TR/07/92 June 1992
Revised Nov. 1993

Design, Implementation and Testing of an Integrated Branch and Bound Algorithm for Piecewise Linear and Discrete Programming Problems within an LP Framework.

By

M T Hajian
G Mitra

Design, Implementation and Testing of an Integrated Branch and Bound Algorithm for Piecewise Linear and Discrete Programming Problems within an LP Framework.

By

M. T. Hajian, Brunel, The University of West London
G. Mitra, Brunel, The University of West London

CONTENTS.

0. Abstract

A number of discrete variable representations are well accepted and find regular use within LP systems. These are Binary variables, General Integer variables, Variable Upper Bounds or Semi Continuous variables, Special Ordered Sets of type One and type Two. The FortLP system has been extended to include these representations. A Branch and Bound algorithm is designed in which the choice of sub-problems and branching variables are kept general. This provides considerable scope of experimentation with tree development heuristics and the tree search can then be guided by search parameters specified by user subroutines. The data structures for representing the variables and the definition of the branch and bound tree are described. The results of experimental investigation for a few test problems are reported.

Introduction and Background.

Alternative ways of representing non-linear programming problems involving variable separable function as discrete programming problems are well known since the 60s, Miller (1963), Falk and Soland (1968), Beale and Tomlin (1969), but not many general non-linear and discrete programming systems are in industrial use. Currently known systems are: SCICONIC, OSL, XPRESSMP, LAMPS and there are a number of systems with only zero-one and integer facilities, these include ZOOM, MPSX, FMPS, LINDO, MPSIII, and MINTO.

Early research results concerning tree search methods applied to such problems were reported in the 60s and early 70s, Benichou (1971), Mitra (1973), Forrest et al (1974). All the above mathematical systems use tree search algorithm in which the original problem (relaxed as an LP) as well as a waiting node or a sub-problem are solved by the Sparse (Revised) Simplex method.

During the 80s most of the research effort has been directed towards, (i) Preprocessed integer programming model to generate constraints which produce deep cuts, Van Roy and Wolsey (1984), (ii) use of such constraints in the root node and also subsequent tree search.

In this report we set out our algorithmic and software design consideration for constructing a general discrete programming systems. This work is a part of our ongoing research in Large Scale Linear and Discrete Optimisation system. It is built around the FortLP which is a sparse simplex solver for large linear programs.

The objectives of this work are summarized below:

(a) to develop a family of Branch and Bound algorithms to solve discrete programming problems in their most general form,
(b) to introduce a general Cutting plane method within the Branch and Bound tree search framework, and
(c) to use this system as an experimental tool to investigate new ideas of constraint generation.

In this report, the development of a generic Branch And Bound algorithm for solving discrete programming problems, namely part (a) above is described and some experimental results are presented.

The contents of this report are organised in the following way. In section 2 the classes of discrete variables and set variables are reviewed. In section 3 the mathematical representation of a general discrete problem is introduced. In section 4 and 5 a general Branch and Bound approach which takes into account different classes of variables and set variables together with alternative rules applied in Branch and Bound are outlined.

In section 6 we discuss the data structure and implementation issues of internal representation of different types of variables and sets. The results of our experiments with a collection of some well known test problems are given in section 7.

function g(y) may be represented as:

$$g(y) = g(\hat{y}_1)x_1 + g(\hat{y}_2)x_2 + \ldots + g(\hat{y}_k)x_k \quad (21)$$

where:

$$\hat{y}_1 x_1 + \hat{y}_2 x_2 + \ldots + \hat{y}_K X_K - y = 0,\ y \geq 0 \quad (22)$$

