive and negative values, respectively. The correctness of the above two cases comes from the feasibility of the system (5.1).

The above two formulae need good tightened bounds to fix more variables, therefore one may identify extraneous variables by the dual test to condition (5.1.1), which may be stated in the following corollary:

<u>Corollary</u>:

If column j is not a slack of a structural constraint and has

$a_{ij} \geq 0$ for all i and $c_j \leq 0$ ...(5.1.4)

then x_j is extraneous in a system (5.1)

The correctness of the above corollary is from the validity of its duality.

Now, let us present our procedure in algorithmic steps:

Initial Step: Set PASS = 1, PSACT = F

Step 1: let j =1,

Step 2: If condition (5.1.4) is satisfied, remove column j, set PSACT=T and go to step 9. Otherwise continue with step 3.

Step 3: If any of conditions (5.1.2-3) is satisfied, update the problem, remove column j, set PSACT = T and go to step 9. Otherwise, continue with step 4.

Step 4: If PASS = 1, go to step 8. Otherwise, continue with step 5.

Step 5: Compute $\bar{x}_j^{\ell}$ and $\bar{x}_j^{u}$ (equations 3.2.2.9-10);

Step 6: If $\bar{x}_j^{\ell} > x_j^{u}$ or $\bar{x}_j^{u}$, or $\bar{x}_j^{u} < x_j^{\ell}$ or $\bar{x}_j^{\ell}$, the problem is infeasible, stop.

Step 7: If the new bounds are more strict than existing values, update these bounds, set PSACT = T, and if the lower and upper bounds on x_j are equal, set x_j to this common value, update the problem, remove column j, then go to step 9. Otherwise, continue with step 8.

Step 8: Compute L_i (equation 3.3.3.1);

Step 9: If no more columns left, continue with step 10. Otherwise, set j=j+1 and go to step 2.

Step 10: let k = 1,

Step 11: let i be the k-th index of largest element of b;

Step 12: If condition (5.1.1) is satisfied, remove row i, set PSACT=T and go to step 17. Otherwise, continue with step 13.

Step 13: let L = 1,

Step 14: let t be the L-th index of smallest element of b;

Step 15: If condition (3.2.1.6) is satisfied, remove row i, set PSACT=T and go to step 17. Otherwise, continue with step 16.

Step 16: If there are no more rows to be compared with b_i, go to step 17. Otherwise, set L=L+1 and go to step 14.

Step 17: If there are no more rows left, go to step 18. Otherwise, set k=k+1 and go to step 11;

Step 18: If PASS = 1, set PASS = PASS + 1, PSACT = F and go to step 1. Otherwise, if PSACT = T, set PASS = PASS +1, PSACT = F and go to step 1. Otherwise, stop.

Now, we present the following numerical example to demonstrate our procedure:

$$\text{Max } 2x_1 + 3x_2 - x_3 - x_4$$

s.t.

$$\begin{aligned}
x_1 + x_2 + x_3 - 2x_4 &\le 4 \\
-x_1 - x_2 + x_3 - x_4 &\le 1 \\
x_1 \qquad\qquad + x_4 &\le 3 \\
2x_1 \qquad + x_3 - 2x_4 &\le 1
\end{aligned}$$

$$0 \le x_j \le 10 \text{ for all } j, \text{ and integers}$$

Solution:

Pass 1:

$x_1^u = 3$, $x_4^u = 3$

X(3) = 0

X(4) = 3

Constraint 2 redundant

Pass 2:

X(1) = 0

x(2) = 10

Stop

5.2 The implication of implementing Preprocessing Reduction Procedure to "Dynamic-Presolve"

Integer problems can be solved by the SCICONIC Package (an algorithmic advanced Mathematical Programming Package), by calling the command "GLOBAL", and with the parameter "PRESOLVE" a Dynamic-Presolve is performed on each sub-problem in the Branch-and-Bound search. It attempts to reduce the discrepancy between the linear solution to each sub-problem and the true optimum for which we are searching, and makes the current sub-problem easier to solve by fixing continuous variables at their lower bounds and tightening the bounds on the variables.

Unfortunately, the Dynamic-Presolve technique becomes less powerful when a branching decision is made on a variable with negative coefficients in many or all constraints. Implementing our preprocessing reduction procedure within the Dynamic-Presolve technique on each sub-problem, could make the whole hybrid processing more powerful in making the current sub-problem much easier to solve, and saving more work in less CPU time.

To show how our procedure, works and could improve the processing of the Dynamic-Presolve technique, let us consider the following example:

Example:

Suppose at a certain subproblem the integer variables x_1, x_2 and x_3 with lower bounds zero and upper bounds 3, appear in the following constraints:

$$R1: \quad 2x_1 + 4x_2 - x_3 \leq 7$$

$$R2: \quad -5x_1 + 2x_2 - x_3 \leq 1$$

and at some branch, we might make the branching decision $x_3 \geq 2$.

Now as far as we know, Dynamic-Presolve technique is unsuccessful in tightening the bounds of these variables, but our procedure may continue the processing by fixing x_1 to 3, tightening the upper bound of x_2 to 2, then removing constraint two, making the current sub-problem much easier to solve than implemented only the Dynamic-Presolve technique.

5.3 <u>Reduction techniques for Special Order Sets</u>.

Special Ordered Sets (SOS) are sets of variables with an explicitly or implicity given order and a specified additional condition. They were introduced by Beale and Tomlin [1970], as a practical device for efficiently handling special classes of non convex optimization problems by Branch-and-Bound with LP relaxation and are now implemented in most commercial codes for mathematical programming. There are two types:

Type 1 (SOS1 set), where only one variable in the set can have a nonzero value. If the variables x_j are not 0-1, indicator 0-1 variables $\delta_1,\ldots\delta_n$ are introduced and linked to the x_j variables. Type 2 (SOS2 set), where up to two adjacent variables in the set can have nonzero values. The model is slightly more complicated, and the problem can be subdivided into two sub-problems by choosing a suitable value of j, say r, in a suitable reference row. So in SOS1 : in one branch $\delta_j=0$ for all $j > r$ and in the other $\delta_j = 0$ for all $j \leq r$, while SOS2: either $\delta_j = 0$ for all $j > r$, or $\delta_j = 0$ for all $j < r$.

The strategy of fixing several variables to zero simultaneously is one reason for the success of the special ordered set (SOS) branching rule (see Beale and Tomlin [1970], Forrest <u>et.al.</u>[1974], Gauthier and Ribiere [1977] and Tomlin [1970])on integer programs with multiple choice constraints.

In fact, in implementing Branch-and-Bound strategies for SOS in commercial codes for MP, the members of an SOS must form a monotonic ascending or descending sequence which is defined by weights w_j and maintained throughout the whole branching process. Otherwise, there is no suitable way developed to determine the branching point. Determining an average weight $\bar{w}$:

$$\bar{w} = \Sigma\, w_j\, x_j^o / \Sigma\, x_j^o \qquad \ldots(5.3.1)$$

where x^o are the values of the set variables in the optimal solution of the LP relaxation, the branching point r is then defined either by

$$w_r \leq \bar{w} < w_{r+1} \text{ or } w_{r-1} < \bar{w} \leq w_r$$

We present some techniques to identify which variables of an SOS set could have zero values or nonzero values, in order to reduce the sub-problem, even if the SOS set does not have a suitable reference row.

Suppose $(x_1, \ldots., x_n)$ is an SOS set, and they are a part of the problem, appearing in the following constraints:

$$\sum_{j \in P} a_{ij}\, x_j + \sum_{j \in N} a_{ij}\, x_j \leq b_i \qquad \ldots(5.3.2)$$

where P and N are the index set of coefficients with positive and negative values, respectively.

The following tests may be used to reduce the SOS set in a sub-problem:

(I) If $\forall j$

$$a_{ij} > (b_i / x_j^u) \qquad \ldots(5.3.3)$$

then $x_j = o$ in every feasible solution at SOS1 set.

(II) If $\exists$ a unique $j \ni$

$$a_{ij} < (b_i / x_j^u) \qquad \ldots(5.3.4)$$

then:

(a) - $x_j = x_j^u$ in every feasible solution at SOS1 set.

(b) - x_j will take a nonzero value qx^u ($q \leq 1$) in every feasible solution, and either x_{j-1} or x_{j+1} could take a nonzero value $(1-q)x^u$ at SOS2. Moreover, if all variables are integers, one may use conditions (5.1.2-3) to fix x_{j-1} and x_{j+1} at their values.

Now,we present the following examples to demonstrate our tests:

Example (1):

Suppose (x_1,x_2,x_3,x_4) are SOS1 and form a part of the problem, and appear in the following constraint:

$$x_1 - 2x_2 + 2x_3 - x_4 \leq -3$$

with bounds of 2.

Implementing our tests may fix $x_1 = x_3 = x_4 = o$ and $x_2 = 2$.

Example (2):

Suppose (x_1,x_2,x_3,x_4) are SOS2 and form a part of the problem, and appear in the following constraint:

$$4x_1 - 2x_2 + x_3 + x_4 \leq -1$$

with bounds of 2, and all are integers.

Implementing our tests may fix $x_1 = x_4 = o$ and x_2, x_3 to nonzero values.

Note: It was not possible to test all the procedures of this Chapter within the Sciconic computer code because the modular capability of the LP code does not extend to the Branch and Bound part.

CHAPTER VI

Programming the Methods and Experimental Results

Some of the size-reduction techniques presented in chapter III and all the extended methods presented in chapters IV and V have been programmed and tested on the Prime Computer System at Loughborough University.

In this chapter, we present some important basic techniques in programming the size-reduction techniques. The structural tested problems, the results and discussion of the results will be the subject of the remainder of this chapter.

6.1 Programming the Methods

The FORTRAN 66 computer language was used for programming the methods, following advice from staff at SCICON Computer Services Ltd.

The SCICONIC package is an algorithmically advanced Mathematical Programming Package developed by SCICON. Its purpose is to provide the mathematical programmer with a convenient and cost-effective way to solve linear, integer and non-linear programming problems. In particular SCICON developed SATL Sciconic Algorithmic Tools library) which allows the user to assemble modules of SCICONIC to his own specification.

We built programs in the form of a sub-routine called "SUBROUTINE USER" which was loaded into a space already designated for a trivial subroutine called "USER" in the package. Then we applied this subroutine as a preprocessor after loading and converting the input data file, and before executing the main LP algorithm (for more details see appendices).

Sciconic stores non-zero elements of the data matrix in column order. All the non-zero matrix column elements are stored in an element pool "array POOL" (the element pool is based on an idea of Kalan (1977)). which only contains unique values; individual matrix elements may be accessed from the pool via the arrays of pointers. This enables the input data to be stored in a very compact form, taking the maximum advantage of matrix sparsity and any non-uniqueness of the matrix elements. Matrix entries are accessed from the POOL by two parallel arrays, the entries within which are stored by columns. If the column has a cost row and/or an upper bound, then there is an additional entry in the parallel arrays.

For certain manipulations, in some tests (such as singleton row "Williams' procedure", the number of non-zero elements and their signs in each row "sign tests" and in order to perform the pairwise comparison columns "Holm and Klein 's method"). it is convenient to have the elements easily accessible in row order as well as in column order. Therefore, some additional storage arrays were created to store the elements of the matrix in a different way. This would let us build the programs using one dimensional arrays instead of using two dimensional arrays as some problems occured in the storage methods with the two dimensional arrays. The one dimensional arrays are packed to save as much space as possible.

We can explain how we managed to store the matrix in one dimensional arrays, by considering the following example.

$$\begin{array}{llcl}
 & \text{Max.} & x_1 + 2x_2 + 4x_3 + 10x_4 + x_5 & \\
 & \text{s.t.} & & \\
\text{R1} & ______ & x_1 + x_2 + x_3 + x_4 + x_5 & \leq 25 \\
\text{R2} & ______ & 2x_1 + x_4 & = 10 \\
\text{R3} & ______ & x_2 + x_3 + x_5 & \leq 5 \\
\text{R4} & ______ & 2x_1 + 4x_3 + x_5 & \geq 15 \\
\text{R5} & ______ & 8x_3 + x_4 & \leq 10
\end{array}$$

Let there be three arrays ROWELL "real", IROWNO "integer" and IROWMK "integer". IROWMK has a dimension of 512, the other two have dimensions of 8192 (equivalent to 16 x 512). IROWNO is created as follows:

IROWNO (1) tell us how many non-zero elements are in R1,

IROWNO (2), (3), tell us the columns in which the non-zeros occur. So

IROWNO (1) = 5, IROWNO (2) = 1, IROWNO (3) = 2, IROWNO (4) = 3, IROWNO (5) = 4, IROWNO (6) = 5.

The next item in IROWNO namely IROWNO (7) tells us how many non-zeros occur in R2 and IROWNO (8), (9), ... tell us where they are. The procedure then repeats for R3, R4 and R5.

Tha actual values of coefficients are now stored in the corresponding positions of array ROWELL:

R1: ROWELL (2) = 1.0, ROWELL (3) = 1.0, ROWELL (4) = 1.0,
ROWELL (5) = 1.0, ROWELL (6) = 1.0

R2: ROWELL (8) = 2.0, ROWELL (9) = 1.0

R3: ROWELL (11) = 1.0, ROWELL (12) = 1.0, ROWELL (13) = 1.0

R4: ROWELL (15) = 2.0, ROWELL (16) = 4.0, ROWELL (17) = 1.0

R5: ROWELL (19) = 8.0, ROWELL (20) = 1.0.

ROWELL (1), (7), (10), (14) and (18) are not used (but could be set to indicate 1, 0, - 1 for $\leq$, =, $\geq$ if required).

The third array IROWMK tell us where the set of information in one row actually begins in IROWNO. Hence IROWMK(1) = 1, IROWMK(2) = 7, IROWMK(3) = 10, IROWMK(4) = 14 and IROWMK(5) = 18.

For certain other purposes it is also convenient to store the columns of data in a similar way to aid testing. Again we have three arrays ICOLNO, ICOLMK and COLELL which perform similar roles for columns as the IROWNO, IROWMK nad ROWELL performed (repsectively) for rows. However, these fit in more naturally with existing SCICONIC storage. They are set as follows:

ICOLNO(1) = 3, ICOLNO(2) = 1, ICOLNO(3) = 2, ICOLNO(4) = 4
ICOLNO(5) = 2, ICOLNO(6) = 1, ICOLNO(7) =3
ICOLNO(8) = 4, ICOLNO(9) = 1, ICOLNO(10) = 3, ICOLNO(11) = 4
ICOLNO(12) = 5

ICOLNO(13) = 3, ICOLNO(14) = 1, ICOLNO(15) -2, ICOLNO(16) = 5

ICOLNO(17) = 4, ICOLNO(18) = 1, ICOLNO(19) = 3, ICOLNO(20) = 4, ICOLNO(21) = 5.

ICOLMK(1) - 1, ICOLMK(2) = 5, ICOLMK(3) = 8, ICOLMK(4) = 13, ICOLMK(5) = 17.

COLELL(2) = 1.0, COLELL(3) = 2.0, COLELL(4) = 2.0

COLELL(6) = 1.0, COLELL(7) = 1.0,

COLELL(9) = 1.0, COLELL(10)= 1.0, COLELL(11) = 4.0, COLELL(12) = 8.0

COLELL(14) = 1.0, COLELL(15) = 1.0, COLELL(16) = 2.0

COLELL(18) = 1.0, COLELL(19) = 1.0, COLELL(20) = 1.0, COLELL(21) = 5.0.

An impoαtant point should be noticed that, when we make a deletion or any change we must update both types of stored data.

Now, we discuss how simplex operations interact with this type of storage in our programs. If we look at the row storage, we can find pivot elements etc. and start the simplex operations. There might be a problem when we update coefficients as often a zero becomes non-zero and will need to be stored. In fact this is straightforward because the trick is that IROWMK tells us where row data starts and we can move around these values. Obviously, we need a duplicate copy of IROWMK, IROWNO, ROWELL calling them JROWMK, JRWONO, ROWELJ for tableau 2.

Let row 3 be first pivot row, we set JRWOMK(3) = 1, then adjust the elements of row 3, store them in positions 2,3,4 ... and set up JROWNO, ROWELJ. Now we update another row eg. row 1, row 2, etc. We now proceed towards a feasible solution or perturbation method or whatever is required.

When performing simplex we might wish to update part of the column arrays so that we can find pivot points more easily. But the column arrays can always be created from the row arrays if necessary.

With the above way an efficient method of storage and carrying out of all tests is achieved.

Now, two important points have to be mentioned:

In programming the methods, care was taken to minimize the effects of the round-off errors on the results of some methods (eg. the simplex pivot, classifiying some redundant constraints as non-redundant). We solved the above problem by considering any number with an absolute value less than or equal to the relative zero 10^{-8} as zero.

As most of the methods required an initial basic feasible solution, and some difficulties arise in getting it due to the techniques used by SCICONIC package, we considered the linear programming problems as being re-expressed with constraints of type "$\leq$".

In order to understand the specifics such as memory space requirements and the order of operations, we now present four miscellaneous points of the programming process used for some methods.

(a) In sign tests (Extended sign tests, Hybrid, Reduce and extended Reduce and extended reduce methods), we stopped the given test before the entire row or column was scanned. For example, we stopped the process of test two as soon as a second negative entry was found. The minimum quotient to perform a simplex pivot as well as updating the tableau were written in the program. Cycling problems could occur, but our problems do not generally contain such cases. Consequently, we did not implement a check for identifying such cases. The computational effort for this process is negligible and does not affect the results reported in this chapter.

(b) In the extended reduce method we propose to stop the tests if the amount of the identification is less than 10% of the number of rows and columns during the pass (unless on the first pass).

(c) In Williams' and extended Williams procedures we utilised the lower bounds of shadow prices at zero, and at some sufficiently positive large real number for the upper bounds of the shadow prices. The bounds on the primal variables were also initialised at zero or at some sufficiently large real number if they had not been set already in the problem file.

(d) As the extended Williams procedure and preprocessing reduction procedure for integer problems implements Klein and Holms' tests in which the pairwise comparisons between rows and columns are performed, we order the cost coefficents and the right-hand side values before starting the test processing and only the values of the right-hand side are re-ordered if there is any change in their values during the preprocessing reduction procedure. While, in programming the pairwise comparisons between columns the original cost coefficients are stored in ascending order, and the updated cost coefficients are not used in this pairwise test. Also, the columns chosen in the comparisons should not have any non-zero elements corresponding to "redundant" constraints with non-zero shadow prices which are removed from the problem.

6.2 Performance of Method

Karwan et al. reported computational tests on most of the common size reduction techniques (the methods of Zionts and Wallenius, Telgen, Gal, Rubin, Boneh and Golan, Mattheiss, Holm and Klein, Williams, Thompson and Sethi as well as Lotfi's improvements, i.e. Extended Sign tests, Hybrid and Reduce methods) in a comprehensive experiment to determine the relative performance of the various tests.

As our objective study is to ascertain how successfully, size-reduction techniques could be implemented in mathematical programming packages, and to avoid any repetition of the results of the performance of the methods, we concentrated our experiments onthe methods which we extended (ie. Reduce method and Williams' procedure). These are described in detail in the tables of results later in this chapter. However, Boneh and Golan's, Holm and Klein's, Extended sign tests and Hybrid methods are discussed briefly in this chapter. The performance of these methods is also discussed in more detail in Karwan et al. (1983).

In order to evaluate the performance of the methods, items such as the relative time, number of iterations, the structure of the tested problem in hand, size, degeneracy and other factors, if known, were noted. A comparison in terms of CPU time was made to solve the tested problems with and without the reduction methods implemented.

A number of problems used were obtained from different sources and most of them have been modified after changing "$\geq$" and "=" to "$\leq$" in order to ensure the problem still has a feasible all-slack solution. The characteristics of these problems are presented in table 6.1 for testing all reduction methods except the pre-processing reduction procedure for integer problemsfor which the characteristics of the tested problems are presented in table 6.6.