$$x_1 + x_2 + \ldots + x_k = 1,\ x_k \geq 0,\ k=1,\ldots,K \quad (23)$$

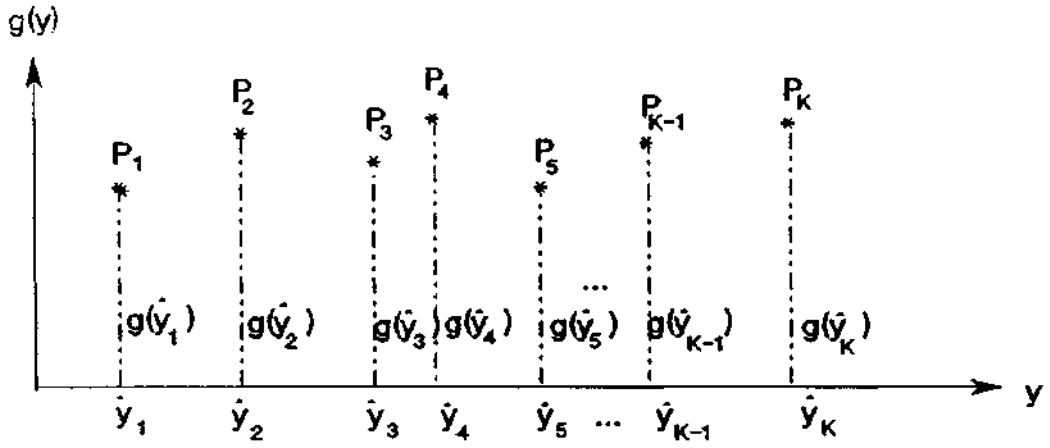


Fig.1

The discrete function can take only one of the K possible values weighted by the variable x_k. This requirement can be easily expressed by adding the condition,

$$x_k \in \{0,1\},\ k=1,\ldots,K.$$

Such a group of variables is called a Special Ordered Set of Type One as only one variable in the group can take a value different from their (lower bound) zero value in a valid solution.

(v) Special Ordered Sets of type Two (SOS2) are sets of variables of which not more than two members may be non-zero in the final solution, with the further condition that if there are as many as two they must be adjacent. SOS2s were introduced Beale (1970) to make it easier to find global optimum solutions to problems containing piecewise linear approximations to a nonlinear function of single argument. The overall problem has an otherwise linear programming or integer programming structure except for such nonlinear

functions.

Consider the function f(y) Fig.2 as a piecewise linear function in one variable defined over the closed intervals $[\hat{y}_k,\hat{y}_{k+1}]$ k=1, … ,k-1, where the coordinates $(\hat{y}_k,f(\hat{y}_k))$,k=1, …,k, represent points $P_1, \ldots, P_k$.

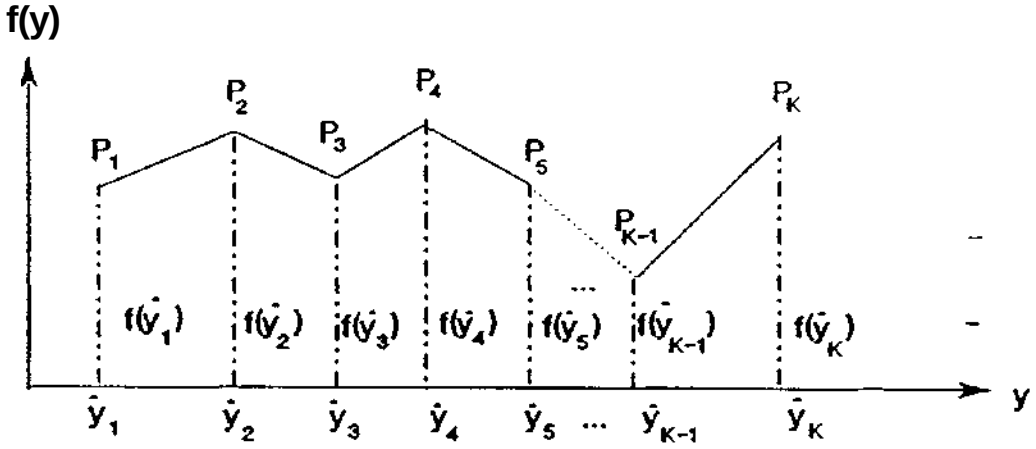


Fig.2

Any point y in the closed interval $[\hat{y}_k,\hat{y}_{k+1}]$ may be written as $y=x_k\hat{y}_k+x_{k+1}\hat{y}_{k+1}$, where $x_k+x_{k+1}=1$ and $x_k, x_{k+1} \geq 0$, similarly as f(y) is linear in the interval, it can be written as $f(y)= f(\hat{y}_k)x_k+ f(\hat{y}_{k+1})x_{k+1}$.

This leads to the representation of f(y) using a set of weighting variabl x_k, k=1,…k, by the following equality:

$$f(y) = f(\hat{y}_1)x_1 + f(\hat{y}_2)x_2 + \ldots + f(_k)x_k \quad (2.4)$$

where:

$$\hat{y}_1x_1 + \hat{y}_2x_2 + \ldots + \hat{y}_kx_k - y = 0,\ y \geq 0 \quad (2.5)$$

$$x_1 + x_2 + \ldots + x_k = 1,\ x_k \geq 0,\ k=1,\ldots,K \quad (2.6)$$

Plus the added condition that not more than two adjacent variables can be non-zero at any one time.

The set of weighted variables x_k are called the special variables and the rows (2.1)&(2.4), (2.2)&(2.5), and (2.3)&(2.6) are called the function row, reference row and the convexity row respectively.