<u>Characteristics of the tested problems</u>

Problem No.	Dimension Row	Column	No of non-zero elements	Starting Percent Degenerate (*)	No. of Simplex Iterations	CPU time (**) (sec)	Source
1	20	30	76	0	24	2.0	Farm Planning. Williams, N (1967)
2	27	48	169	0	21	4.0	Production Planning. Williams, N (1967)
3	17	40	191	0	5	3.1	Mixing Problem. Williams, N (1967)
4	45	37	140	40	12	3.3	Tischer, H. J (1968)
5	30	44	139	0	21	4.8	AHMED, A. N (1977)
6	35	50	136	14	25	4.8	SCICON Ltd, Company
7	46	63	217	0	26	4.8	AHMED, A. N (1977)
8	59	79	281	13	60	10.0	Brunel University. Private Communication.
9	40	94	941	0	14	6.0	Chvâtal, V (1984)
10	21	115	900	0	8	5.8	Oil Company
11	56	125	416	0	36	17.8	Brunel University. Private Comminication
12	64	133	415	0	12	6.8	London School of Economics. Private Communication

Problem No.	Dimension Row	Column	No of non-zero elements	Starting Percent Degenerate (*)	No. of Simplex Iterations	CPU time (**) (sec)	Source
13	90	137	463	0	13	7.0	Oil Company
14	100	130	380	55	20	8.8	Brunel University. Private Communication
15	100	140	471	0	25	7.3	SCICON Ltd, Company
16	140	180	890	0	27	22.0	SCICON Ltd, Company
17	180	249	830	60	108	35.0	Brunel University. Private Communication
18	200	290	1010	65	161	45.0	Brunel University. Private Communication
19	230	300	1070	27	158	57.0	Brunel University. Private Communication
Mean	78.95	120.21	477.47	14.42	40.84	13.44	

(*) The starting percent degenerate, a measure of a problems' degeneracy, is the percentage of starting "Right-hand side" vector entries that are zero.

(**) Average CPU time to get an optimal solution by a series of runs is considered to take into account variations in timing caused by the business of the Prime Computer System

6.2.1 Boneh and Golan's method

As mentioned earlier, this method attempts to identify the non-redundant constraints and labels the remaining unidentified constraints as redundant (possibly with some errors). The method, as originally suggested by Boneh, would stop after a certain number of iterations. The results show that more than 90% of the non-negativity constraints and more than 70% of the structural constraints are identified as non-redundant. The method did very well in identifying almost all the non-redundant constraints especially in terms of computation time, since it did not require any simplex pivots.

The existence of extraneous variables in the problems affects the results by classifying some redundant constraints as non-redundant. This occured because of perturbing the problem where extraneous variables could have small positive values in an interior feasible point. Also, the above results can arise when the direction from the interior feasible point to all constraints is along one of the extraneous variables.

Also, as we mentioned above most of the non-negativity constraints are labelled as non-redundant, and that tells us very little about their variables since their values may turn out to be equal to zero or not.

We believe that such a method with its design and purpose is not useful for implementation in mathematical programming packages as a size-reduction technique. Therefore, we modified this method and implemented it in our extended reduce method to identify redundant constraints instead of non-redundant constraints, which becomes more helpful.

6.2.2 Klein and Holm's method

This method attempted to identify extraneous variables and non-binding constraints by consecutive pairwise comparisons of columns and rows. As the tested problems were different from the ones in the other methods in that they all had non-negative A matrix for this method, consequently we could not solve all the tested problems presented in table 6.1, using this method.

The efficiency of this method depends on the rate of degeneracy and the number of variables with non-positive cost coefficients. First, because of the non-negativity condition on the A matrix, any variable with a negative cost coefficient is extraneous. Secondly, in the non-negative constraints with a zero right-hand side, every positive entry corresponds to an extraneous variable. These variables and constraints may be dropped immediately, and therefore lower average execution times apply. The results show that this method is not efficient in terms of size reduction rate and the computation time used, and that is due to the weakness in tightening the bounds on both primal and dual. Therefore, we believe this method is not helpful to be implemented alone as a reduction method in mathematical programming packages. We combined their tests in our improvements methods, within which they become more helpful in their reductions (see extended Williams procedure, chapter 4 and preprocessing reduction procedure for integer problems, chapter 5).

6.2.3 Extended Sign Test Method

As we mentioned earlier the extended sign test method is an improved version of the sign test (Zionts and Wallenius, Telgen, Gal and Rubin) methods. A full comparative efficiency of each test and the extended sign method is reported in Karwan et al. (1983).

The results show that test three is not performed well in both degenerate and non-degenerate problems, in terms of nuumber of identifications. Although, test four performed very well in identifying a large number of the non-negativity constraints as non-redundant, it is not helpful for reducing the problem size, as we mentioned before regarding Boneh and Golan's method. The performance of test five is efficient in terms of number of identifications.

The method identified more than 70% of the non-redundant constraints but not more than 40% of the redundant constraints in the early iterations (an iteration is a series of tests between two pivots of the simplex algorithm). The method becomes less powerful as the number of iterations increases, since the number of unsuccessful iterations (an iteration which didn't identify any constraints at its tests) increases and therefore more wasteful execution time is used.

6.2.4 Hybrid Method

This method is an improvement on the sign test methods, and consists of two parts. In the first part, one iteration is performed using the co-ordinate direction method to identify some non-redundant constraints. In the second part, the E.S.T. method is used to determine the status of the remaining constraints.

The results show that the performance of this method is better than the extended sign test method in terms of the execution times. The efficiency of the method is due to the power of the first part which identified more than

65% of the total constraints at an average execution time about 10% of the total testing time. However, in the second part of the method the number of iterations is less than the number of iterations performed in the extended sign tests method. The unsuccessful iterations and the method of identifying redundant constraints in the Hybrid method have the same characteristics as in the extended sign test method.

6.2.5 Reduce Method

The Reduce method reduces the problem size (when possible) while solving the problem. The reductions are achieved by identifying redundant as well as non-binding constraints and extraneous variables. The results of this method are presented in table 6.2

As can be seen from table 6.2, the size reduction ranges between zero (problem 9) and 99% (problem 17) and the overall size reduction is 58.21%. The times range between -14% ie. 14% more execution time used (problem 13) and 90% (problem 17) and the overall reduction is 34.53% (about 51. less than in the simplex methods). The reasons which affect the success of the reduce method are the extra execution time due to repeating the processing of the tests (steps 2 - 6) with no more identifications, the unhelpful tests (step 4 and step 6), and more unhelpful iterations (the iteration with fewer number of identification, comparing with the size of the reduced problem). Also, the number of iterations of the reduced problem is about 15% lower. Finally, the structure of the problems at hand have greatly affected the results of the reduce method.

Table 6.2

Results of the Reduce Method

oblem	Dimension Row	Column	Size (mxn) Actual	Reduced	% Size Reduction
(20x30)	17	23	600	391	35
(27x48)	17	25	1296	425	67
(17x40)	17	14	680	238	65
(45x37)	17	14	1665	238	86
(30x44)	29	25	1320	1015	23
(35x50)	24	29	1750	696	60
(46x63)	38	35	2898	1330	54
(59x79)	49	42	4661	2058	56
(40x94)	40	94	3760	3760	0
(21x115)	21	32	2415	672	72
(56x125)	20	40	7000	800	89
(64x133)	59	106	8512	6254	27
(90x137)	89	126	12330	11214	10
(100x130)	49	48	13000	2352	82
(100x140)	33	30	14000	990	93
(140x180)	123	137	25200	16851	33
(180x249)	29	16	44820	464	99
(200x290)	175	68	58000	11900	79
(230x300)	130	128	69000	16640	76
in	51.37	54.32	14363.53	4120.42	58.21

Table 6.2 (continued)

Problem	Iterations	Time (sec) Testing	Time (sec) Total	% Time Reduction
1 (20x30)	17	0.727	1.227	39
2 (27x48)	19	0.860	2.4	40
3 (17x40)	5	1.0	2.2	28
4 (45x37)	8	1.0	1.95	41
5 (30x44)	19	0.9	3.3	30
6 (35x50)	24	0.9	3.75	20
7 (46x63)	21	1.3	3.6	23
8 (59x79)	40	1.3	5.0	50
9 (40x94)	14	0.25	6.25	-6
10 (21x115)	5	1.3	4.0	32
11 (56x125)	36	2.7	10.5	41
12 (64x133)	12	1.5	6.5	4
13 (90x137)	13	1.4	8.0	-14
14 (100x130)	17	1.7	5.6	37
15 (100x140)	14	2.0	3.4	51
16 (140x180)	25	1.5	15.75	29
17 (180x249)	20	2.2	3.7	90
18 (200x290)	70	2.4	15.75	59
19 (230x300)	90	3.75	21.9	62
Mean	24.37	1.51	6.57	34.53

6.2.6 Williams' Procedure

Williams' procedure attempts to reduce the size of the problem by removing extraneous variables and non-binding constraints. Moreover, singleton rows and columns are replaced by primal and dual variable bounds, respectively. The results of Williams' procedure are summarised in table 6.3.

As can be seen the procedure reduces the size of the problems to about 49.31%. The overall average execution time reduction is 25.78%, with an average of 9.1 seconds (about 33% less than in simplex methods). The average number of iterations for all the problems is 27.0 (about 34% less than in simplex methods).

The success of Williams' procedure depends on the extent of tightening of the bounds on the dual variables and the structure of the problems, such as degeneracy (on the optimality) and redundancy. Also the number of variables which have been fixed are non-zero values (problems 4, 11 and 15) affects the number of iterations and consequently the execution time. Also it should be noted that the average reducing time is 0.75 seconds which is about 50% less than the reducing time in the Reduce method (an average of 1.5 second)

Finally, the performance of Williams' procedure could be better with problems of mixed types of constraints (ie. $\leq$, = and $\geq$) where more and better bounds are tightened on both primal and dual variables.

Table 6.3

Results of Williams' Procedure

Problem	Dimension Row	Dimension Column	Size (mxn) Actual	Size (mxn) Reduced	% Size Reduction
1 (20x30)	20	21	600	420	30
2 (27x40)	17	34	1296	578	55
3 (17x40)	6	26	680	156	77
4 (45x37)	37	22	1665	814	51
5 (30x44)	15	31	1320	465	65
6 (35x50)	22	29	1750	638	64
7 (46x63)	19	44	2898	836	71
8 (59x70)	30	62	4661	1860	60
9 (40x94)	28	94	3760	2632	30
10 (21x115)	21	115	2415	2415	0
11 (56x125) (*)	-	-	7000	-	100
12 (64x133)	64	133	8512	8512	0
13 (90x137)	89	126	12330	11214	9
14 (100x130)	20	105	13000	2100	84
15 (100x140)	26	23	14000	598	96
16 (140x180)	100	148	25200	14800	41
17 (180x249)	134	221	44820	29614	34
18 (200x290)	170	262	58000	44540	23
19 (230x300)	152	265	69000	40280	42
Mean	51.05	92.69	14363.53	8551.16	49.31

(*) Problem is solved during the reduction procedure.

Table 6.3 (continued)

Problem	Iterations	Time (sec)		% Time Reduction
		Reducing	Total	
1 (20x30)	15	0.4	1.55	23
2 (27x48)	17	0.4	2.3	43
3 (17x40)	5	0.4	2.8	10
4 (45x37)	5	0.4	2.6	20
5 (30x44)	15	0.4	3.0	38
6 (35x50)	20	0,4	3.5	25
7 (46x63)	20	0.5	3.6	22
8 (59x79)	40	0.5	5.3	47
9 (40x94)	14	0.5	6.75	-10
10 (21x115)	8	0.8	6.55	-13
11 (56x125) (*)	0	1.15	1.15	94
12 (64x133)	12	0.6	7.4	-9
13 (90x137)	13	0.7	7.2	-3
14 (100x130)	16	0.7	6.5	27
15 (100x140)	9	0.8	2.5	66
16 (140x180)	25	1.2	17.5	20
17 (180x249)	80	1.3	29.0	17
18 (200x290)	100	1.5	32.0	29
19 (230x300)	100	1.7	32.0	44
Mean	27.0	0.75	9.10	25.78

(*) Problem is solved during reduction procedure

6.2.7 Extended Reduce Method

The extended reduce method reduces the problem size (when possible) while solving the problem and this is achieved by removing redundant as well as non-binding constraints and extraneous variables. This method is an improvement on the earlier Reduce method made by not considering some unsuccessful tests and implementing a modified version of the co-ordinate direction method at certain steps if necessary to identify redundant constraints.

Table 6.4 presents the results of the extended reduce method. As can be seen from table 6.4, the overall average size reduction is 56% which is about the same as the reduce method achieved, and that is due to performing less iterations during processing than the Reduce method. The extended reduce method attempts to minimise the number of unhelpful iterations (defined in section 6.2.4) by terminating the processing tests after one unhelpful iteration. Step six (modified co-ordinate direction method) is helpful in identifying more redundant constraints (if possible) at earlier iterations than in the Reduce method. Also, this step depends on the structure of the problem, since such redundant constraints exist only when the pivot ratio is not unique (problems 13, 18 and 19). Removing such redundant constraints at early iterations could lead us to identify more extraneous variables (problem 19) earlier than in the Reduce method.

An important consequence of the extraneous variables and non-binding constraints is the decrease in the number of simplex iterations. This may be explained by comparing the results of the extended reduce method with those of the simplex method (table 6.1). As can be seen from these tables, in the problems with lower reductions (problems 1 and 9), the numbers of iterations are the same or only slightly different. On the other hand, in problems with higher reductions (problems 17, 18 and 19) large differences are found in the number of iterations between the extended reduce method and the simplex method. However, the number of iterations overal for the problem is about 50% (averaging 23.90) less than that of the simplex methods (averaging 40.84). The reason that the extended reduce method has fewer iterations is the elimination of more extraneous variables.

Minimising the number of unhelpful iterations during the tests may avoid extra wasteful execution time by not repeating the tests for more than one pass at each iteration, and not considering steps 4 and 6 of the Reduce method in our extended reduce method. Also step 6 is successful (modified co-ordinate direction method) in identifying redundant constraints (if they exist) and achieving more eliminations of extraneous variables, with consequently smaller numbers of iterations to be performed. The total execution times to solve overall the problems has been reduced by 44.42%. The overall average reducing processing time is 0.52 seconds (about 67% less than in reduce method). The overall average total execution time is 5.74 seconds (about 13% less than in the reduce method and 57% less than in the simplex methods.

Table 6.4

Results of the Extended Reduce Method

Problem	Dimension Row	Dimension Column	Size (mxn) Actual	Size (mxn) Reduced	% Size Reduction
1 (20x30)	17	23	600	391	35
2 (27x48)	17	25	1296	425	67
3 (17x40)	17	14	680	238	65
4 (45x37)	44	17	1665	748	55
5 (30x44)	29	35	1320	1015	23
6 (35x50)	25	29	1750	725	59
7 (46x63)	44	40	2898	1760	39
8 (59x79)	49	43	4661	2107	55
9 (40x94)	40	94	3760	3760	0
10 (21x115)	21	32	2415	672	72
11 (56x125)	53	50	7000	2650	62
12 (64x133)	60	90	8512	5400	37
13 (90x137)	69	126	12330	8964	29
14 (100x130)	51	48	13000	2448	81
15 (100x140)	33	33	14000	1089	92
16 (140x180)	123	137	25200	16851	33
17 (180x249)	29	16	44820	464	99
18 (200x290)	120	68	58000	8160	86
19 (230x300)	111	131	69000	14541	79
Mean	50.11	55.32	14363.53	3782.53	56.21

Table 6.4 continued

Problem	Iterations	Time (sec) Testing	Time (sec) Total	% Time Reduction
1 (20x30)	17	0.3	1.1	45
2 (27x48)	19	0.4	1.15	71
3 (17x40)	5	0.4	1.8	42
4 (45x37)	9	0.15	1.8	46
5 (30x44)	19	0.25	2.45	50
6 (35x50)	24	0.3	3.2	34
7 (46x63)	22	0.4	3.0	38
8 (59x79)	40	0.35	4.5	55
9 (40x94)	14	0.15	6.15	-2
10 (21x115)	5	0.35	3.4	42
11 (56x125)	36	0.45	9.0	49
12 (64x133)	12	0.45	6.0	12
13 (90x137)	13	0.5	7.4	-5
14 (100x130)	17	0.5	5.0	44
15 (100x140)	14	1.0	2.7	63
16 (140x180)	25	0.75	15.5	30
17 (180x249)	16	1.0	3.0	92
18 (200x290)	65	1.15	14.0	69
19 (230x300)	82	1.1	18.0	69
Mean	23.90	0.52	5.74	44.42

6.2.8 Extended Williams Procedure

As we mentioned before this procedure is a new version of Williams' procedure by combining the test of Klein and Holm (1975) to identify extraneous variables.

The results of this procedure are presented in table 6.5. As can be seen from table 6.5, the overall average size reduction is 74.47% which is about 25% more than Williams' procedure reduced. Specifically, as can be seen from table 6.3, Williams' procedure had 0% size reduction on problems 10 and 12. On the other hand, the extended Williams' procedure reduced the size problems 10 and 12 by 72% and 37% respectively. The results from these two problems explain many reasons such as the difference in size reductions between the two procedures. Williams' procedure fails to tighten any bounds on the dual variables and only bounds on the primal variables have been tightened, with fewer redundant constraints being removed. While the extended procedure (on these problems 10 and 12) identified more extraneous variables and more "redundant" constraints have been removed consequently, some bounds on the dual variables have been tightened in the successive passes, giving the whole procedure more strength in fixing more variables.

To discuss the performance of extended Williams' procedure in terms of the execution time, Table 6.5 shows that the overall average execution time reduction is 54% (about 28% more than Williams'procedure). The average number of iterations over all the problems is 18.27 (about 8.33% less than in Williams' procedure). The average of the total execution is 5.5 seconds (about 40% less than in Williams' procedure and 60% less than in simplex methods).

The success of the extended Williams procedure over Williams' procedure, as the results show is due to the size reduction, the number of iterations and the amount of the execution time used in reducing the problems. It is quite clear that more size reduction achieved may result in less execution time to solve the reduced problems (problems 4 and 11 have been reduced and solved during the procedure). However, the number of iterations is affected by the number of variables (extraneous and non-extraneous) which have been removed from the problems (problems 5 and 16). Consequently, such effects on the number of iterations will lower the execution time to solve the reduced problems. However, the amount of execution time used in reducing the problems is not affected by the computation times used in the pairwise comparisons between columns. The average amount of such execution times by the extended Williams'procedure is 6% and 5% by Williams' procedure of the average amount of the execution time by the simplex methods, and that is due to programming and designing such pairwise comparisons in a way to avoid wasted execution time. Also, the phase is terminated after one unsuccessful pass, and part two is not to be performed if neither any singleton columns nor "redundant" constraints with non-zero shadow prices have been removed. Finally, the structure of the problems may affect both Williams' and extended Williams' procedure.

Table 6.5

Results of the Extended Williams Procedure

Problem	Dimension Row	Column	Size (mxn) Actual	Reduced	% Size Reduction
1 (20x30)	17	21	600	420	30
2 (27x48)	17	20	1296	340	74
3 (17x40)	6	23	680	138	80
4 (45x37) (*)	-	-	1665	-	100
5 (30x44)	13	12	1320	156	88
6 (35x50)	21	5	1750	105	94
7 (46x63)	19	21	2898	399	86
8 (59x79)	24	8	4661	192	96
9 (40x94)	28	80	3760	2240	40
10 (21x115)	21	32	2415	672	72
11 (56x125) (*)	-	-	7000	-	100
12 (64x133)	60	190	8512	5400	37
13 (90x137)	89	126	12330	11214	9
14 (100x130)	20	65	13000	900	93
15 (100x140)	23	13	14000	299	98
16 (140x180)	84	35	25200	2940	88
17 (180x249)	133	44	44820	5852	87
18 (200x290)	118	160	58000	18880	67
19 (230x300)	126	130	69000	16380	76
Mean	48.37	54.15	14363.95	3591.95	74.47

(*) Problem is solved during reduction procedure

Table 6.5 (continued)

Problem	Iterations	Time (sec) Reducing	Total	% Time Reduction
1 (20x30)	15	0.45	1.6	20
2 (27x48)	17	0.50	2.1	50
3 (17x40)	5	0.55	2.2	30
4 (45x37) (*)	0	0.65	0.65	80
5 (30x44)	11	0.55	2.0	59
6 (35x50)	5	0.55	1.5	69
7 (46x63)	13	0.60	1.55	68
8 (59x79)	8	0.65	1.55	85
9 (40x94)	14	0.70	5.75	5
10 (21x115)	5	0.75	2.5	57
11 (56x125) (*)	0	1.0	1.0	95
12 (64x133)	12	0.75	6.0	12
13 (90x137)	13	0.70	7.3	-2
14 (100x130)	17	0.75	5.5	38
15 (100x140)	8	0.85	1.8	76
16 (140x180)	9	1.3	2.8	88
17 (180x249)	44	1.4	10.0	72
18 (200x290)	65	1.65	24.0	47
19 (230x300)	90	1.85	24.0	58
Mean	18.27	0.85	5.5	54.0

(*) Problem is solved during the reduction procedure

6.2.9 Preprocessing Reduction Procedure for Integer Problems

This procedure reduces the size of integer problems (when possible) by tightening the bounds on primal variables and constructing new formulae to use only the primal bounds to fix the variables at their bounds. Extraneous variables and redundant constraints as well as non-binding constraints are removed, where the test of Klein and Holm (condition 3.2.1.6) is used to identify non-binding constraints. This reduction procedure is implemented prior to solving the integer problems by the established techniques.

The results of this reduction procedure are summarised in table 6.7. As can be seen from this table, the overall average size reduction is 65.67% and the overall average execution time is 50%. The performance of the procedure in terms of the size is dependent on the structure of the problems, where tighter bounds on the primal variables required by the formulae (5.1.2 - 5.1.3) to fix integer variables at their bounds, and condition (3.2.1.6) to identify non-binding constraints. However, the amount of size reduction is affected by the performance of the reduction procedure in terms of the execution times. The numbers of branches and iterations have much effect on the total execution times. The overall average of the total execution times is 18.86 seconds (about 55% less than by the simplex methods and Branch-and-Bound algorithms). Also, as can be seen from the table 6.7, in problem 5, 62% of its size has been reduced, while 30% of its former execution time has been reduced, and that is due to no change in the number of branches and iterations. Also, the effectiveness of the number of branches and iterations may be seen from problem 9, where 41% of its size has been reduced and 76% of its former execution time has been reduced and that is due to the changes in the number of branches (about 78% less) and in the number of iterations (about 74% less). Therefore, the reduction process will result in problems which require fewer branches and iterations and consequently much less execution time.

Table 6.6

Characteristics of Tested Integer Problems*

Problem No.	Dimension Row	Column	No. of non-zero Elements	No. of Iterations	No. of Branches	CPU time (sec)
1	9	19	78	17	43	7.5
2	15	11	88	34	37	7.0
3	13	20	70	30	13	4.2
4	19	20	87	11	3	2.20
5	20	25	61	8	1	2.0
6	27	28	96	7	5	2.9
7	20	44	139	30	39	7.5
8	29	63	217	125	217	42.0
9	56	80	320	291	483	123.5
19	89	137	463	136	210	188.0
11	109	160	519	114	99	70.53
12	140	180	582	64	60	50.45
Mean	45.5	65.58	226.67	71.42	100.83	43.32

* These problems are modified versions of the problems in Table 6.1

Table 6.7

Results of Preprocessing Reduction Procedure

Problem	Dimension Row	Dimension Column	Size (mxn) Actual	Size (mxn) Reduced	% Size Reduction
1 (9x19)	6	2	171	12	93
2 (15x11)	3	3	165	9	95
3 (13x20)	11	8	260	88	66
4 (10x20)	11	7	380	77	80
5 (20x25)	10	19	500	190	62
6 (27x28)	10	16	756	160	79
7 (20x44)	17	25	880	425	52
8 (29x63)	11	20	1827	220	88
9 (56x80)	53	50	4480	2650	41
10 (89x137)	88	109	12193	9592	21
11 (109x160)	62	136	17440	8432	52
12 (140x180)	110	140	25200	15400	59
Mean	32.67	44.58	5354.33	3104.58	65.67

Table 6.7 (continued)

Problem	No of Iterations	No of Branches	Time (sec) Reducing	Total	% Time Reduction
1 (9x19)	3	5	0.5	3.0	60
2 (15x11)	9	9	0.7	3.0	57
3 (13x20)	8	2	0.8	1.65	61
4 (19x20	4	1	0.3	1.36	38
5 (20x25)	8	1	0.35	1.4	30
6 (27x28)	7	3	0.4	1.95	33
7 (20x44)	16	9	0.5	4.0	47
8 (29x63)	64	87	0.80	18.0	57
9 (56x80)	76	108	1.35	29.0	76
10 (89x137)	148	175	1.66	100	47
11 (109x160)	43	62	1.77	30.0	57
12 (140x180)	48	46	2.0	32.0	37
Mean	30.17	42.33	0.93	18.86	50.0

CHAPTER VII

Conclusions and Recommendations For Further Research

The principle objective of the research reported in this thesis was to ascertain how successfully, size-reduction techniques could be implemented in mathematical programming packages. To achieve this goal, we selected the most promising size-reduction techniques studied them and tested some of them on some Linear programming problems with different characteristics,obtained from different sources. Consequently, we were able to determine the performance of these techniques.

The test process enabled us to determine the most efficient size-reduction techniques. During this process we determined some modifications for extensions and improvements to these techniques.

The test process enabled us to determine the most efficient size-reduction techniques. During this process we determined some modifications for extensions and improvements to these techniques. The details of our extensions were presented in Chapters IV and V. We then tested these methods and compared their results with the earlier ones. The results and the discussion on all techniques are presented in Chapter VI.

Now we present a summary of the conclusions made for the various techniques. Also we discuss possible changes for future improvements and extensions.

7.1 Summary and Conclusions

Although Boneh and Golan's method did very well in terms of computation time, their results indicated some error in the identifications.

Holm and Klein's method required problems with non-negative constraint coefficients and right-hand side sectors. The results show that, this method is not so efficient in terms of size-reduction rates and computation times.

The extended sign tests and Hybrid methods performed equivalently, but their results are not useful for our objective study.

The results of the Reduce method indicated that the success of this method over the simplex depends on the structure of the problem.

However, the results of the extended reduce method are slightly different from the Reduce method in terms of size reduction. The extended reduce method is more successful over the Reduce method in terms of computation times. Moreover, it was indicated that on the average, both methods have a faster convergence rate than the simplex method.

However, the results of Williams' and the extended Williams procedures indicated that tightening of better bounds on primal and dual variables depends on the structure of the problems, and affects the performance of reductions. The extended Williams procedure showed consistent superiority over the Williams' procedure in terms of size and time reductions.

The improvement called preprocessing reduction procedure for integer problems attempted to reduce the size of integer problems using only the primal bounds, prior to solving the problems by the established techniques. The results indicate a reasonable success over the simplex and Branch-and-Bound techniques.

From the proceeding a general conclusion may be reached that implementing such reduction techniques in mathematical programming packages could be desirable with large size problems rather than small problems from the economical view.

7.2 Recommendtions for Future Research

In the previous section, we presented the conclusions of some of the size-reduction techniques studied in this thesis. In this section we present some ideas which may result in further extensions and improvements to the existing methods. We restrict our discussions to those methods which appear most useful in our objective study.

The Reduce and the Extended reduce method may be utilized in a number of different ways. Among the most promising approaches is one in which a certain number of tests are no longer employed when their efficiency falls below a specified level. Of course, the level at which the test is discontinued must be determined empirically.

Another approach is to use these two methods for partial classification. This may be achieved by terminating the methods after a certain number of iterations. The number of iterations at which the processing stops is a function of the problem size and should be determined through further investigations.

Also, another extension to these two methods consists of obtaining the maximum possible reduction for a given problem. In that case, the Reduce and the Extended reduce methods are used in a fashion similar to that of the Extended sign test method. Namely, we attempt to minimize the slack variable associated with each constraint. However, we include the tests which identify the extraneous variables and update the objective function at each iteration as well.

As in Thompson and Sethi's method the candidate constraints were those which contained a pivot element in columns with potential variables for entering into the basis.

These constraints were updated at each iteration. The remaining constraints, called non-candidate, were not updated with the hope that they would never become violated. In fact, we may implement the tests which are used in Extended-Reduce method to identify redundant constraints on the set of the non-candidate constraints only.

Now, we discuss the possible improvements to Holm and Klein's method, Williams and Extended Williams procedures.

Holm and Klein's method was restricted to the specially-structured problems due to the lack of bounds on variables in the other problems (those with a general A matrix). However these bounds may be obtained in a fashion similar to that of Williams' procedure. One may utilize the complementary slackness theorem to obtain better bounds on all of the variables. That is, the optimal objective function value may be written as

$$CX^* = W^*b$$

where X^* and W^* are the values of the primal and dual variables at optimality. Using the above relationship in conjunction with bounds on some variables we may obtain bounds on the other variables. The above equality may be written as an inequality in either direction (i.e., $\leq$, $\geq$) depending on the existing bounds and the desired new bounds.

The above utilization may be implemented to improve the bounds in Williams' and Extended Williams procedures.

Finally, another extension to Williams' procedure and Holm and Klein's method is to combine the methods with each other and utilize the above procedure for obtaining better bounds as well.

In that case, after the bounds have been tightened Holm and Klein's method may be used to remove some extraneous variables and nonbinding constraints. Then, Williams' procedure is applied to the remaining constraints and variables to reduce the problem further.

APPENDIX A

In this Appendix, some details of the necessary arrays used in the Sciconic Algorithmic Tools Library (SATL) and the specification of the commands to run the package are presented.

SCICONIC/VM was designed to be implemented in a highly modular fashion, so that extensions and enhancements could be easily incorporated. In order to help the user to be able to create FORTRAN routines of his own employing the primitives of the SCICONIC/VM SATL, the user must have an understanding of the design concepts behind SCICONIC/VM, in particular those behind SCICONIC/VM's algorithmic routines.

The variables used by SCICONIC/VM may be accessed via their associated ACCESS KEYS. The inclusion statement takes the form:

⟨ include keyword⟩ ⟨file name specification⟩

where ⟨include keyword⟩ is $INSERT (in Prime Computer System),

⟨filename specification⟩ may well be filename. In almost all cases, the filename for an entity with access key AAAAAA will be of the form PDPAAAAAA.

An example, suppose the array PARAMS is required in a routine. Then the statement

```
$INSERT SCICON > S > PDPPARAMS
```

should appear in the Source Code.

To describe the data structure created in core ready for an algorithmic routine to access, first, some preliminary sizing definitions are given:

NROW - The number of rows in the in-core matrix (including the objective function row which is row KPTOBJ)

NSEQ - The total number of vectors in the in-core matrix (i.e. slacks, structural vectors and any range vectors (q.v.) created).

Now, we describe some of the main necessary Arrays used in the SATL for the access of matrix elements:

NAME	TYPE	ACCESS KEY	USE
POOL	real*8	POOL	Pool of unique element values
BETA	"	BETA	Right-hand sides
MRKEY	integer*2	MRKEY	Key information of variable basic in this row.
MCKEY	integer*2	MCKEY	Column key information.
MRWME	integer*2	MATRIX)	Parallel arrays, MRWME contai
MPTME	"	")	row number whose element in POOL is indexed by MPTME.
MSMEL	integer*4	"	Start of column information in MRWME/MPTME.
MSKMEB	integer*2	"	Skip value: 0 for rows 1 if no UB/Cost 2 if UB and/or Cos

The input for the simplest SCICONIC run can be considered as being made up of two parts:

1. Input Data: This contains the actual problem to be solved in coded form. The data of the LP problem has to be input from the matrix fo coefficients.

The data must be input to a file created by the editor and then the file created is used by SCICONIC. In fact, we shall not discuss the details of the input data in this Appendix.

2. Control Commands: Within this part commands required to run the package are made. Assuming we have a file of data and we wish to run the LP problem. We start by accessing the package. We type

SCICONIC

we get a prompt of (these prompts continue throughout the run)

1|>

we type INFILE = 'MYDATA'

(MYDATA is the file in the UFD to which we are attached, quotes are mandatory)

and it prompts

2|>

and we type CONVERT

(this command will load the input data from the data-file on the problem file and it will focus on possible data errors), and it replies with information and them prompts

3|>

we type SETUP (MAXIMISE/MINIMISE)

(this command will load the problem into core from the problem file)

and it replies with information and then prompts

4|>

we type PRIMAL

(it will try to solve the problem, printing out some information such as number of iterations.... etc) and then prompt

5I)

we type PRINTSOLN

(it will print out details of the solution). When complete we received the prompt

6I)

we conclude the Session with STOP

It replies ****STOP then OK.

To run an integer program, basically the same procedures are used as for LP. The main exceptions are:-

(i) In the input data, each variable must be declared as integer and specified under the bounds section.

(ii) In the program commands, the PRIMAL is followed by the command GLOBAL. This performs the Branch-and-Bound algorithm until a solution is reached (or the problem is declared infeasible). Subsequent solutions are found by repeating the GLOBAL command.

Now, if we wish to execute the 'SUBROUTINE USER' which the tests have been built into, we type USER after the problem has been loaded into core by SETUP, and before we type PRIMAL or GLOBAL.

All the above commands will be shown by solving the problem in Appendix B.

APPENDIX B

In this Appendix, one tested problem is selected. Its original data and computer results to get an optimal solution with and without reducing the problem by Extended Williams procedure, are presented. Then the program listings of the three main extended methods (Extended Reduce method, Extended Williams procedure and Preprocessing Reduction procedure) respectively, are presented.

All computation work was carried out on the PRIME 400 Computer System at Loughborough University of Technology.

ME QA4RT32
NS
L R0001
L R0002
L R0003
L R0004
L R0005
L R0006
L R0007
L R0008
L R0009
L R0010
L R0011
L R0012
L R0013
L R0014
L R0015
L R0016
L R0017
L R0018
L R0019
L R0020
L R0021
L R0022
L R0023
L R0024
L R0025
L R0026
L R0027
L R0028
L R0029
L R0030
L R0031
L R0032
L R0033
L R0034
L R0035
L R0036
L R0037
L R0038
L R0039
L R0040
L R0041
L R0042
L R0043
L R0044
L R0045
L R0046
L R0047
L R0048
L R0049
L R0050
L R0051
L R0052
L R0053
L R0054
L R0055
L R0056
N OBJ

JMNS

C0001	R0001	1,000000
C0001	R0002	1,000000
C0001	R0006	1,000000
C0001	R0007	1,000000
C0001	R0048	1,000000
C0001	R0054	0,880000
C0001	R0056	0,880000
C0002	R0003	1,000000
C0002	R0004	1,000000
C0002	R0005	1,000000
C0002	R0007	1,000000
C0002	R0008	1,000000
C0002	R0049	1,000000
C0002	R0054	0,926667
C0002	R0056	0,926667
C0003	R0003	1,000000
C0003	R0004	1,000000
C0003	R0005	1,000000
C0003	R0018	1,000000
C0003	R0019	1,000000
C0003	R0049	1,000000
C0003	R0054	0,948889
C0003	R0056	0,948889
C0004	R0003	1,000000
C0004	R0004	1,000000
C0004	R0005	1,000000
C0004	R0020	1,000000
C0004	R0021	1,000000
C0004	R0022	1,000000
C0004	R0049	1,000000
C0004	R0054	1,000000
C0004	R0056	1,000000
C0005	R0003	1,000000
C0005	R0004	1,000000
C0005	R0005	1,000000
C0005	R0021	1,000000
C0005	R0022	1,000000
C0005	R0049	1,000000
C0005	R0054	0,948889
C0005	R0056	0,948889
C0006	R0007	1,000000
C0006	R0008	1,000000
C0006	R0013	1,000000
C0006	R0014	1,000000
C0006	R0015	1,000000
C0006	R0049	1,000000
C0006	R0054	0,971111
C0006	R0056	0,971111
C0007	R0013	1,000000
C0007	R0014	1,000000
C0007	R0015	1,000000
C0007	R0020	1,000000
C0007	R0021	1,000000
C0007	R0022	1,000000
C0007	R0049	1,000000
C0007	R0054	1,044444
C0007	R0056	1,044444
C0008	R0013	1,000000
C0008	R0014	1,000000

C0008	R0015	1,000000
C0008	R0021	1,000000
C0008	R0022	1,000000
C0008	R0049	1,000000
C0008	R0054	0,993333
C0008	R0056	0,993333
C0009	R0016	1,000000
C0009	R0017	1,000000
C0009	R0031	1,000000
C0009	R0032	1,000000
C0009	R0051	1,000000
C0009	R0055	0,906667
C0010	R0008	1,000000
C0010	R0010	1,000000
C0010	R0011	1,000000
C0010	R0012	1,000000
C0010	R0052	1,000000
C0010	R0056	0,860000
C0011	R0008	1,000000
C0011	R0036	1,000000
C0011	R0037	1,000000
C0011	R0038	1,000000
C0011	R0052	1,000000
C0011	R0056	0,824444
C0012	R0008	1,000000
C0012	R0009	1,000000
C0012	R0011	1,000000
C0012	R0012	1,000000
C0012	R0052	1,000000
C0012	R0056	0,837778
C0013	R0008	1,000000
C0013	R0009	1,000000
C0013	R0037	1,000000
C0013	R0038	1,000000
C0013	R0052	1,000000
C0013	R0056	0,824444
C0014	R0008	1,000000
C0014	R0009	1,000000
C0014	R0039	1,000000
C0014	R0040	1,000000
C0014	R0041	1,000000
C0014	R0052	1,000000
C0014	R0056	0,891111
C0015	R0008	1,000000
C0015	R0009	1,000000
C0015	R0040	1,000000
C0015	R0041	1,000000
C0015	R0052	1,000000
C0015	R0056	0,866667
C0016	R0010	1,000000
C0016	R0011	1,000000
C0016	R0026	1,000000
C0016	R0027	1,000000
C0016	R0028	1,000000
C0016	R0052	1,000000
C0016	R0056	0,915556
C0017	R0010	1,000000
C0017	R0011	1,000000
C0017	R0012	1,000000
C0017	R0026	1,000000

C0017	R0027	1,000000
C0017	R0028	1,000000
C0017	R0052	1,000000
C0017	R0056	1,044444
C0018	R0024	1,000000
C0018	R0025	1,000000
C0018	R0026	1,000000
C0018	R0027	1,000000
C0018	R0028	1,000000
C0018	R0052	1,000000
C0018	R0056	0,882222
C0019	R0026	1,000000
C0019	R0027	1,000000
C0019	R0028	1,000000
C0019	R0030	1,000000
C0019	R0031	1,000000
C0019	R0052	1,000000
C0019	R0056	0,920000
C0020	R0026	1,000000
C0020	R0027	1,000000
C0020	R0028	1,000000
C0020	R0037	1,000000
C0020	R0038	1,000000
C0020	R0052	1,000000
C0020	R0056	0,831111
C0021	R0026	1,000000
C0021	R0027	1,000000
C0021	R0028	1,000000
C0021	R0039	1,000000
C0021	R0040	1,000000
C0021	R0041	1,000000
C0021	R0052	1,000000
C0021	R0056	0,897778
C0022	R0023	1,000000
C0022	R0024	1,000000
C0022	R0025	1,000000
C0022	R0033	1,000000
C0022	R0034	1,000000
C0022	R0053	1,000000
C0022	R0056	1,064444
C0023	R0029	1,000000
C0023	R0030	1,000000
C0023	R0032	1,000000
C0023	R0033	1,000000
C0023	R0034	1,000000
C0023	R0053	1,000000
C0023	R0056	1,055556
C0024	R0029	1,000000
C0024	R0030	1,000000
C0024	R0042	1,000000
C0024	R0043	1,000000
C0024	R0044	1,000000
C0024	R0053	1,000000
C0024	R0056	1,000000
C0025	R0029	1,000000
C0025	R0030	1,000000
C0025	R0031	1,000000
C0025	R0043	1,000000
C0025	R0044	1,000000
C0025	R0053	1,000000

C0025	R0056	1,011111
C0026	R0029	1,000000
C0026	R0030	1,000000
C0026	R0031	1,000000
C0026	R0046	1,000000
C0026	R0047	1,000000
C0026	R0053	1,000000
C0026	R0056	1,035556
C0027	R0032	1,000000
C0027	R0033	1,000000
C0027	R0034	1,000000
C0027	R0035	1,000000
C0027	R0036	1,000000
C0027	R0037	1,000000
C0027	R0053	1,000000
C0027	R0056	1,000000
C0028	R0035	1,000000
C0028	R0036	1,000000
C0028	R0037	1,000000
C0028	R0042	1,000000
C0028	R0043	1,000000
C0028	R0044	1,000000
C0028	R0053	1,000000
C0028	R0056	0,944444
C0029	R0035	1,000000
C0029	R0036	1,000000
C0029	R0037	1,000000
C0029	R0045	1,000000
C0029	R0046	1,000000
C0029	R0047	1,000000
C0029	R0053	1,000000
C0029	R0056	0,940000
C0030	R0033	1,000000
C0030	R0034	1,000000
C0030	R0035	1,000000
C0030	R0036	1,000000
C0030	R0037	1,000000
C0030	R0038	1,000000
C0030	R0053	1,000000
C0030	R0056	1,020000
C0031	R0035	1,000000
C0031	R0036	1,000000
C0031	R0037	1,000000
C0031	R0038	1,000000
C0031	R0045	1,000000
C0031	R0046	1,000000
C0031	R0047	1,000000
C0031	R0053	1,000000
C0031	R0056	1,095556
U0001	R0001	1,000000
U0001	OBJ	-1,827095
U0002	R0002	1,000000
U0002	OBJ	1,483520
U0003	R0003	1,000000
U0003	OBJ	-1,196927
U0004	R0004	1,000000
U0004	OBJ	1,312849
U0005	R0005	1,000000
U0005	OBJ	1,312849
U0006	R0006	1,000000

U0006	OBJ	1,115922
U0007	R0007	1,000000
U0007	OBJ	1,050279
U0008	R0008	1,000000
U0008	OBJ	1,050279
U0009	R0009	1,000000
U0009	OBJ	1,050279
U0010	R0010	1,000000
U0010	OBJ	1,181564
U0011	R0011	1,000000
U0011	OBJ	1,247207
U0012	R0012	1,000000
U0012	OBJ	1,050279
U0013	R0013	1,000000
U0013	OBJ	-1,853352
U0014	R0014	1,000000
U0014	OBJ	1,115922
U0015	R0015	1,000000
U0015	OBJ	1,115922
U0016	R0016	1,000000
U0016	OBJ	1,247207
U0017	R0017	1,000000
U0017	OBJ	1,181564
U0018	R0018	1,000000
U0018	OBJ	1,050279
U0019	R0019	1,000000
U0019	OBJ	1,181564
U0020	R0020	1,000000
U0020	OBJ	-1,131285
U0021	R0021	1,000000
U0021	OBJ	1,115922
U0022	R0022	1,000000
U0022	OBJ	1,115922
U0023	R0023	1,000000
U0023	OBJ	1,050279
U0024	R0024	1,000000
U0024	OBJ	1,115922
U0025	R0025	1,000000
U0025	OBJ	1,115922
U0026	R0026	1,000000
U0026	OBJ	-1,262570
U0027	R0027	1,000000
U0027	OBJ	1,115922
U0028	R0028	1,000000
U0028	OBJ	1,050279
U0029	R0029	1,000000
U0029	OBJ	1,260335
U0030	R0030	1,000000
U0030	OBJ	1,168436
U0031	R0031	1,000000
U0031	OBJ	1,286592
U0032	R0032	1,000000
U0032	OBJ	-1,800838
U0033	R0033	1,000000
U0033	OBJ	1,050279
U0034	R0034	1,000000
U0034	OBJ	1,404749
U0035	R0035	1,000000
U0035	OBJ	-1,262570
U0036	R0036	1,000000

U0036	OBJ	1,050279
U0037	R0037	1,000000
U0037	OBJ	1,076536
U0038	R0038	1,000000
U0038	OBJ	-1,853352
U0039	R0039	1,000000
U0039	OBJ	-1,078771
U0040	R0040	1,000000
U0040	OBJ	1,129050
U0041	R0041	1,000000
U0041	OBJ	1,050279
U0042	R0042	1,000000
U0042	OBJ	1,220950
U0043	R0043	1,000000
U0043	OBJ	1,050279
U0044	R0044	1,000000
U0044	OBJ	-1,656425
U0045	R0045	1,000000
U0045	OBJ	1,050279
U0046	R0046	1,000000
U0046	OBJ	1,050279
U0047	R0047	1,000000
U0047	OBJ	-1,800838
O0001	R0001	2,500000
O0001	OBJ	-1,827095
O0002	R0002	2,500000
O0002	OBJ	1,483520
O0003	R0003	2,500000
O0003	OBJ	-1,196927
O0004	R0004	2,500000
O0004	OBJ	1,312849
O0005	R0005	2,500000
O0005	OBJ	1,312849
O0006	R0006	2,500000
O0006	OBJ	1,115922
O0007	R0007	2,500000
O0007	OBJ	1,050279
O0008	R0008	2,500000
O0008	OBJ	1,050279
O0009	R0009	2,500000
O0009	OBJ	1,050279
O0010	R0010	2,500000
O0010	OBJ	1,181564
O0011	R0011	2,500000
O0011	OBJ	1,247207
O0012	R0012	2,500000
O0012	OBJ	1,050279
O0013	R0013	2,500000
O0013	OBJ	-1,853352
O0014	R0014	2,500000
O0014	OBJ	1,115922
O0015	R0015	2,500000
O0015	OBJ	1,115922
O0016	R0016	2,500000
O0016	OBJ	1,247207
O0017	R0017	2,500000
O0017	OBJ	1,181564
O0018	R0018	2,500000
O0018	OBJ	1,050279
O0019	R0019	2,500000

00019	OBJ	1,181564
00020	R0020	2,500000
00020	OBJ	-1,131285
00021	R0021	2,500000
00021	OBJ	1,115922
00022	R0022	2,500000
00022	OBJ	1,115922
00023	R0023	2,500000
00023	OBJ	1,050279
00024	R0024	2,500000
00024	OBJ	1,115922
00025	R0025	2,500000
00025	OBJ	1,115922
00026	R0026	2,500000
00026	OBJ	-1,262570
00027	R0027	2,500000
00027	OBJ	1,115922
00028	R0028	2,500000
00028	OBJ	1,050279
00029	R0029	2,500000
00029	OBJ	1,260335
00030	R0030	2,500000
00030	OBJ	1,168436
00031	R0031	2,500000
00031	OBJ	1,286592
00032	R0032	2,500000
00032	OBJ	-1,800838
00033	R0033	2,500000
00033	OBJ	1,050279
00034	R0034	2,500000
00034	OBJ	1,404749
00035	R0035	2,500000
00035	OBJ	-1,262570
00036	R0036	2,500000
00036	OBJ	1,050279
00037	R0037	2,500000
00037	OBJ	1,076536
00038	R0038	2,500000
00038	OBJ	-1,853352
00039	R0039	2,500000
00039	OBJ	-1,078771
00040	R0040	2,500000
00040	OBJ	1,129050
00041	R0041	2,500000
00041	OBJ	1,050279
00042	R0042	2,500000
00042	OBJ	1,220950
00043	R0043	2,500000
00043	OBJ	1,050279
00044	R0044	2,500000
00044	OBJ	-1,656425
00045	R0045	2,500000
00045	OBJ	1,050279
00046	R0046	2,500000
00046	OBJ	1,050279
00047	R0047	2,500000
00047	OBJ	-1,800838
RHS	R0001	1,000000
RHS	R0002	1,000000

```
 RHS       R0003        1,000000
 RHS       R0004        1,000000
 RHS       R0005        1,000000
 RHS       R0006        1,000000
 RHS       R0007        1,000000
 RHS       R0008        1,000000
 RHS       R0009        1,000000
 RHS       R0010        1,000000
 RHS       R0011        1,000000
 RHS       R0012        1,000000
 RHS       R0013        1,000000
 RHS       R0014        1,000000
 RHS       R0015        1,000000
 RHS       R0016        1,000000
 RHS       R0017        1,000000
 RHS       R0018        1,000000
 RHS       R0019        1,000000
 RHS       R0020        1,000000
 RHS       R0021        1,000000
 RHS       R0022        1,000000
 RHS       R0023        1,000000
 RHS       R0024        1,000000
 RHS       R0025        1,000000
 RHS       R0026        1,000000
 RHS       R0027        1,000000
 RHS       R0028        1,000000
 RHS       R0029        1,000000
 RHS       R0030        1,000000
 RHS       R0031        1,000000
 RHS       R0032        1,000000
 RHS       R0033        1,000000
 RHS       R0034        1,000000
 RHS       R0035        1,000000
 RHS       R0036        1,000000
 RHS       R0037        1,000000
 RHS       R0038        1,000000
 RHS       R0039        1,000000
 RHS       R0040        1,000000
 RHS       R0041        1,000000
 RHS       R0042        1,000000
 RHS       R0043        1,000000
 RHS       R0044        1,000000
 RHS       R0045        1,000000
 RHS       R0046        1,000000
 RHS       R0047        1,000000
 RHS       R0048        1,000000
 RHS       R0049        2,000000
 RHS       R0051        1,000000
 RHS       R0052        2,000000
 RHS       R0053        3,000000
 RHS       R0054        3,000000
 RHS       R0055        1,000000
 RHS       R0056        8,000000
JNDS
/ BNDVAL   C0001        1,000000
/ BNDVAL   C0002        1,000000
/ BNDVAL   C0003        1,000000
/ BNDVAL   C0004        1,000000
/ BNDVAL   C0005        1,000000
/ BNDVAL   C0006        1,000000
```

```
/ BNDVAL    C0007          1,000000
/ BNDVAL    C0008          1,000000
/ BNDVAL    C0009          1,000000
/ BNDVAL    C0010          1,000000
/ BNDVAL    C0011          1,000000
/ BNDVAL    C0012          1,000000
/ BNDVAL    C0013          1,000000
/ BNDVAL    C0014          1,000000
/ BNDVAL    C0015          1,000000
/ BNDVAL    C0016          1,000000
/ BNDVAL    C0017          1,000000
/ BNDVAL    C0018          1,000000
/ BNDVAL    C0019          1,000000
/ BNDVAL    C0020          1,000000
/ BNDVAL    C0021          1,000000
/ BNDVAL    C0022          1,000000
/ BNDVAL    C0023          1,000000
/ BNDVAL    C0024          1,000000
/ BNDVAL    C0025          1,000000
/ BNDVAL    C0026          1,000000
/ BNDVAL    C0027          1,000000
/ BNDVAL    C0028          1,000000
/ BNDVAL    C0029          1,000000
/ BNDVAL    C0030          1,000000
/ BNDVAL    C0031          1,000000
)ATA
```

```
. SCICONIC

         SCICONIC/VM       VERSION VM/P1.32
             COPYRIGHT SCICON LTD, 1983

               AUTHORISED FOR USE AT:
             UNIVERSITY OF LOUGHBOROUGH
·INFILE='MTSP10'
·CONVERT
:W PROBLEM QA4RT32
iS   VECTOR - RHS
)UND VECTOR - BNDVAL
{OBLEM HAS     57 ROWS,    125 COLUMNS AND   416 NON-ZERO ELEMENTS
)NVERT TOOK   3.87 SECONDS
·SETUP(MAXIMISE)
{OBLEM QA4RT32  ON FILE
(EATED ON  13-JUL-1986 AT 12:40:28
{OBLEM HAS     57 ROWS,     125 COLUMNS AND    416 NON-ZERO ELEMENTS
iS      - RHS
)UND    - BNDVAL
iJECTIVE - OBJ
ICORE MATRIX HAS     57 ROWS AND    125 COLUMNS
:TUP TOOK   1.34 SECONDS
·PRIMAL


    NITS              OBJECT              INFEAS                  SECS
       0               0.000000            0.000000(       0)       1.58
      36             -41.275975            0.000000(       0)       4.50

   SOLUTION IS OPTIMAL
·PRINTSOLN


     PROBLEM QA4RT32    - SOLUTION NUMBER     1  - OPTIMAL

     CREATED ON    13-JUL-1986 AT 12:41:10, AFTER      36 ITERATIONS

     PRINTED ON    13-JUL-1986 AT 12:41:19

     ...NAME...              ..ACTIVITY..   DEFINED AS

     FUNCTIONAL                41.275975     OBJ
     RESTRAINTS                              RHS
     BOUNDS....                              BNDVAL


  ..ROW...     AT       ....ACTIVITY....
  OBJ          BS             -41.275975
  R0002        UL               1.000000
  R0004        UL               1.000000
  R0005        UL               1.000000
  R0006        UL               1.000000
  R0007        UL               1.000000
  R0008        UL               1.000000
  R0009        UL               1.000000
  R0010        UL               1.000000
  R0011        UL               1.000000
  R0012        UL               1.000000
  R0014        UL               1.000000
```

```
   R0015        UL                 1.000000
   R0016        UL                 1.000000
   R0017        UL                 1.000000
   R0018        UL                 1.000000
   R0019        UL                 1.000000
   R0021        UL                 1.000000
   R0022        UL                 1.000000
   R0023        UL                 1.000000
?:

   ..ROW...     AT          ....ACTIVITY....
   R0024        UL                 1.000000
   R0025        UL                 1.000000
   R0027        UL                 1.000000
   R0028        UL                 1.000000
   R0029        UL                 1.000000
   R0030        UL                 1.000000
   R0031        UL                 1.000000
   R0033        UL                 1.000000
   R0034        UL                 1.000000
   R0036        UL                 1.000000
   R0037        UL                 1.000000
   R0040        UL                 1.000000
   R0041        UL                 1.000000
   R0042        UL                 1.000000
   R0043        UL                 1.000000
   R0045        UL                 1.000000
   R0046        UL                 1.000000
*** END OF ROWS ***
?:

   .COLUMN.     AT          ....ACTIVITY....
   U0002        BS                 1.000000
   U0004        BS                 1.000000
   U0005        BS                 1.000000
   U0006        BS                 1.000000
   U0007        BS                 1.000000
   U0008        BS                 1.000000
   U0009        BS                 1.000000
   U0010        BS                 1.000000
   U0011        BS                 1.000000
   U0012        BS                 1.000000
   U0014        BS                 1.000000
   U0015        BS                 1.000000
   U0016        BS                 1.000000
   U0017        BS                 1.000000
   U0018        BS                 1.000000
   U0019        BS                 1.000000
   U0021        BS                 1.000000
   U0022        BS                 1.000000
   U0023        BS                 1.000000
   U0024        BS                 1.000000
?:

   .COLUMN.     AT          ....ACTIVITY....
   U0025        BS                 1.000000
   U0027        BS                 1.000000
   U0028        BS                 1.000000
   U0029        BS                 1.000000
   U0030        BS                 1.000000
```

```
  U0031        BS                  1.000000
  U0033        BS                  1.000000
  U0034        BS                  1.000000
  U0036        BS                  1.000000
  U0037        BS                  1.000000
  U0040        BS                  1.000000
  U0041        BS                  1.000000
  U0042        BS                  1.000000
  U0043        BS                  1.000000
  U0045        BS                  1.000000
  U0046        BS                  1.000000
** END OF COLUMNS ***
:
>STOP

** STOP
,
```

```
 C #SCIMY
 SEG
G Rev. 19.4.4]
O * #SCIMY
L B_EXTWILM
TA    ": SMALLER REDEFINITION OF COMMON
OL    ": SMALLER REDEFINITION OF COMMON
X4CM  ": SMALLER REDEFINITION OF COMMON
X3CM  ": SMALLER REDEFINITION OF COMMON
X2CM  ": SMALLER REDEFINITION OF COMMON
X1CM  ": SMALLER REDEFINITION OF COMMON
KEY   ": SMALLER REDEFINITION OF COMMON
D COMPLETE

 SEG #SCIMY

        SCICONIC/VM        VERSION VM/P1.32
            COPYRIGHT SCICON LTD. 1983

              AUTHORISED FOR USE AT:
            UNIVERSITY OF LOUGHBOROUGH
INFILE='MTSP10'
CONVERT
:W PROBLEM QA4RT32
IS    VECTOR - RHS
IUND VECTOR - BNDVAL
ROBLEM HAS     57 ROWS,    125 COLUMNS AND    416 NON-ZERO ELEMENTS
INVERT TOOK   3.76 SECONDS
SETUP(MAXIMISE)
ROBLEM QA4RT32  ON FILE
REATED ON  13-JUL-1986 AT 12:28:17
ROBLEM HAS     57 ROWS,    125 COLUMNS AND    416 NON-ZERO ELEMENTS
IS      - RHS
)UND    - BNDVAL
BJECTIVE - OBJ
NCORE MATRIX HAS     57 ROWS AND    125 COLUMNS
:TUP TOOK   1.36 SECONDS
>USER
 57   182
 PART  A
 PHASE  1
 PASS  1
 X( 44)=         0.000
 X( 69)=         0.000
 X( 91)=         0.000
 X(116)=         0.000
 X( 32)=         0.000
 X( 79)=         0.000
 X( 63)=         0.000
 X( 78)=         0.000
```

```
X(110)=         0.000
X(125)=         0.000
X( 75)=         0.000
X(122)=         0.000
X( 57)=         0.000
X( 66)=         0.000
X(104)=         0.000
X(113)=         0.000
X( 34)=         0.000
X( 81)=         0.000
X( 51)=         0.000
X( 98)=         0.000
X( 70)=         0.000
X(117)=         0.000
X(  1) EXTRANEOUS
X(  2) EXTRANEOUS
X(  3) EXTRANEOUS
X(  4) EXTRANEOUS
X(  5) EXTRANEOUS
X(  6) EXTRANEOUS
X(  7) EXTRANEOUS
X(  8) EXTRANEOUS
X(  9) EXTRANEOUS
X( 10) EXTRANEOUS
X( 11) EXTRANEOUS
X( 12) EXTRANEOUS
X( 13) EXTRANEOUS
X( 14) EXTRANEOUS
X( 15) EXTRANEOUS
X( 16) EXTRANEOUS
X( 17) EXTRANEOUS
X( 18) EXTRANEOUS
X( 19) EXTRANEOUS
X( 20) EXTRANEOUS
X( 21) EXTRANEOUS
X( 22) EXTRANEOUS
X( 23) EXTRANEOUS
X( 24) EXTRANEOUS
X( 25) EXTRANEOUS
X( 26) EXTRANEOUS
X( 27) EXTRANEOUS
X( 28) EXTRANEOUS
X( 29) EXTRANEOUS
X( 30) EXTRANEOUS
X( 31) EXTRANEOUS
LOWER SHADOW PRICE ON CONSTRAINT(  7)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT(  8)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT(  9)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 12)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 18)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 23)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 28)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 33)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 36)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 41)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 43)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 45)=         1.050
LOWER SHADOW PRICE ON CONSTRAINT( 46)=         1.050
X( 85)=         0.000
X( 86)=         0.000
```

```
X( 87)=          0,000
X( 90)=          0,000
X( 96)=          0,000
X(101)=          0,000
X(106)=          0,000
X(111)=          0,000
X(114)=          0,000
X(119)=          0,000
X(121)=          0,000
X(123)=          0,000
X(124)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 37)=          1,077
X(115)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT(  6)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 14)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 15)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 21)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 22)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 24)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 25)=          1,116
LOWER SHADOW PRICE ON CONSTRAINT( 27)=          1,116
X( 84)=          0,000
X( 92)=          0,000
X( 93)=          0,000
X( 99)=          0,000
X(100)=          0,000
X(102)=          0,000
X(103)=          0,000
X(105)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 40)=          1,129
X(118)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 30)=          1,168
X(108)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 10)=          1,182
LOWER SHADOW PRICE ON CONSTRAINT( 17)=          1,132
LOWER SHADOW PRICE ON CONSTRAINT( 19)=          1,182
X( 88)=          0,000
X( 95)=          0,000
X( 97)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 42)=          1,221
X(120)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 11)=          1,247
LOWER SHADOW PRICE ON CONSTRAINT( 16)=          1,247
X( 89)=          0,000
X( 94)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 29)=          1,260
X(107)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 31)=          1,287
X(109)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT(  4)=          1,313
LOWER SHADOW PRICE ON CONSTRAINT(  5)=          1,313
X( 82)=          0,000
X( 83)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT( 34)=          1,405
X(112)=          0,000
LOWER SHADOW PRICE ON CONSTRAINT(  2)=          1,484
X( 80)=          0,000
PASS  2
PHASE  2
PASS  1
```

```
PASS  2
PART  B
PHASE  1
PASS  1
UPPER BOUND X( 33) =          1.000
X( 33) =          1.000
UPPER BOUND X( 35) =          1.000
X( 35) =          1.000
UPPER BOUND X( 36) =          1.000
X( 36) =          1.000
UPPER BOUND X( 37) =          1.000
X( 37) =          1.000
UPPER BOUND X( 38) =          1.000
X( 38) =          1.000
UPPER BOUND X( 39) =          1.000
X( 39) =          1.000
UPPER BOUND X( 40) =          1.000
X( 40) =          1.000
UPPER BOUND X( 41) =          1.000
X( 41) =          1.000
UPPER BOUND X( 42) =          1.000
X( 42) =          1.000
UPPER BOUND X( 43) =          1.000
X( 43) =          1.000
UPPER BOUND X( 45) =          1.000
X( 45) =          1.000
UPPER BOUND X( 46) =          1.000
X( 46) =          1.000
UPPER BOUND X( 47) =          1.000
X( 47) =          1.000
UPPER BOUND X( 48) =          1.000
X( 48) =          1.000
UPPER BOUND X( 49) =          1.000
X( 49) =          1.000
UPPER BOUND X( 50) =          1.000
X( 50) =          1.000
UPPER BOUND X( 52) =          1.000
X( 52) =          1.000
UPPER BOUND X( 53) =          1.000
X( 53) =          1.000
UPPER BOUND X( 54) =          1.000
X( 54) =          1.000
UPPER BOUND X( 55) =          1.000
X( 55) =          1.000
UPPER BOUND X( 56) =          1.000
X( 56) =          1.000
UPPER BOUND X( 58) =          1.000
X( 58) =          1.000
UPPER BOUND X( 59) =          1.000
X( 59) =          1.000
UPPER BOUND X( 60) =          1.000
X( 60) =          1.000
UPPER BOUND X( 61) =          1.000
X( 61) =          1.000
UPPER BOUND X( 62) =          1.000
X( 62) =          1.000
UPPER BOUND X( 64) =          1.000
X( 64) =          1.000
UPPER BOUND X( 65) =          1.000
X( 65) =          1.000
```

```
  UPPER BOUND X( 67) =          1.000
  X( 67) =          1.000
  UPPER BOUND X( 68) =          1.000
  X( 68) =          1.000
  UPPER BOUND X( 71) =          1.000
  X( 71) =          1.000
  UPPER BOUND X( 72) =          1.000
  X( 72) =          1.000
  UPPER BOUND X( 73) =          1.000
  X( 73) =          1.000
  UPPER BOUND X( 74) =          1.000
  X( 74) =          1.000
  UPPER BOUND X( 76) =          1.000
  X( 76) =          1.000
  UPPER BOUND X( 77) =          1.000
  X( 77) =          1.000
  PROBLEM IS SOLVED          OBJ =          41.275
I>STOP

*** STOP
K,
```

EXTENDED REDUCE METHOD

```
      SUBROUTINE USER
$INSERT SCICON>S>PDPPARAMS
$INSERT SCICON>S>PDPMCKEY
$INSERT SCICON>S>PDPMRKEY
$INSERT SCICON>S>PDPITER
$INSERT SCICON>S>PDPBITS
$INSERT SCICON>S>PDPUSEFUL
$INSERT SCICON>S>PDPMATRIX
$INSERT SCICON>S>PDPPOOL
$INSERT SCICON>S>PDPBETA
      REAL*8 COLELL(2048),ROWELL(2048),ROWELJ(2048),ROWGLL(2048),
     *       RHS(512),RHS1(512),X(512),X0(512),C(512),P1(512),DST(512),
     *       AAX,AMIN,BOV,R,CMIN,GMIN,DOV,DEF
      INTEGER*2 ICOLNO(2048),IROWNO(2048),JROWNO(2048),IRGWNO(2048),
     *          ICOLMK(512),IROWMK(512),JROWMK(512),IRGWMK(512),
     *          NRD(512),IC(512),IS(512),NRDD(512),IDX(512),IDDX(512),
     *          NCD(512),ISS(512),IRT(512),
     *          JNEGT(512),JPVC(512),IPVR(512),JNCD(512)
      INTEGER*2 TIM(28)
      COMMON/882COM/COLELL,ROWELL,ROWELJ,ROWGLL
      COMMON/ALAA2/ICOLNO,IRGWNO,IROWNO,JROWNO
      COMMON/ALAA3/X,X0,C,RHS,RHS1,P1,DST
      COMMON/ALAA4/ICOLMK,IROWMK,IRGWMK,JROWMK
      COMMON/ALAA5/IS,IC,ISS,IRT,IDX,NRD,NCD,LS,
     *               NRDD,IDDX,JNEGT,JPVC,IPVR,JNCD
      ITR=0
      JQ2=0
      WRITE(1,994) NROW,NSEQ
994   FORMAT(2X,I3,3X,I3)
      XX=1000000.0
C
C
C
C     SETTING THE VALUES FROM ARRAY POOL ...
C
C
      NNROW=NROW+1
      N=1
      K=1
      DO 1500 JSEQ=NNROW,NSEQ
      J=JSEQ-NROW
      IC(J)=JSEQ
      ICOLMK(J)=K
      KLMEL=MSMEL(JSEQ)+MSKMEB(JSEQ)
      LLMEL=MSMEL(JSEQ+1)
      L=0
      DO 1600 ILMEL=KLMEL,LLMEL
      IROW=MRWME(ILMEL)
      IPOOL=MPTME(ILMEL)
      N=N+1
      L=L+1
      COLELL(N)=POOL(IPOOL)
      ICOLNO(N)=IROW
1600  CONTINUE
      ICOLNO(K)=L
      NCD(J)=L
      IF(AND(MCKEY(JSEQ),XCBUBC).EQ.0) GO TO 1660
      IPOOL=MPTME(KLMEL-1)
      COLELL(K)=POOL(IPOOL)
      C(J)=COLELL(K)
      IF(C(J).LT.-0.1E-8) JQ2=1
1660  K=K+L+1
      N=N+1
1500  CONTINUE
      K=1
      DO 1700 I=2,NROW
      RHS(I)=BETA(I)
      IDX(I)=I
      IS(I)=I
      IR=K
      IROWMK(I)=K
      IRGWMK(I)=K
      L=0
      DO 1800 JSEQ=NNROW,NSEQ
      J=JSEQ-NROW
      KLMEL=MSMEL(JSEQ)+MSKMEB(JSEQ)
      LLMEL=MSMEL(JSEQ+1)
      DO 1900 ILMEL=KLMEL,LLMEL
      IROW=MRWME(ILMEL)
      IF(IROW.NE.I) GO TO 1900
      L=L+1
      K=K+1
      IPOOL=MPTME(ILMEL)
      ROWELL(K)=POOL(IPOOL)
      ROWGLL(K)=ROWELL(K)
      IROWNO(K)=J
      IRGWNO(K)=J
      ILMEL=LLMEL
1900  CONTINUE
1800  CONTINUE
      IROWNO(IR)=L
      IRGWNO(IR)=L
      NRD(I)=L
      K=K+1
1700  CONTINUE
      NRW=NROW
C
C
C
C
```

```
C
103   FORMAT(2X,'CONSTRAINT',I3,'REDUNDANT')
104   FORMAT(2X,'X(',I3,') EXTERANEOUS')
105   FORMAT(2X,'S(',I3,') EXTRANEOUS')
107   FORMAT(2X,'PASS(',I2,')')
1     FORMAT(2X,'STEP 1')
2     FORMAT(2X,'STEP 2')
3     FORMAT(2X,'STEP 3')
4     FORMAT(2X,'STEP 4')
5     FORMAT(2X,'STEP 5')
6     FORMAT(2X,'STEP 6')
7     FORMAT(2X,'STEP 7')
C
C
C
C
C
10    IPASS=IPASS+1
      WRITE(1,107) IPASS
C
C
C     STEP (1)
C
      WRITE(1,1)
      IOCOR=0
      IF(JC2.EQ.0) GO TO 100
C
C
C
C
C     STEP (2) ...
C
C
      WRITE(1,2)
      DO 21 J2=1,NCOL
      JNEG=0
      IRT(J2)=0
      JNCD(J2)=0
      IF(NCD(J2).EQ.0) GO TO 21
      IF(C(J2).LT.0.0) GO TO 21
      JX=ICOLMK(J2)
      JY=ICOLNO(JX)
      NS=JX+1
      NL=JX+JY
      DO 25 K=NS,NL
      IF(COLELL(K).GE.0.0) GO TO 25
      JNEG=1
      K=NL
25    CONTINUE
      IF(JNEG.EQ.1) GO TO 21
C
C
      DO 22 K1=NS,NL
      IF(COLELL(K1).EQ.0.0) GO TO 22
      COLELL(K1)=0.0
      I2=ICOLNO(K1)
      IX=IROWMK(I2)
      IY=IROWNO(IX)
      IXX=IX+1
      IYY=IX+IY
      DO 23 K2=IXX,IYY
      IF(IROWNO(K2).NE.J2) GO TO 23
      ROWELL(K2)=0.0
      NRD(I2)=NRD(I2)-1
      K2=IYY
23    CONTINUE
22    CONTINUE
      NCD(J2)=0
      IF(IC(J2).GT.NROW) GO TO 24
      INR=IC(J2)-1
      WRITE(1,105) INR
      MRKEY(IC(J2))=OR(KRBFRE,MRKEY(IC(J2)))
      GO TO 21
24    INR=IC(J2)-NROW
      WRITE(1,104) INR
      MCKEY(IC(J2))=AND(MCKEY(J2)),KCBAR)
      IOCOR=1
21    CONTINUE
C
C
C
C
C     STEP (3) ...
C
C
      WRITE(1,3)
      IRD=0
      DO 31 I3=2,NRW
      IPOS=0
      IPVR(I3)=0
      IF(IS(I3).GT.NROW) GO TO 31
      IF(NRD(I3).EQ.0) GO TO 31
      IX=IROWMK(I3)
      IY=IROWNO(IX)
      JS=IX+1
      JL=IX+IY
      DO 30 K=JS,JL
      IF(ROWELL(K).LE.0.0) GO TO 30
      IPOS=1
      K=JL
```

```
30    CONTINUE
      IF(IPOS.EQ.1) GO TO 31
C
C
      NRD(I3)=0
      IOCOR=1
      DO 33 K=JS,JL
      IF(ROWELL(K).EQ.0.0) GO TO 33
      ROWELL(K)=0.0
      J=IROWNO(K)
      JR=ICOLMK(J)
      JT=ICOLNO(JR)
      IX=JR+1
      IY=JR+JT
      DO 34 KK=IX,IY
      IF(ICOLNO(KK).NE.I3) GO TO 34
      COLELL(KK)=0.0
      NCD(J)=NCD(J)-1
      KK=IY
34    CONTINUE
33    CONTINUE
      RHS(I3)=0.0
      INR=IS(I3)-1
      WRITE(1,103) INR
      MRKEY(IS(I3))=OR(KRBFRE,MRKEY(IS(I3)))
31    CONTINUE
C
C
C     STEP (4) ...
C
C
      WRITE(1,4)
      DO 41 I5=2,NROW
      IPOZ=0
      IF(NRD(I5).EQ.0) GO TO 41
      IF(RHS(I5).NE.0.0) GO TO 41
      IX=IROWMK(I5)
      IY=IROWNO(IX)
      JS=IX+1
      JL=IX+IY
      DO 40 K=JS,JL
      IF(ROWELL(K).GE.0.0) GO TO 40
      IPOZ=1
      K=JL
40    CONTINUE
      IF(IPOZ.EQ.1) GO TO 41
C
C
      DO 42 K=JS,JL
      IF(ROWELL(K).EQ.0.0) GO TO 42
      ROWELL(K)=0.0
      J5=IROWNO(K)
      JX=ICOLMK(J5)
      JY=ICOLNO(JX)
      IX=JX+1
      IY=JX+JY
      DO 43 K1=IX,IY
      IF(COLELL(K1).EQ.0.0) GO TO 43
      COLELL(K1)=0.0
      IR=ICOLNO(K1)
      IN=IROWMK(IR)
      IM=IROWNO(IN)
      IXX=IN+1
      IYY=IN+IM
      DO 44 K2=IXX,IYY
      IF(IROWNO(K2).NE.J5) GO TO 44
      ROWELL(K2)=0.0
      K2=IYY
44    CONTINUE
43    CONTINUE
      IF(IC(J5).GT.NROW) GO TO 45
      INR=IC(J5)-1
      WRITE(1,105) INR
      MRKEY(IC(J5))=OR(KRBFRE,MRKEY(IC(J5)))
      GO TO 516
45    INR=IC(J5)-NROW
      WRITE(1,104) INR
      MCKEY(IC(J5))=AND(MCKEY(IC(J5),KCBART)
516   NCD(J5)=0
      IOCOR=1
42    CONTINUE
      NRD(I5)=0
      INR=IS(I5)-1
      WRITE(1,103) INR
      MRKEY(IS(I5))=OR(KRBFRE,MRKEY(IS(I5)))
41    CONTINUE
C
C
C
C     STEP (5) ...
C
C
50    WRITE(1,5)
C
C     FINDING THE PIVOT COLUMN ...
C
51    J=0
      CMIN=XX
      DO 55 J5=1,NCOL
      IF(JPVC(J5).EQ.1) GO TO 55
```

```
      IF(NCD(J5).EQ.0) GO TO 55
      IF(C(J5).GE.-0.1E-8) GO TO 55
      IF(C(J5).GT.CMIN) GO TO 55
      CMIN=C(J5)
      J=J5
55    CONTINUE
      IF(J.EQ.0) GO TO 100
C
C
C     FINDING THE PIVOT ROW ...
C
      IQ=0
      IUQ=0
      GMIN=XX
      IX=ICOLMK(J)
      IY=ICOLNO(IX)
      JS=IX+1
      JL=IX+IY
      DO 52 K=JS,JL
      IF(COLELL(K).LE.0.1E-8) GO TO 52
      IA=ICOLNO(K)
      BOV=RHS(IA)/COLELL(K)
      DEF=GMIN-BOV
      IF(DEF.GT.0.1E-5) GO TO 53
      IF(DEF.LT.-0.1E-5) GO TO 52
      IUQ=IUQ+1
      GO TO 54
53    IUQ=1
      GMIN=BOV
54    IQ=IA
      KJ=K
52    CONTINUE
      IF(IUQ.EQ.1) GO TO 80
      IF(IUQ.NE.0) GO TO 60
      JPVC(J)=1
      GO TO 51
C
C
C     STEP (6) ...
C
60    WRITE(1,6)
C
C     FINDING THE NEW INTERIOR POINT ...
C
      XO(J)=RHS(IQ)/COLELL(KJ)
      DO 67 K=JS,JL
      IA=ICOLNO(K)
      IF(IA.EQ.IQ) GO TO 67
      IF(NRD(IA).EQ.0) GO TO 67
      IF(IS(IA).LE.NROW) GO TO 67
      JV=IS(IA)-NROW
      XO(JV)=RHS(IA)-XO(J)*COLELL(K)
      XO(JV)=XO(JV)-0.01
67    CONTINUE
      XO(J)=XO(J)-0.01
C
C
C     UPDATING THE PIVOT ROW ...
C
C     FINDING THE NEXT PIVOT COLUMN ...
C
      CMIN=0.0
      JNC=0
      IX=IROWMK(IQ)
      IY=IROWNO(IX)
      JA=IX+1
      JB=IX+IY
      K=JA
      DO 64 J6=1,NCOL
      IF(IROWNO(K).NE.J6) GO TO 65
      IF(NCD(J6).EQ.0) GO TO 68
      ZC=C(J6)-C(J)*(ROWELL(K)/COLELL(KJ))
      IF(ZC.GE.0.0.OR.ZC.GE.CMIN) GO TO 68
      CMIN=ZC
      JNC=J6
68    K=K+1
      GO TO 64
65    IF(NCD(J6).EQ.0) GO TO 64
      IF(C(J6).GE.CMIN) GO TO 64
      CMIN=C(J6)
      JNC=J6
64    CONTINUE
      IF(JNC.EQ.0) GO TO 80
      XO(JNC)=0.01
C
      DO 61 K=JS,JL
      IF(COLELL(K).LE.0.1E-5) GO TO 61
      IA=ICOLNO(K)
      DOV=RHS(IA)/COLELL(K)
      IF(GMIN.NE.DOV) GO TO 61
66    IPVR(IA)=1
61    CONTINUE
C
C
C     FINDING THE DISTANCES BETWEEN THE INTERIOR POINT
      AND THE CONSTRAINTS ...
C
C
69    NT=0
      AMIN=XX
```

```
      DO 62 I6=2,NRW
      IF(IPVR(I6).NE.1) GO TO 62
      KS=0
      AAX=0.0
      IR=IDX(I6)
      IX=IROWMK(IR)
      IY=IROWNO(IX)
      IXX=IX+1
      IYY=IX+IY
      DO 63 K=IXX,IYY
      JR=IROWNO(K)
      IF(JR.EQ.JNC) KS=K
      AAX=AAX+XO(JR)*ROWGLL(K)
63    CONTINUE
      IF(KS.EQ.0) GO TO 62
      DST(I6)=(BETA(IR)-AAX)/ROWGLL(KS)
      IF(DST(I6).GT.AMIN) GO TO 62
      AMIN=DST(I6)
62    CONTINUE
      IF(AMIN.NE.XX) GO TO 70
      IF(NT.EQ.1) GO TO 30
      NT=1
      JNC=J
      GO TO 69
C
C
70    DO 71 KS=JS,JL
      I=ICOLNO(KS)
      IF(IPVR(I).EQ.0) GO TO 71
      IF(DST(I).EQ.AMIN) GO TO 74
      NRD(I)=0
      IOCOR=1
      INR=IS(I)-1
      WRITE(1,103) INR
      MRKEY(IS(I))=OR(KRBFRE,MRKEY(IS(I)))
      IX=IROWMK(I)
      IY=IROWNO(IX)
      IXX=IX+1
      IYY=IX+IY
      DO 72 K1=IXX,IYY
      IF(ROWELL(K1).EQ.0.0) GO TO 72
      ROWELL(K1)=0.0
      JR=IROWNO(K1)
      JX=ICOLMK(JR)
      JY=ICOLNO(JX)
      JSS=JX+1
      JLL=JX+JY
      DO 73 K2=JSS,JLL
      IF(ICOLNO(K2).NE.I) GO TO 73
      COLELL(K2)=0.0
      NCD(JR)=NCD(JR)-1
      K2=JLL
73    CONTINUE
72    CONTINUE
      GO TO 71
74    KJ=KS
      IQ=I
71    CONTINUE
C
C
C
80    IF(IOCOR.EQ.1) GO TO 81
      IF(IPASS.EQ.1) GO TO 81
      GO TO 100
C
C
C     STEP (7) ...
C
C     PERFORMING THE SIMPLEX ITERATION ...
C
C
81    WRITE(1,7)
      R=1/COLELL(KJ)
      IE=0
      IW=1
      IROW=2
      JROWMK(IROW)=IW
      JX=IROWMK(IQ)
      JY=IROWNO(JX)
      JR=JX+1
      JQ=JX+JY
      ROWELJ(IW)=ROWELL(JX)
      IW=IW+1
      DO 8017 K=JR,JQ
      IF(ROWELL(K).EQ.0.0) GO TO 8017
      JA=IROWNO(K)
      IF(JA.NE.J) GO TO 8117
      ROWELL(K)=R
      ROWELJ(IW)=R
      JROWNO(IW)=JA
      IW=IW+1
      IE=IE+1
      JNCD(JA)=JNCD(JA)+1
      GO TO 8017
8117  ROWELL(K)=R*ROWELL(K)
      ROWELJ(IW)=ROWELL(K)
      JROWNO(IW)=JA
      IW=IW+1
      IE=IE+1
      JNCD(JA)=JNCD(JA)+1
```

```
      JNCD(JA)=JNCD(JA)+1
8017  CONTINUE
      RHS(IQ)=R*RHS(IQ)
      JROWNO(1)=IE
C
C
      RHS1(IROW)=RHS(IQ)
      ISS(IROW)=IS(IQ)
      NRDD(IROW)=IE
      IDDX(IROW)=IDX(IQ)
C
C
      DO 8070 IA=2,NRW
      IF(IA.EQ.IQ) GO TO 8070
      IF(NRD(IA).EQ.0) GO TO 8070
      IROW=IROW+1
      IS=0
      IZ=IW
      IW=IW+1
      IX=IROWMK(IA)
      IY=IROWNO(IX)
      ROWELJ(IZ)=ROWELL(IX)
      JPV=0
      JS=IX+1
      JL=IX+IY
      DO 8170 KS=JS,JL
      IF(IROWNO(KS).NE.J) GO TO 8170
      IF(ROWELL(KS).EQ.0.0) GO TO 8770
      JPV=KS
8770  KS=JL
8170  CONTINUE
C
C
      K=JS
      DO 8171 JA=1,NCOL
      IF(JA.EQ.J) GO TO 8177
      IPV=0
      JR=2
      JQ=1+JROWNO(1)
      DO 8172 K1=JR,JQ
      IF(JROWNO(K1).NE.JA) GO TO 8172
      IPV=K1
      K1=JQ
8172  CONTINUE
C
C
      IF(IRT(JA).EQ.1) GO TO 8178
      IF(IPV.EQ.0) GO TO 8178
      C(JA)=C(JA)-ROWELJ(IPV)*C(J)
      IRT(JA)=1
C
8178  IF(IROWNO(K).NE.JA) GO TO 8179
C
C
      IF(ROWELL(K).EQ.0.0) GO TO 8173
      IF(JPV.EQ.0.OR.IPV.EQ.0) GO TO 8174
      ROWELJ(IW)=ROWELL(K)-ROWELL(JPV)*ROWELJ(IPV)
      IF(ROWELJ(IW).LE.0.1E-8.AND.ROWELJ(IW).GE.-0.1E-8) GO TO 8176
      GO TO 8175
8173  IF(NCD(JA).EQ.0) GO TO 8176
      IF(JPV.EQ.0.OR.IPV.EQ.0) GO TO 8176
      ROWELJ(IW)=-ROWELL(JPV)*ROWELJ(IPV)
      GO TO 8175
8174  ROWELJ(IW)=ROWELL(K)
      GO TO 8175
8177  IF(IROWNO(K).NE.JA) GO TO 8171
      IF(JPV.EQ.0) GO TO 8176
      ROWELJ(IW)=-R*ROWELL(JPV)
8175  JROWNO(IW)=JA
      IW=IW+1
      IE=IE+1
      JNCD(JA)=JNCD(JA)+1
8176  K=K+1
      GO TO 8171
8179  IF(NCD(JA).EQ.0) GO TO 8171
      IF(JPV.EQ.0.OR.IPV.EQ.0) GO TO 8171
      ROWELJ(IW)=-ROWELL(JPV)*ROWELJ(IPV)
      JROWNO(IW)=JA
      IW=IW+1
      IE=IE+1
      JNCD(JA)=JNCD(JA)+1
8171  CONTINUE
C
C
      JROWMK(IROW)=IZ
      JROWNO(IZ)=IE
      NRDD(IROW)=JROWNO(IZ)
      IF(JPV.EQ.0) GO TO 8771
      RHS(IA)=RHS(IA)-ROWELL(JPV)*RHS(IQ)
8771  RHS1(IROW)=RHS(IA)
      ISS(IROW)=IS(IA)
      IDDX(IROW)=IDX(IA)
8070  CONTINUE
      C(J)=-R*C(J)
      NRW=IROW
C
C
C
C
C
```

```
      JQ2=0
      DO 8091 I1=2,NRW
      IROWMK(I1)=JROWMK(I1)
      IX=IROWMK(I1)
      ROWELL(IX)=ROWELJ(IX)
      IROWNO(IX)=JROWNO(IX)
      IY=IROWNO(IX)
      IXX=IX+1
      IYY=IX+IY
      DO 8092 K2=IXX,IYY
      IROWNO(K2)=JROWNO(K2)
      ROWELL(K2)=ROWELJ(K2)
8092  CONTINUE
8091  CONTINUE
C
C
      N=1
      DO 8093 J1=1,NCOL
      JPVC(J1)=0
      NCD(J1)=JNCD(J1)
      IF(NCD(J1).EQ.0) GO TO 8093
      ICOLMK(J1)=N
      IZ=N
      N=N+1
      DO 8094 I1=2,NRW
      JX=IROWMK(I1)
      JY=IROWNO(JX)
      JXX=JX+1
      JYY=JX+JY
      DO 8095 K=JXX,JYY
      IF(IROWNO(K).NE.J1) GO TO 8095
      COLELL(N)=ROWELL(K)
      ICOLNO(N)=I1
      N=N+1
      K=JYY
8095  CONTINUE
8094  CONTINUE
      ICOLNO(IZ)=N-IZ-1
      IF(C(J1).LT.-0.1E-5) JQ2=1
8093  CONTINUE
C
C
C
      ISTOR=ISS(2)
      ISS(2)=IC(J)
      IC(J)=ISTOR
C
C
C
      DO 8096 IROW=2,NRW
      IS(IROW)=ISS(IROW)
      RHS(IROW)=RHS1(IROW)
      NRD(IROW)=NRDD(IROW)
      IDX(IROW)=IDDX(IROW)
8096  CONTINUE
C
C
C
      GO TO 10
C
      RETURN
      END
C
C
```

EXTENDED WILLIAMS METHOD

```
      SUBROUTINE USER
$INSERT SCICON>S>PDPPARAMS
$INSERT SCICON>S>PDPMCKEY
$INSERT SCICON>S>PDPITER
$INSERT SCICON>S>PDPBITS
$INSERT SCICON>S>PDPUSEFUL
$INSERT SCICON>S>PDPMATRIX
$INSERT SCICON>S>PDPPOOL
$INSERT SCICON>S>PDPBETA
      REAL*8 RHS(512)
      REAL*8 COLELL(2048)
      REAL*8 ROWELL(2048)
      REAL*8 S(2048)
      REAL*8 SN(2048)
      REAL*8 U(512)
      REAL*8 UP(512)
      REAL*8 RWC(512)
      REAL*8 RUC(512)
      REAL*8 RW(512)
      REAL*8 RU(512)
      REAL*8 P(512),PP(512)
      REAL*8 Q(512),QP(512)
      REAL*8 SC(512)
      REAL*8 C(512)
      REAL*8 CC(512)
      REAL*8 X(512)
      REAL*8 PS,QS,WT,UT,DFC,DFU,DFE,ELM1.ELM2,AMIN
      INTEGER*2 ICOLNO(2048)
      INTEGER*2 IROWNO(2048)
      INTEGER*2 ICOLMK(512)
      INTEGER*2 IROWMK(512)
      INTEGER*2 KK(512)
      INTEGER*2 KZ(512)
      INTEGER*2 ISC(512)
      INTEGER*2 JCN(512)
      INTEGER*2 IRN(512)
      INTEGER*2 JV(512)
      INTEGER*2 IC(512)
      INTEGER*2 JSC(512)
      COMMON/ALAA1/COLELL,ROWELL,ICOLNO,IROWNO,ICOLMK,IROWMK
      COMMON/ALAA2/SC,S,SN,RHS,X,C,CC
      COMMON/ALAA3/KK,KZ,ISC,JCN,IRN,JV,IC,JSC
      COMMON/ALAA4/U,UP,RWC,RUC,P,Q,RW,RU,PP,QP
      WRITE(1,998) NROW,NSEQ
998   FORMAT(3X,I3,3X,I3)
      XX=1000000.0
C
C
      NNROW=NROW+1
      N=1
      K=1
      DO 1500 JSEQ=NNROW,NSEQ
      J=JSEQ-NROW
      IC(J)=J
      ICOLMK(J)=K
      KLMEL=MSMEL(JSEQ)+MSKMEB(JSEQ)
      LLMEL=MSMEL(JSEQ+1)
      L=0
      DO 1600 ILMEL=KLMSL,LLMEL
      IROW=MRWME(ILMEL)
      IPOOL=MPTME(ILMEL)
      N=N+1
      L=L+1
      ICOLNO(N)=IROW
      COLELL(N)=POOL(IPOOL)
      IF(COLELL(N).GT.0.0) GO TO 1600
      JCN(J)=JCN(J)+1
      IRN(IROW)=IRN(IROW)+1
1600  CONTINUE
      ICOLNO(K)=L
      KK(J)=L
      IF(AND(MCKEY(JSEW),KCBUBC).EQ.0) GO TO 1650
      IPOOL=MPTME(KLMEL-1)
      COLELL(K)=POOL(IPOOL)
      C(J)=-COLELL(K)
      SC(J)=-COLELL(K)
      CC(J)=-COLELL(K)
      IPOOL=MRWME(KLMEL-1)
      IF(IPOOL.NE.KPTPLI) GO TO 1660
1650  U(J)=XX
      GO TO 1670
1660  U(J)=POOL(IPOOL)
1670  K=K+L+1
      N=N+1
1500  CONTINUE
      K=1
      DO 1700 I=2,NROW
      RHS(I)=BETA(I)
      Q(I)=XX
      P(I)=0.0
      L=0
      IROWMK(I)=K
      IR=K
      DO 1800 JSEQ=NNROW,NSEQ
      J=JSEQ-NROW
      KLMEL=MSMEL(JSEQ)+MSKMEB(JSEQ)
      LLMEL=MSMEL(JSEQ+1)
      DO 1900 ILMEL=KLMEL,LLMEL
      IROW=MRWME(ILMEL)
      IF(IROW.NE.I) GO TO 1900
```

```
      L=L+1
      K=K+1
      IPOOL=MPTME(ILMEL)
      ROWELL(K)=POOL(IPOOL)
      IROWNO(K)=J
      ILMEL=LLMEL
1900  CONTINUE
1800  CONTINUE
      IROWNO(IR)=L
      KZ(I)=L
      K=K+1
1700  CONTINUE
C
C
C
      WRITE(1,2001)
2001  FORMAT(3X,'PART  A')
      WRITE(1,2002)
2002  FORMAT(3X,'PHASE  1')
      IPART=0
      IPHASE=0
      IPASS=1
      IPSACT=0
      IDSC2=1
2003  FORMAT(3X,'PASS ',I2)
C
C
C     SETTING THE COLUMNS IN AN ACSENDING ORDER ACCORDING
      TO THEIR COST COEFFICIENTS ...
C
C
      DO 1 J1=1,NCOL
      AMIN=XX
      DO 2 J2=1,NCOL
      IF(IC(J2).EQ.0) GO TO 2
      IF(AMIN.LE.C(J2)) GO TO 2
      AMIN=C(J2)
      J=J2
2     CONTINUE
      JV(J1)=J
      IC(J)=0
1     CONTINUE
C
C
C
1009  WRITE(1,2003) IPASS
      DO 1000 JG1=1,NCOL
      IC(JG1)=JG1
      J=JV(JG1)
      IF(ISC(J).EQ.1.OR.ISC(J).EQ.2) GO TO 1000
C
C
C
C
      N=ICOLMK(J)
      M=ICOLNO(N)
      IN=N+1
      IM=N+M
      IF(KK(J).EQ.0) GO TO 1004
      IF(IPASS.EQ.1) GO TO 1004
C
C
C
C     TIGHTENING THE PRIMAL BOUNDS ...
C
      UP(J)=U(J)
      DO 901 K=IN,IM
      IF(COLELL(K).LE.0.0) GO TO 901
      I=ICOLNO(K)
      IF(RWC(I).EQ.-XX) GO TO 901
      UT=(RHS(I)-RWC(I))/COLELL(K)
      IF(UP(J).LE.UT) GO TO 901
      UP(J)=UT
901   CONTINUE
C
C
C
      IF(UP(J).NE.0.0) GO TO 902
      U(J)=0.0
      X(J)=U(J)
      WRITE(1,202) J,X(J)
      GO TO 140
902   IF(IPHASE.EQ.1) GO TO 1004
      IF(UP(J).GE.U(J)) GO TO 1004
      U(J)=UP(J)
      IPSACT=1
      WRITE(1,590)J,UP(J)
590   FORMAT(3X,'UPPER BOUND (',I3,')TIGHTENED TO',F14.3)
C
C
C
C     CALCULATING THE UPPER AND LOWER COST OF THE VARIABLE ...
C
1004  PS=0.0
      QS=0.0
      IF(KK(J).EQ.0) GO TO 1006
      DO 10 K=IN,IM
      I=ICOLNO(K)
      IF(COLELL(K)) 30,10,20
20    JF=K
```

```
      IF(Q(I).EQ.XX.OR.QS.EQ.XX) GO TO 13
      QS=QS+COLELL(K)*Q(I)
      GO TO 14
13    QS=XX
14    IF(PS.EQ.-XX) GO TO 10
      PS=PS+COLELL(K)*P(I)
      GO TO 10
30    JF=K
      IF(Q(I).EQ.XX.OR.PS.EQ.-XX) GO TO 15
      PS=PS+COLELL(K)*Q(I)
      GO TO 16
15    PS=-XX
16    IF(QS.EQ.XX) GO TO 10
      QS=QS+COLELL(K)*P(I)
10    CONTINUE
C
C
C
C     FIXING VARIABLES AT THEIR BOUNDS ...
C
1006  IF(PS.GT.C(J)) GO TO 120
      IF(QS.LT.C(J)) GO TO 130
      GO TO 1005
120   X(J)=0.0
      IPSACT=1
      WRITE(1,160) J,X(J)
160   FORMAT(3X,'X(',I3,')=',F14.3,3X,'IV')
      GO TO 140
130   IF(U(J).EQ.XX) GO TO 170
      X(J)=U(J)
      IPSACT=1
      WRITE(1,160) J,X(J)
140   DO 6 K=IN,IM
      I=ICOLNO(K)
      IF(S(K).NE.0.0) RHS(I)=RHS(I)-S(K)*X(J)
      S(K)=0.0
      IF(COLELL(K).EQ.0.0) GO TO 6
      COLELL(K)=0.0
      NN=IROWMK(I)
      MM=IROWNO(NN)
      INN=NN+1
      IMM=NN+MM
      DO 6666 IK=INN,IMM
      IF(IROWNO(IK).NE.J) GO TO 6666
      RHS(I)=RHS(I)-ROWELL(IK)*X(J)
      BETA(I)=BETA(I)-ROWELL(IK)*X(J)
      IF(ROWELL(IK).LT.0.0) IRN(I)=IRN(I)-1
      ROWELL(IK)=0.0
      KZ(I)=KZ(I)-1
      IK=IMM
6666  CONTINUE
6     CONTINUE
      KK(J)=0
      ISC(J)=1
      JSEQ=NROW+J
      MCKEY(JSEQ)=AND(MCKEY(JSEQ),KCBART)
      GO TO 1000
170   WRITE(1,150) J
150   FORMAT(3X,'X(',I3,') IS UNBOUNDED')
      GO TO 999
C
C
C
1005  IF(IPART.EQ.1) GO TO 301
      IF(C(J).EQ.0.0) GO TO 301
      IF(KK(J).NE.1) GO TO 301
      IF(U(J).NE.XX) GO TO 301
C
C
C
C     REPACING SINGLETON COLUMN BY SHADOW PRICE ...
C
C
      I=ICOLNO(JF)
      IF(COLELL(JF)) 318,301,319
318   UT=C(J)/COLELL(JF)
      IF(UT.GE.Q(I)) GO TO 3000
      SN(JF)=COLELL(JF)
      ISC(J)=2
      Q(I)=UT
      IPSACT=1
      INROW=I-1
      WRITE(1,314)INROW,Q(I)
314   FORMAT(3X,'UPPER SHADOW PRICE ON CONSTRAINT(',I3,')=',F14.3)
      IPS=1
      IDS=1
      IDSC2=1
      GO TO 3200
319   WT=C(J)/COLELL(JF)
      IF(WT.LE.P(I)) GO TO 3000
      SN(JF)=COLELL(JF)
      ISC(J)=2
      P(I)=WT
      IPSACT=1
      INROW=I-1
      WRITE(1,317)INROW,P(I)
317   FORMAT(3X,'LOWER SHADOW PRICE ON CONSTRAINT(',I3,')=',F14.3)
      IPS=1
      IDS=1
      IDSC2=1
```

```
      GO TO 3200
C
C
C
C     TIGHTENING THE UPPER AND LOWER SHADOW PRICS ...
C
C
301   IF(QS.EQ.XX) GO TO 3000
      IF(C(J).EQ.0.0.AND.QS.EQ.0.0) GO TO 3000
C
C
C
      DO 730 K=IN,IM
      IP=0
      IQ=0
      I=ICOLNO(K)
      PP(I)=P(I)
      QP(I)=Q(I)
      IF(COLELL(K)) 731,730,732
732   WT=Q(I)+(C(J)-QS)/COLELL(K)
      IF(WT.LE.PP(I)) GO TO 730
      PP(I)=WT
      IP=1
      GO TO 733
731   UT=P(I)+(C(J)-QS)/COLELL(K)
      IF(UT.GE.QP(I)) GO TO 730
      QP(I)=UT
      IQ=1
733   IF(IPART.EQ.1) GO TO 735
      IF(PP(I).NE.QP(I)) GO TO 735
      P(I)=PP(I)
      Q(I)=QP(I)
      JX=IROWMK(I)
      JY=IROWNO(JX)
      JS=JX+1
      JL=JX+JY
      DO 734 K2=JS,JL
      IF(ROWELL(K2).EQ.0.0) GO TO 734
      JQ=IROWNO(K2)
      JSC(JQ)=3
      C(JQ)=C(JQ)-ROWELL(K2)*P(I)
      ROWELL(K2)=0.0
      IX=ICOLMK(JQ)
      IY=ICOLNO(IX)
      IS=IX+1
      IL=IX+IY
      DO 736 K3=IS,IL
      IF(ICOLNO(K3).NE.I) GO TO 736
      IF(COLELL(K3).LT.0.0) JCN(JQ)=JCN(JQ)-1
      S(K3)=COLELL(K3)
      COLELL(K3)=0.0
      KK(JQ)=KK(JQ)-1
      K3=IL
736   CONTINUE
734   CONTINUE
      KZ(I)=0
      IRN(I)=0
      IPSACT=1
      INROW=I-1
      GO TO 730
735   IF(IPHASE.EQ.0) GO TO 730
      IF(IP.EQ.0) GO TO 737
      IF(PP(I).LE.P(I)) GO TO 730
      P(I)=PP(I)
      IPSACT=1
      INROW=I-1
      WRITE(1,763) INROW,P(I)
      IDSC2=1
      GO TO 730
737   IF(IQ.EQ.0) GO TO 730
      IF(QP(I).GE.Q(I)) GO TO 730
      Q(I)=QP(I)
      IPSACT=1
      INROW=I-1
      WRITE(1,765) INROW,Q(I)
      IDSC2=1
730   CONTINUE
C
C
763   FORMAT(3X,'LOWER SHADOW PRICE ON CONSTRAINT',I3,'TIGHTENED TO',
     *F14.3)
765   FORMAT(3X,'UPPER SHADOW PRICE ON CONSTRAINT',I3,'TIGHTENED TO',
     *F14.3)
C
C
C
C     A COMPARISON BETWEEN THE COLUMNS ...
C
3000  IF(JSC(J).EQ.3) GO TO 3200
      IB=0
      IF(IDSC2.EQ.0) GO TO 3200
      IF(CC(J).LE.0.0.AND.JCN(J).EQ.0) GO TO 3222
      IF(JG1.EQ.NCOL) GO TO 3200
C
C
C
      JCL=JG1+1
      DO 3100 JG2=JCL,NCOL
      J2=JV(JG2)
      IF(KK(J2).EQ.0) GO TO 3100
```

```
      IF(ISC(J2).EQ.1.OR.ISC(J2).EQ.2) GO TO 3100
      IF(JSC(J2).EQ.3) GO TO 3100
C
C
C
      DFC=CC(J2)-CC(J)
      IF(DFC,LT,0,0) GO TO 3100
      DFU=0,0
      IE=0
      DO 3220 I=2,NROW
      IF(KZ(I),EQ,0) GO TO 3220
      DFE=0,0
      ELM1=0,0
      ELM2=0,0
      JX=IROWMK(I)
      JY=IROWNO(JX)
      JS=JX+1
      JL=JX+JY
      DO 3221 K=JS,JL
      IF(ROWELL(K),EQ,0,0) GO TO 3221
      JR=IROWNO(K)
      IF(JR,EQ,J2) ELM1=ROWELL(K)
      IF(JR,EQ,J) ELM2=ROWELL(K)
3221  CONTINUE
      DFE=ELM1-ELM2
      IF(DFE,LE,0,1E-5) GO TO 3224
      IF(Q(I),EQ,XX) GO TO 3223
      DFU=DFU+DFE*Q(I)
      GO TO 3220
3223  IE=1
      I=NROW
      GO TO 3220
3224  DFU=DFU+DFE*P(I)
3220  CONTINUE
      IF(IE,EQ,1) GO TO 3100
      IF(DFC,LE,DFU) GO TO 3100
C
C
C
3222  IPSACT=1
      WRITE(1,3101) J
3101  FORMAT(3X,'X(',I3,') EXTRANEOUS')
      JSEQ=NROW+J
      MCKEY(JSEQ)=AND(MCKEY(JSEQ),KCBART)
      KK(J)=0
      ISC(J)=1
      DO 3011 K1=IN,IM
      IF(COLELL(K1).EQ.0,0) GO TO 3011
      COLELL(K1)=0,0
      S(K1)=0,0
      I1=ICOLNO(K1)
      JX=IROWMK(I1)
      JY=IROWNO(JX)
      JS=JX+1
      JL=JX+JY
      DO 3033 K2=JS,JL
      IF(IROWNO(K2),NE,J) GO TO 3033
      IF(ROWELL(K2),LT,0,0) IRN(I1)=IRN(I1)-1
      ROWELL(K2)=0,0
      KZ(I1)=KZ(I1)-1
      K2=JL
3033  CONTINUE
3011  CONTINUE
      IB=1
      JC2=NCOL
C
C
3100  CONTINUE
      IF(IB,EQ,1) GO TO 1000
C
C
C
C     CALCULATING THE UPPER AND LOWER ACTIVITY CONSTRAINT ...
C
C
3200  DO 811 K=IN,IM
      I=ICOLNO(K)
      IF(COLELL(K)) 810,811,812
810   IF(U(J),EQ,XX) GO TO 8101
      IF(RW(I),EQ,-XX) GO TO 811
      RW(I)=RW(I)+COLELL(K)*U(J)
      GO TO 811
8101  RW(I)=-XX
      GO TO 811
812   IF(U(J),EQ,XX) GO TO 8120
      IF(RU(I),EQ,XX) GO TO 811
      RU(I)=RU(I)+COLELL(K)*U(J)
      GO TO 811
8120  RU(I)=XX
811   CONTINUE
C
C
C
1000  CONTINUE
C
C
      DO 1028 I=2,NROW
      RWC(I)=RW(I)
      RUC(I)=RU(I)
      RW(I)=0,0
```

```
      RU(I)=0,0
1028  CONTINUE
      IDSC2=0
C
C
      IF(IPART.EQ.1) GO TO 2000
C
C
C     REMOVING REDUNDANT CONSTRAINTS ...
C
C
      DO 3400 I1=2,NROW
      IF(KZ(I1),EQ,0) GO TO 3400
      IX=IROWMK(I1)
      IY=IROWNO(IX)
      IS=IX+1
      IL=IX+IY
      IF(P(I1),NE,0,0) GO TO 112
      IF(RHS(I1),EQ,0,0,AND,IRN(I1),EQ,0) GO TO 113
      IF(Q(I1),NE,XX) GO TO 112
      IF(RUC(I1),LT,RHS(I1)) GO TO 110
112   IF(RUC(I1),LT,RHS(I1),AND,P(I1),GT,0,0)GO TO 111
      GO TO 3400
C
C
110   DO 125 IK=IS,IL
      IF(ROWELL(IK),EQ,0,0) GO TO 125
      ROWELL(IK)=0,0
      JR=IROWNO(IK)
      N=ICOLMK(JR)
      M=ICOLNO(N)
      NS=N+1
      MS=N+M
      DO 1255 K=NS,MS
      IF(ICOLNO(K),NE,I1) GO TO 1255
      IF(COLELL(K),LT,0,0) JCN(JR)=JCN(JR)-1
      COLELL(K)=0,0
      KK(JR)=KK(JR)-1
      K=MS
1255  CONTINUE
125   CONTINUE
      KZ(I1)=0
      IRN(I1)=0
      IPSACT=1
      INROW=I1-1
      WRITE(1,116)INROW
      MRKEY(I1)=OR(KRBFRE,MRKEY(I1))
      IDSC2=1
      GO TO 3400
111   DO 115 IK=IS,IL
      IF(ROWELL(IK),EQ,0,0) GO TO 115
      J1=IROWNO(IK)
      JS=ICOLMK(J1)
      C(J1)=C(J1)-ROWELL(IK)*P(I1)
      ROWELL(IK)=0,0
      JL=ICOLNO(JS)
      NS=JS+1
      NL=JS+JL
      DO 1155 K=NS,NL
      IF(ICOLNO(K),NE,I1) GO TO 1155
      IF(COLELL(K),LT,0,0) JCN(J1)=JCN(J1)-1
      S(K)=COLELL(K)
      COLELL(K)=0,0
      KK(J1)=KK(J1)-1
      JSC(J1)=3
      K=NL
1155  CONTINUE
115 , CONTINUE
      KZ(I1)=0
      IRN(I1)=0
      Q(I1)=P(I1)
      IDS=1
      GO TO 114
113   DO 117 IK=IS,IL
      IF(ROWELL(IK),EQ,0,0) GO TO 117
      J=IROWNO(IK)
      JX=ICOLMK(J)
      JY=ICOLNO(JX)
      JS=JX+1
      JL=JX+JY
      DO 119 K1=JS,JL
      IF(COLELL(K1),EQ,0,0) GO TO 119
      COLELL(K1)=0,0
      S(K1)=0,0
      I2=ICOLNO(K1)
      IX=IROWMK(I2)
      IY=IROWNO(IX)
      IXX=IX+1
      IYY=IX+IY
      DO 121 K2=IXX,IYY
      IF(IROWNO(K2),NE,J) GO TO 121
      IF(ROWELL(K2),LT,0,0) IRN(I2)=IRN(I2)-1
      ROWELL(K2)=0,0
      KZ(I2)=KZ(I2)-1
      K2=IYY
121   CONTINUE
119   CONTINUE
      ISC(J)=1
      KK(J)=0
      U(J)=0,0
```

```
      X(J)=U(J)
      WRITE(1,202) J,X(J)
117   CONTINUE
      KZ(I1)=0
      IRN(I1)=0
114   IPSACT=1
      INROW=I1-1
118   WRITE(1,116) INROW
116   FORMAT(3X,'CONSTRAINT ',I3,X,'IS REDUNDANT')
      IDSC2=1
C
3400  CONTINUE
C
C
C     REPLACING SINGLETON ROW BY PRIMAL BOUND ...
C
C
2000  DO 200 I=2,NROW
      IF(KZ(I).NE.1) GO TO 200
      IF(IRN(I).NE.0) GO TO 200
      IF(P(I).NE.0.0.OR.Q(I).NE.XX) GO TO 200
      JF=0
      NN=IROWMK(I)
      MM=IROWNO(NN)
      INN=NN+1
      IMM=NN+MM
      DO 280 IK=INN,IMM
      IF(ROWELL(IK).LE.0.0) GO TO 280
      J2=IROWNO(IK)
      JF=IK
280   CONTINUE
      IF(JF.EQ.0) GO TO 200
      UP(J2)=RHS(I)/ROWELL(JF)
      IF(UP(J2).GE.U(J2)) GO TO 200
      IPSACT=1
      U(J2)=UP(J2)
      WRITE(1,291) J2,U(J2)
291   FORMAT(3X,'UPPER BOUND X(',I3,') =',F14.3,2X,'II')
      IF(KK(J2).NE.1) GO TO 203
      IF(C(J2).LT.0.0) GO TO 204
      X(J2)=U(J2)
      GO TO 205
204   X(J2)=0.0
205   ISC(J2)=1
      WRITE(1,202) J2,X(J2)
202   FORMAT(3X,'X(',I3,') =',F14.3)
203   ROWELL(JF)=0.0
      KZ(I)=0
      JX=ICOLMK(J2)
      JY=ICOLNO(JX)
      JXX=JX+1
      JYY=JX+JY
      DO 201 K2=JXX,JYY
      IF(ICOLNO(K2).NE.I) GO TO 201
      COLELL(K2)=0.0
      S(K2)=0.0
      KK(J2)=KK(J2)-1
      K2=JYY
201   CONTINUE
200   CONTINUE
C
C
C
1001  IF(IPSACT.EQ.0) GO TO 1008
      IPSACT=0
      IPASS=IPASS+1
      IF(IPART.EQ.1) GO TO 1009
      IF(IPS.EQ.0) GO TO 1009
      IPS=0
      DO 1035 J3=1,NCOL
      IF(ISC(J3).NE.2) GO TO 1035
      IF(KK(J3).EQ.0) GO TO 1035
      N=ICOLMK(J3)
      M=ICOLNO(N)
      NS=N+1
      MS=N+M
      DO 1036 K=NS,MS
      IF(SN(K).EQ.0.0)GO TO 1036
      IF(COLELL(K).LT.0.0) JCN(J3)=JCN(J3)-1
      COLELL(K)=0.0
      KK(J3)=0
      I=ICOLNO(K)
      NN=IROWMK(I)
      MM=IROWNO(NN)
      INN=NN+1
      IMM=NN+MM
      DO 1336 IK=INN,IMM
      IF(IROWNO(IK).NE.J3) GO TO 1336
      IF(ROWELL(IK).LT.0.0) IRN(I)=IRN(I)-1
      ROWELL(IK)=0.0
      KZ(I)=KZ(I)-1
      IK=IMM
1336  CONTINUE
1036  CONTINUE
1035  CONTINUE
      GO TO 1009
1008  IF(IPASS.GT.1) GO TO 1010
      IPSACT=0
      IPASS=IPASS+1
      GO TO 1009
```

```
1010  IF(IPHASE,EQ,0) GO TO 1012
      IF(IPART,EQ,1)GO TO 999
      IF(IDS,EQ,0) GO TO 999
      DO 605 J=1,NCOL
      JSC(J)=0
      IF(ISC(J).EQ.1) GO TO 605
      N=ICOLMK(J)
      M=ICOLNO(N)
      NS=N+1
      MS=N+M
      DO 606 K=NS,MS
      I=ICOLNO(K)
      IF(ISC(J).NE.2) GO TO 608
      ISC(J)=0
      IF(SN(K).EQ.0.0) GO TO 608
      COLELL(K)=SN(K)
      IF(COLELL(K).LT.0.0) JCN(J)=JCN(J)+1
      KK(J)=KK(J)+1
      IB=IROWMK(I)
      IG=IROWNO(IB)
      JB=IB+1
      JG=IB+IG
      DO 6888 ID=JB,JG
      IF(IROWNO(ID),NE,J) GO TO 6888
      ROWELL(ID)=SN(K)
      IF(ROWELL(ID).LT.0.0) IRN(I)=IRN(I)+1
      KZ(I)=KZ(I)+1
      ID=JG
6888  CONTINUE
      GO TO 609
608   IF(S(K),EQ.0.0) GO TO 606
      COLELL(K)=S(K)
      IF(COLELL(K).LT.0.0) JCN(J)=JCN(J)+1
      KK(J)=KK(J)+1
      IX=IROWMK(I)
      IY=IROWNO(IX)
      JS=IX+1
      JL=IX+IY
      DO 6088 IZ=JS,JL
      IF(IROWNO(IZ),NE,J) GO TO 6088
      ROWELL(IK)=S(K)
      IF(ROWELL(IK).LT.0.0) IRN(I)=IRN(I)+1
      KZ(I)=KZ(I)+1
      IZ=JL
6088  CONTINUE
609   C(J)=SC(J)
606   CONTINUE
605   CONTINUE
      DO 607 I=2,NROW
      P(I)=0.0
      Q(I)=XX
607   CONTINUE
      IPART=1
      WRITE(1,2004)
2004  FORMAT(3X,'PART  B')
      IPHASE=0
      WRITE(1,2002)
      IPASS=0
      IPSACT=0
      IPASS=IPASS+1
      DO 1113 J=1,NCOL
      JSC(J)=0
1113  CONTINUE
      GO TO 1009
1012  IPHASE=1
      WRITE(1,2005)
2005  FORMAT(3X,'PHASE  2')
      IPASS=1
      IPSACT=0
      GO TO 1009
999   RETURN
      END
C
C
C
```

PREPROCESSING REDUCTION PROCEDURE

```
      SUBROUTINE USER
$INSERT SCICON>S>PDPPARAMS
$INSERT SCICON>S>PDPMCKEY
$INSERT SCICON>S>PDPITER
$INSERT SCICON>S>PDPBITS
$INSERT SCICON>S>PDPUSEFUL
$INSERT SCICON>S>PDPMATRIX
$INSERT SCICON>S>PDPPOOL
$INSERT SCICON>S>PDPBETA
      REAL*8 RHS(512)
      REAL*8 COLELL(2048)
      REAL*8 ROWELL(2048)
      REAL*8 W(512)
      REAL*8 WP(512)
      REAL*8 U(512)
      REAL*8 UP(512)
      REAL*8 RL(512)
      REAL*8 RLC(512)
      REAL*8 X(512)
      REAL*8 DFE
      REAL*8 DFR
      REAL*8 DFU
      REAL*8 UT
      REAL*8 WT
      REAL*8 ELM1
      REAL*8 ELM2
      INTEGER*2 ICOLNO(2048)
      INTEGER*2 ICOLMK(512)
      INTEGER*2 IROWNO(2048)
      INTEGER*2 IROWMK(512)
      INTEGER*2 KK(512)
      INTEGER*2 KZ(512)
      INTEGER*2 KZN(512)
      INTEGER*2 JZN(512)
      INTEGER*2 IOCRU(512)
      INTEGER*2 IOCRL(512)
      INTEGER*2 IV(512)
      INTEGER*2 IR(512)
      COMMON/COMA1/COLELL,ROWELL
      COMMON/COMB1/ICOLNO,IROWNO
      COMMON/COMC1/ICOLMK,IROWMK,KK,KZ,KZN,JZN,IOCRU,IOCRL,IV,IR
      COMMON/COMD1/W,WP,U,UP,X
      COMMON/COME1/RL,RLC,RHS
      WRITE(1,998) NROW,NSEQ
998   FORMAT(2X,I3,3X,I3)
      XX=1000000.0
      NNROW=NROW+1
      IPASS=1
      N=1
      K=1
      DO 1500 JSEQ=NNROW,NSEQ
      J=JSEQ-NROW
      ICOLMK(J)=K
      KLMEL=MSMEL(JSEQ)+MSKMEB(JSEQ)
      LLMEL=MSMEL(JSEQ+1)
      L=0
      DO 1600 ILMEL=KLMEL,LLMEL
      IROW=MRWME(ILMEL)
      IF(IROW.EQ.1) GO TO 1600
      IPOOL=MPTME(ILMEL)
      N=N+1
      L=L+1
      COLELL(N)=POOL(IPOOL)
      ICOLNO(N)=IROW
      KK(J)=KK(J)+1
      KZ(IROW)=KZ(IROW)+1
      IF(COLELL(N).GT.0.1E-8) GO TO 1600
      JZN(J)=JZN(J)+1
      KZN(IROW)=KZN(IROW)+1
1600  CONTINUE
      ICOLNO(K)=L
      IF(AND(MCKEY(JSEQ),KCBUBC).EQ.0) GO TO 1650
      IPOOL=MPTME(KLMEL-1)
      COLELL(K)=POOL(IPOOL)
      COLELL(K)=-COLELL(K)
1650  IPOOL=MRWME(KLMEL-1)
      IF(IPOOL.EQ.KPTPLI) GO TO 1660
      U(J)=POOL(IPOOL)
      UP(J)=U(J)
1660  K=K+L+1
      N=N+1
1500  CONTINUE
C
C
      LL=1
      K=1
      DO 1400 I=2,NROW
      IR(I)=I
      RHS(I)=BETA(I)
      IROWMK(I)=K
      IC=0
      DO 1401 J=1,NCOL
      N=ICOLMK(J)
      M=ICOLNO(N)
      IN=N+1
      IM=N+M
      DO 1402 L=IN,IM
      IF(ICOLNO(L).NE.I) GO TO 1402
      IC=IC+1
      LL=LL+1
```

```
      ROWELL(LL)=COLELL(L)
      IROWNO(LL)=J
      L=IM
      IF(ROWELL(LL).GT.0.0) GO TO 1402
      RL(I)=RL(I)+ROWELL(LL)*U(J)
1402  CONTINUE
1401  CONTINUE
      IROWNO(K)=IC
      K=K+IC+1
      LL=LL+1
1400  CONTINUE
      JFX=1
      IDR=1
C
C
C
1200  WRITE(1,2000) IPASS
2000  FORMAT(3X,'PASS ',I2)
C
C
C
      DO 1 J=1,NCOL
      IF(KK(J).EQ.0) GO TO 1
      N=ICOLMK(J)
      M=ICOLNO(N)
      IN=N+1
      IM=N+M
C
C
      IF(JZN(J).EQ.0.AND.COLELL(N).LE.0.1E-8) GO TO 705
C
C
900   DO 901 K=IN,IM
      IF(COLELL(K).EQ.0.0) GO TO 901
      I=ICOLNO(K)
      IF(COLELL(K).LT.0.0) GO TO 902
      DFE=RHS(I)-RL(I)
      IF(COLELL(K).GT.DFE) GO TO 705
      GO TO 901
902   RLM=RL(I)-COLELL(K)*U(J)
      DFE=RHS(I)-RLM
      ELM=-COLELL(K)
      IF(ELM.GT.DFE) GO TO 706
      GO TO 901
C
C
C
595   FORMAT(3X,'LOWER BOUND (',I3,') TIGHTENED TO',F14.3)
590   FORMAT(3X,'UPPER BOUND (',I3,') TIGHTENED TO',F14.3)
585   FORMAT(3X,'X(',I3,') EXTRANEOUS')
580   FORMAT(3X,'X(',I3,')=',F14.3)
570   FORMAT(3X,'CONSTRAINT(',I3,') REDUNDANT')
C
C
C
705   WRITE(1,585) J
      JSEQ=NROW+J
      MCKEY(JSEQ)=AND(MCKEY(JSEQ),KODART)
      X(J)=W(J)
      GO TO 707
706   X(J)=U(J)
      WRITE(1,580) J,X(J)
      JFX=1
707   DO 714 K2=IN,IM
      I2=ICOLNO(K2)
      IF(COLELL(K2).EQ.0.0) GO TO 714
      RL(I2)=RL(I2)-COLELL(K2)*X(J)
      RHS(I2)=RHS(I2)-COLELL(K2)*X(J)
      BETA(I2)=BETA(I2)=COLELL(K2)*X(J)
      COLELL(K2)=0.0
      N=IROWMK(I2)
      M=IROWNO(N)
      JN=N+1
      JM=N+M
      DO 715 K3=JN,JM
      IF(IROWNO(K3).NE.J) GO TO 715
      IF(ROWELL(K3).LT.0.0) KZN(I2)=KZN(I2)-1
      ROWELL(K3)=0.0
      KZ(I2)=KZ(I2)-1
      K3=JM
715   CONTINUE
714   CONTINUE
      IPHASE=1
      IDR=1
      KK(J)=0
      K=IM
C
C
C
901   CONTINUE
C
C
C
C
C
C
      IF(KK(J).EQ.0) GO TO 1
      N=ICOLMK(J)
      M=ICOLNO(N)
      IN=N+1
```

```
      IM=N+M
      DO 3 K=IN,IM
      IF(COLELL(K),EQ,0,0) GO TO 3
      I=ICOLNO(K)
C
C
      IF(COLELL(K),GT,0,0) GO TO 4
C
C
      WT=W(J)+(RHS(I)-RL(I))/COLELL(K)
      IF(WT,LE,WP(J)) GO TO 3
      WP(J)=INT(WT)+1
      IOCRL(J)=1
      GO TO 3
C
C
4     UT=W(J)+(RHS(I)-RL(I))/COLELL(K)
      IF(UT,GE,UP(J)) GO TO 3
      UP(J)=INT(UT)
      IOCRU(J)=1
C
C
3     CONTINUE
C
C
      IF(IOCRL(J),EQ,0) GO TO 5
      W(J)=WP(J)
      WRITE(1,595) J,W(J)
      IOCRL(J)=0
      IPHASE=1
      IDR=1
5     IF(IOCRU(J),EQ,0) GO TO 903
      U(J)=UP(J)
      WRITE(1,590) J,U(J)
      IOCRU(J)=0
      IPHASE=1
      IDR=1
C
C
C
903   DO 2 K=IN,IM
      IF(COLELL(K),EQ,0,0) GO TO 2
      I=ICOLNO(K)
      IF(COLELL(K),GT,0,0) GO TO 7
      RLC(I)=RLC(I)+COLELL(K)*U(J)
      GO TO 2
7     RLC(I)=RLC(I)+COLELL(K)*W(J)
2     CONTINUE
C
C
C
1     CONTINUE
C
C
C
      DO 6 I=2,NROW
      RL(I)=RLC(I)
      RLC(I)=0,0
6     CONTINUE
C
C
C
      IF(JFX,EQ,0) GO TO 103
      JFX=0
      DO 101 I1=2,NROW
      AMAX=0,0
      DO 102 I2=2,NROW
      IF(IR(I2),EQ,0) GO TO 102
      IF(AMAX,GE,RHS(I2)) GO TO 102
      AMAX=RHS(I2)
      I=I2
102   CONTINUE
      IV(I1)=I
      IR(I)=0
101   CONTINUE
C
C
103   IF(IDR,EQ,0) GO TO 999
      DO 104 IG1=2,NROW
      IR(IG1)=IG1
      I=IV(IG1)
      IF(KZ(I),EQ,0) GO TO 104
      IF(KZ(I),EQ,KZN(I)) GO TO 108
C
C
C
      IF(IG1,EQ,NROW) GO TO 104
      IRL=IG1+1
      DO 105 IG2=IRL,NROW
      I2=IV(IG2)
      IF(KZ(I2),EQ,0) GO TO 105
C
C
      DFR=RHS(I)-RHS(I2)
      DFU=0,0
      DO 106 J=1,NCOL
      IF(KK(J),EQ,0,0) GO TO 106
      DFE=0,0
      ELM1=0,0
      ELM2=0,0
```

```
      IX=ICOLMK(J)
      IY=ICOLNO(IX)
      IS=IX+1
      IL=IX+IY
      DO 107 K=IS,IL
      I3=ICOLNO(K)
      IF(I3.EQ.I) ELM1=COLELL(K)
      IF(I3.EQ.I2) ELM2=COLELL(K)
107   CONTINUE
      DFE=ELM1-ELM2
      IF(DFE.LE.0.0) GO TO 106
      DFU=DFU+DFE*U(J)
106   CONTINUE
C
C
      IF(DFR.LT.DFU) GO TO 105
      IG2=NROW
C
C
108   IPHASE=1
      INR=I-1
      WRITE(1,570) INR
      MRKEY(I)=OR(KRBFRE,MRKEY(I))
      JX=IROWMK(I)
      JY=IROWNO(JX)
      JS=JX+1
      JL=JX+JY
      DO 109 K1=JS,JL
      IF(ROWELL(K1).EQ.0.0) GO TO 109
      J1=IROWNO(K1)
      IF(ROWELL(K1).LT.0.0) JZN(J1)=JZN(J1)-1
      ROWELL(K1)=0.0
      IXX=ICOLMK(J1)
      IYY=ICOLNO(IXX)
      ISS=IXX+1
      ILL=IXX+IYY
      DO 110 K2=ISS,ILL
      IF(ICOLNO(K2).NE.I) GO TO 110
      COLELL(K2)=0.0
      KK(J1)=KK(J1)-1
      K2=ILL
110   CONTINUE
109   CONTINUE
      KZ(I)=0
C
C
105   CONTINUE
104   CONTINUE
C
      IF(IPHASE.EQ.0) GO TO 999
      IPHASE=0
      IPASS=IPASS+1
      IOR=0
      GO TO 1000
C
C
C
C
999   RETURN
      END
C
C
C
```

REFERENCES AND *BIBLIOGRAPHY

* Austin, L. M., and Michael, E. H. (1983), "A Bounded Dual (All Integer) Integer Programming Algorithm with objective cut, " Naval Res. Logtcs. Quarterly, Vol. 30, pp.271-281.

*Austin, L. M., and Michael, E. H. (1985), "An Advance Start Algorithm For All-Integer Programming, " Comput. & Ops. Res. Vol. 12, No. 3, pp.301-309.

Ahmed, A. N. (1977), "Application of Linear Programming to Transportation Problem in Iraq", MSc Thesis, Baghdad University, Iraq.

*Ahmed, A. N (1984), "A Modified Reduction Procedure for Linear Programming Problems," Working Paper, Management Studies Department, Loughborough University.

*Ahmed, A.N. (1985), "A Reduction Procedure for Integer Programming Problems," Working Paper, Management Studies Department, Loughborough University.

*Ahmed, A.N. (1985), "Size Reduction of Linear Programs," Working Paper, Management Studies Department, Loughborough University.

Balas, E. (1962), "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Ops. Res., Vol. 13, No. 4, pp.517-546.

Balinsky, M. L. (1961), "An Algorithm for Finding All Vertices of Convex Polyhedral Set," Journal of the Society for Industrial and Applied Mathematics, Vol. 9, No. 1, pp.72-88.

Benders, J. F. (1962), "Partitioning Procedures for Solving Mixed Variables Programming Problems, Numerische Mehtmatik, Vol. 4, pp.238-252.

Beale, E. M. L. and Forrest, J. J. H. (1976), "Global Optimisation Using Special Ordered Sets", Mathematical Programming, No. 10, pp.52-69.

Beale, E. M. L. and Forrest, J. J. H. (1978), "Global Optimisation as an Extension of Integer Programming," in Towards Global Optimisation 2, eds., L. C. W. Dixon and G. P. Szego, North-Holland, Holland.

Beale, E. M. L. and Tomlin, J. A. (1970), "Special Facilities in a General Mathematical Programming System for Non-convex Problems Using Ordered Sets of Variables," in Proceeding of the Fifth International Conference on Operations Research, ed. J. Lawrence, pp.447-454.

Bixby, R. E. (1981), "Hidden Structure in Linear Programs," in Computer Assisted Analysis and Mole Simplification, ed. H. Greenberg and J. Maybee, Academic Press, New York, pp.327-360.

Bixby, R. E. and Cunningham, W. H. (1980), "Converting Linear Programs to Network Problems," Maths. of Ops. Research., Vol.5, pp.321-357.

Boneh, A. (1981), "Minimal Representation of Nonlinear Inequalities by a Probabilistics Set Covering Problem Equivalence", Technical Report TRCS8-05, Computer Science Department, University of California, Santa Barbara.

Boneh, A. (1983), "PREDUCE - A Probabilistic Algorithm Identifying Redundancy by a Random Feasible Point Generator (RFPG)," in Redundancy in Mathematical Programming ed. M. H. Karwan, V. Lotfi, J. Telgen and S. Zionts, Springer-Verlag.

Boneh, A. and Golan, A. (1979), "Constraints Redundancy and Feasible Region Boundedness by Random Feasible Points Generator," paper presented at EURO III, Amsterdam.

Boot, J. C. G. (1962), "On Trivial and Binding Constraints in Programming Problems," Management Science, Col. 8, No. 4, pp.419-441.

*Boot, J. C. G. (1963), Quadratic Programming, Amsterdam, North Holland.

Bradley, G., Brown, G. and Galatas, P (1980), "An Interactive System to Analyse Large-scale Optimisation Models", Naval Postgraduate School, Technical Report NPS52-80-005.

*Bradley, G., Brown, G. and Graves, G. (1977), "Design and Implementation of Large-scale Primal Transshipment Algorithms," Management Science., Vol. 24, No. 1.

*Bradley, G., Borwn, G. and Graves, G. (1977), "Preprocessing Large-scale Optimisation Models," in Redundancy in Mathematical Programming, ed. M. H. Karwan, V. Lotfi, J. Telgen and S. Zionts, Springer-Verlag.

Bradley, G., Brown, G. and Graves, G. (1983), "Structural Redundancy in Large-Scale Optimisation Models", in Redundancy in Mathematical Programming, ed. M. H. Karwan, V, Lotfi, J. Telgen and S. Zionts, Springer-Verlag.

Brearley, A. L., Mitra, G and Williams H. P. (1975), "Analysis of Mathematical Programming Models Prior to Applying the Simplex Algorithm," Mathematical Programming Vol. 8, pp.54-83.

Brown, G. and Thomen, D. (1980), "Automatic Identification of Generalised Upper Bounds in Large-scale Optimisation Models," Management Science, Vol. 26, No. 11, 00.1166-1184.

Brown, G. and Wright, W. (1980), "Automatic Identification of Network Rows in Large-scale Optimisation Models," in Proceeding of the Symposium of Computer Associated Analysis and Model Simplification, Boulder.

Charnes, A. and Cooper, W. W. (1961), Management Models and Industrial Applications of Linear Programming, Vol. I and II, John Wiley & Sons, New York

Charnes, A. Cooper, W. W. and Farr, D. (1953), "Linear Programming and Profit Performance Scheduling for a Manufacturing Firm," Journal of the ORSA, Vol. 1.

Charnes, A., Cooper, W. W. and Thompson, G. L. (1962), "Some Properties of Redundant Constraints and Extraneous Variables in Direct and Dual Linear Programming Problems," Ops. Res., Vol.10, No. 5, pp.711-723.

*Cheng, M. C. (1980), "New Criteria for the Simplex Algorithm," Mathematical Programming, Vol. 19, pp.230-236.

Chvatal, V. (1984), Linear Programming, W. H. Freeman and Company, New York.

Cooper, Dale O. (1962), "Techniques for Reducing the Size of Process Plant Models for Linear Programming," Bonner and Moore Associates, Houston.

Crowder, H., Johnson, E. L. and Padberg, M. W. (1983), "Solving Large-scale Zero-One Linear Programming Problems," Ops. Research., Vol. 31, No. 4.

*Dantzig, G. B. (1948), "Programming in a Linear Structure," Comptroller USAF, Washington, D.C.

Dantzig, G. B. (1955), "Upper Bounds, Secondary Constraints, and Block Trianguarity," Econometrica, Vol. 23, No. 2, pp.174-183.

*Dantzig, G. B. (1963), Linear Programming and Extensions, Princeton: Princeton University Press.

Dantzig, G. B. and Wolfe, P. (1960), "The Decomposition Principle for Linear Programs," Ops. Res., Vol. 8, pp.101-111.

Dyer, M. E. and Proll, L. G. (1977), "Vertex Enumeration in Convex Polyhedra- a Comparative Computational Study," in T. B. Boffey, ed, Proceeding of the CP77 Combinatorial Programming Conference.

Eckhardt, U. (1971), "Redundant Ungleichungen bei linearen Ungleichungssytemnr Unternehmenforschung, Vol. 12, pp279-286.

*Ferguson, R. O. and Sargent, L. F. (1958), Linear Programming: Fundementals and Applications, McGraw-Hill Book Company Inc, New York.

Forrest, J. J. H. and Tomlin, J. A. (1972), "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product-Form Simplex Method," Mathematical Programming, Vol. 2, pp.263-278.

Forrest, J. J. H., Tomlin, J. A. and Hirst, J. P. H. (1974), "Practical Solutions of Large Mixed Integer Programming Problems with UMPIRE," Management Science, Col. 20, pp.736-773.

*Fourier, J. B. J. (1926), "Solution d'une question particuliere du calcul des inequalities."

Gal, T. (1975), "Redundancy Reudction in the Restrictions Set Given in the Form of Linear Inequalities," Progress in Cybernetics and Systems Research, Vol. 1, pp.177-179.

*Gal, T. (1977), "A General Method for Determining the Set of All Efficient Solutions to a Linear Vectormaximum Problem," European Journal of Operational Research, Vol. 1, pp.307-329.

Gal, T. (1978), "Redundancy in Systems of Linear Inequalities Revisited," Discussion Paper No. 19, Fernuniversitat, Hagen.

*Gal, T. (1979), "Postoptimal Analysis, Parametric Programming and Related Topics, McGraw-Hill.

Gal, T. (1983), "Another Method for Determining Redundant Constraints," in Redundancy in Mathematical Programming, ed. M. H. Karwan, V. Lotfi, J. Telgen and S. Zionts, Springer-Verlag.

*Gal, T., and Leberling, H. (1977), "Redundant Objective Functions in Linear Vecotrmaximum Problems and Their Determination," European Journal Of Operational Research, Vol. 1, pp.176-184.

Gale, D. (1960) The Theory of Linear Economic Models, New York:Mc Graw-Hill.

*Gale, G. (1979), "How to Solve Linear Inequalities" American Mathematical Monthly, Vol. 76, pp.589.599.

Garfinkel, R. S. and Nemhauser, G. L. (1972), Integer Programming, John Wiley.

Gauthier, J. M. and Ribiere, G. (1977), "Experiments in Mixed-Integer Linear Programming Pseudo-Costs," Mathematical Programming, Vol. 12, pp.26-47.

Gomory, R. E. (1958), "Essentials of an Algorithm for Integer Solutions to Linear Programs, Bull. American Mathematical Society, Vol. 64, No. 5, pp.275-278.

Graves, G. and McBride ,R. (1976), "The Factorisation Approach to Large-Scale Linear Programming," Mathematical Programming, Vol. 10, No. 1, p.91.

Graves, G. and Van Troy, T. (1979), "Decomposition for Large-scale Linear and Mixed Integer Linear Programming," UCLA Technical Report.

Graves, R. and Wolfe, P. (1963), Recent Advances in Mathematical Programming, New York:McGraw-Hill.

Greenberg, H. (1975), "An Algorithm for Determining Redundant Inequalities and All Solutions to Convex Polyhedral," Numerische Mathematika, Vol. 24, pp.19-26.

Hoffman, A. J. (1955), "How to Solve a Linear Programming Problem," in H. A. Antosiewicz, ed. pp.397-423.

Holm, S. and Klein, D. (1975), "Size Reduction of Linear Programs with Special Structure," Working Paper, Odense University.

Holm, S. and Klein, D. (1976), "Identification of Nonbinding Constraints and Zero Variables in Linear Programming." Ops. Res. Verfahren, Vol. 25, No. 1, pp.58-65.

Holm, S. and Klein, D (1979), "Size Reduction of Linear Programs Using Bounds on Problem Variables," Working Paper, Florida International University.

*Jackson, R. and O'Neill, R. (1983), Mixed Integer Programming in Mathematical Systems, ORSA/committee on Algorithms Publication. Special Issue.

Jarvis, J. J. and Bazaraa, M. S. (1977), Linear Programming and Network, John Wiley & Sons Inc., New York.

Karwan, M. H., Lotfi, V., Telgen, J. and Zionts, S. (1983), Redundancy in Mathematical Programming, Lecture Notes in Economics and Mathematical Systems, No. 206, Springer-Verlag.

Kalan, J. E. (1977), "Aspects of Large-scale In-core Linear Programming", in Proceeding of the ACM conference, Chicago University Press.

Kelly, J. E. (1963), "The Cutting Plane Method for Solving Convex Programs," J, Soc. Ind. Appl. Math., Vol. 8, No. 4, pp.703-712.

*Land, A. H. and Powell, S. (1981), "A Survey of Available Computer Codes to Solve Integer Linear Programming Problems," Rapport de recherche No. 81-09, Montreal University, Canada.

Lisy, J. (1971), in (Ekonomiko Matematicky Obzor, Vol. 7, No. 3, pp.285-298)

Lotfi, V. (1981), A Study of Size-Reduction Techniques in Linear Programming, PhD Dissertation, State University of New York, Buffalo.

Luenberger, D. G. (1973), Introduction to Linear and Non-Linear Programming, Addison-Wesley.

Mattheiss, T. H.(1973), "An Algorithm for Determining Irrelevant Constraints and All Vertices in Systems of Linear Inequalities," Ops. Res., Vol. 21, No. 1, pp.247-260.

Mattheiss, T. H. (1983), "A Method for Finding Redundant Constraints of a System of Linear Inequalities," in Redundancy in Mathematical Programming, ed. M. H. Karwan, V. Lotfi, J Telgen and S. Zionts, Springer-Verlag.

Mattheiss, T. H. and Rubin, D. S. (1980), "A Survey and Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets," Mathematics of Ops. Res., Vol. 5, No. 2, pp.167-185.

Mattheiss, T. H. and Schmidt, B. K. (1980), "Computational Results on an Algorithm for Finding All Vertices of a Polytope," Mathematical Programming, Vol. 18, pp.308-329.

McBride, R. (1973), Factorisation in Large-Scale Linear Programming, PhD Dissertation, UCLA.

Meyerman, B. G. (1979), "Some Results of a Reduction Algorithm for Linear Programming Problems," Department of Ops. Research. Groningen University.

*Musalem, S. (1979), Converting Linear Models to Network Models, PhD Dissertation, UCLA.

Motzkin, T. S. (1936), "Beitage Sun Theorie Ider Linearen Ungleichringen," PhD Dissertation, University of Zurich.

Motzkin, T. S, Raiffa, H. Thompson, G. L. and Thrall, R. M. (1953), "The Double Description Method," in Contribution to the Theory of Games, ed. by Kuhn, H. W. and Tucker, A. W., Vol. 2, Annals of Mathematics Studies, No. 28.

Rubin, D. S. (1972), "Redundant Constraints and Extraneous Variables in Integer Programs," Management Science, Vol. 18, No. 7, pp.423-427.

Rubin, D. S. (1983), "Redundant Constraints in Linear Programs," in Redundancy and Mathematical Programming, ed. M. H. Karwan, V. Lotfi, J. Telgen and S. Zionts, Springer-Verlag.

SCICONIC VM (1983), Scicon Services Ltd, Milton Keynes, England.

Sethi, A. P. and Thompson, G. L. (1983), "The Non-Candidate Constraint Method for Reducing the Size of a Linear Program," in Redundancy in Mathematical Programming, ed. Karwan, M. H., Lotfi, V, Telgen, J. and Zionts, S.

Sethi, A. P. and Thompson, G. L. (1984), "The Pivot and Probe Algorithm for Solving a Linear Program," Mathematical Programming, Vol. 29, pp. 219-233.

Shefi, A. (1969), Reduction of Linear Inequality Constraints and Determination of All Possible Extreme Points, PhD Dissertation, Stanford University.

Sherman, R. F. (1977), "A Counterexample to Greenberg's Algorithm for Solving Linear Inequalities," Numerische Mathematik, Vol. 27, pp. 491-492.

* Spronk, J. and Telgen, J. (1979), "A Note on Multiple Objective Programming and Redundancy," Report No. 7906, Centre for Research in Business Economics , Erasmus University, Rotterdam.

* Telgen, J. (1977), "On Redundancy in Systems of Linear Inequalities," Report 7718, Econometric Institute, Erasmus University, Rotterdam.

* Telgen, J. (1977), "Redundant and Nonbinding Constraints in Linear Programming Problems," Report 7720, Econometric Institute, Erasmus University, Rotterdam.

Telgen, J. (1979), "On Llewellyn's Rules to Identify Redundant Constraints in Systems of Linear Equalities," Zeitschrift for Ops. Res., Vol. 23, pp.197-206.

Telgen, J. (1980), "Identifying Redundant Constraints and Implicit Equalities in Systems of Linear Constraints," Working Paper 90, College of Business Administration, University of Tennessee.

Telgen, J. (1981), "Minimal Representation of Convex Polyhedral Sets," Journal of Optimisation, Theory and Applications.

Telgen ,J. (1981), Redundancy and Linear Programs, Mathematisch Centrum, Amsteradam.

Telgen, J. (1983), "Identifying Reundancy in Systemsof Linear Constraints," in Redundancy in Mathematical Programming ed. M.H. Karwan, V. Lotfi, J. Telgen and S. Zionts, Springer-Verlag.

Thompson, G. L., Tonge, F. M. and Zionts, S. (1966), "Techniques for Removing Nonbinding Constraints and Extraneous Variables from Linear Programming Problems," Management Science, Vol. 12, No. 7, pp.588-608.

Tischer, H. j. (1968), Mathematische Verfahren Zur Reduzierung der Zeilenund Spaltenzahl Linearer Opimierrungsaufgaben, Dissertation, Zentralinstitut fur Fertigungstechnik des mMaschinenbaues, Karl Marx Stad.

* Tomlin, J. A. and Welch, J. S. (1983), "A Pathological Case in the Reduction of Linear Programs," Opes. Res., Letters, Vol. 2, No. 2.

* Tomlin, J. A. and Welch, J. S. (1983), "Formal Optimisation of Some Reduced Linear Programming Problem," Mathematical Programming, Vol. 27, pp.232-240.

* Williams, H. P. (1973), "Simplifying Linear Programming Problems," Research Report, No. 73-2, University of Sussex.

* Williams, H. P. (1975), "Further Simplification of Linear Programming Problems," Research Report 75-1, University of Sussex.

Williams, H. P. (1978), Modelling in Mathematical Programming, J. Wiley, New York.

Williams, H. P (1983), "A Reduction Procedure for Linear and Integer Problems,' in Redundancy in Mathematical Programming, ed. M. H. Karwan V. Lotfi, J. Telgen, S. Zionts, Springer-Verlag,

Williams, N. (1967), Linear and Non-linear Programming in Industry. ed. by S. Vadja (A Series of 'Topics in Operational Research'). Pitman.

Wilson, J. M. (1983), "Removing Certain Redundancies from a Set of (0-1) Linear Inequalities," Management Studies Department, Loughborough University.

Wolf, P. (1955), "Reduction of Systems of Linear Relations (abstract)," in H. A. Antosiewicz, pp.449-451.

Wright, W. (1980), Automatic Identification of Network Rows in Large-Scale Optimisation Models," MSc Thesis, Naval Postgraduate School.

Zeleny, M. (1974), Linear Multiobjective Programming, Lecture Notes in Economics and Mathematical Systems, No. 95, Springer-Verlag.

Zionts, S. (1965), Size Reduction Techniques of Linear Programming and Their Application, PhD Dissertation, Carnegie Institute of Technology.

* Zionts, S. (1960), "Toward a Unifying Theory for Integer Linear Programming," Ops. Res. Vol. 17, No. 2, pp.359-367.

Zionts, S. and Wallenius, J. (1976), "An Interactive Programming Method for Solving the Multiple Criteria Problem," Management Science, Vol. 22, No. 6, pp.652-663.

Zionts, S. and Wallenius, J, (1980), "Identifying Efficient Vectors: Some Theory and Computational Results," Ops. Res. Vol, 28, pp.785-793.

Zionts, S. and Wallenius, J. (1983), "A Method for Determining Redundant Constraints and Extraneous Variables in Linear Programming Problems," Zionts, S. and Wallenius, J. (1983), "A Method for Determining Redundant Constraints and Extraneous Variables in Linear Programming Problems," in Redundancy in Mathematical Programming, ed. M. H. Karwan, V. Lotfi, J. Telgen and S. Zionts, Springer-Verlag.