3. Mathematical Statement of a General Discrete and Separable Programming Problem.

The classes of discrete variables are described in the previous section, as Binary, General integer, Semi Continuous variables and Special Ordered Sets of variables of type One, and Two. A statement of the general discrete programming problem, which contains the above types of variables and sets as well as continuous variables is set out below. Consider the index sets N1 ,..., N7 which are used to specify the different variable types. The data structure which is used to implement this representation within our experimental optimiser FortLP system also follows this variable indexing scheme.

N_1 = set of indices of Bounded Variables

$$l_i \leq x_i \leq u_i \qquad 0 \leq l_j,\ u_j \prec +\infty \quad j \in N_1 \tag{3.1}$$

N_2 = set of indices of Free Variables

$$-\infty \prec x_j \prec +\infty \qquad j \in N_2 \tag{3.2}$$

N_3 = set of indices of Binary Variables

$$x_j = 0 \text{ or } 1, \qquad j \in N_3 \tag{3.3}$$

N_4 = set of indices of General Integer Variables

$$l_j \leq x_j \leq u_j, \quad x_j \equiv 0 \ \ mod(1), \qquad j \in N_4 \tag{3.4}$$

N_5 = set of indices of Semi-continuous variables

$$\textit{either} \begin{cases} 0 < l_j \leq x_j \leq u_j \\ \qquad\qquad j \in N_5 \end{cases} \qquad \text{or} \begin{cases} x_j = 0 \end{cases} \tag{3.5}$$

$N6 = \bigcup_{\ell} N_{6\ell}$ where $N_{6\ell}$ is the set of indices of the ℓ^{th} SOS1 type Variables, and only one x_j can be non-zero.

$$x_j = 0 \text{ or } 1,\ j \in N_{6\ell},\ \ell=1,\ldots,\mathcal{L} \qquad (3.6)$$
$$\mathcal{L} = \text{No. of SOSls}$$

$N_7 = \bigcup_k N_{7k}$, where N_{7k} is the set of indices of the k^{th} SOS2 type Variables, and x_j satisfies the adjacency condition of this set.

$$0 \le x_j \le 1,\ j \in N_{7k}\ k = 1,\ldots,k \qquad (3.7)$$
$$k = \text{No. of S0S2s}$$

The total number of variables defined by these index sets (which are mutually exclusive, namely $\bigcap_{j=1}^{7} N_j = \varnothing$) is denoted by n,

$$n = \sum_{j=1}^{7} [N_j] \qquad (3.8)$$

Using these set definitions a general discrete programming model may be presented as:

$$Max\ x_0 = \sum_{j=1}^{n} a_{0j} x_j$$

subject to constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \begin{Bmatrix} \le \\ \ge \\ = \end{Bmatrix} b_i, \qquad i = 1,\ldots,P \qquad (3.10)$$

$$\sum_{j \in N_{6\ell}} x_j = 1, \qquad \ell = 1,\ldots,\mathcal{L} \qquad (3.11)$$

$$\sum_{j \in N_{7k}} x_j = 1, \qquad k = 1,\ldots,K$$

where:

$$\begin{aligned} & l_j \le x_j \le u_j, && j \in N_1 \cup N_4, \\ & -\infty \prec l_j,\ u_j \prec +\infty, && \\ & -\infty \prec x_j \prec +\infty, && j \in N_2, \\ & 0 \le x_j \le 1, && j \in N_3 \cup N_6 \cup N_7, \\ & 0 \le x_j \le u_j, && j \in N_5, \end{aligned} \qquad ((3.12)$$

with the further discrete restrictions such as,

$$
\begin{aligned}
& x_j = 0 \text{ or } 1, && \text{if } j \in N_3, \\
& x_j = 0 \bmod(l) && \text{if } j \in N_4, \\
& \text{either } \begin{cases} 0 < l_j \le x_j \le u_j \\ \\ \text{or} \quad x_j = 0 \end{cases} && \text{if } j \in N_5,
\end{aligned}
\tag{3.13}
$$

only one x_j can be non-zero if $j \in N_{6\ell}$, *and at*

most two adjacent x_j can be non-zero if $j \in N_{7k}$,

Function rows and reference rows included in (3.10), (3.11) represent the convexity rows of special ordered sets of type one and two.

The model therefore has $m = P + \mathcal{L} + K$ rows ad columns given by (3.8)

4.0 General Branch and Bound Algorithm

Preliminary Discussion; before starting with the algorithm some terminology needs to be defined. Maximisation is considered for all the discussion through out this report unless minimisation is explicitlymentioned.

a) A solution is said to be LP feasible if it satisfies all of the problem's linear constraint (3.10, 3.11, and 3.12), and similarly is integer feasible if it satisfies all the linear constraints and integer requirements (3.13).

b) An optimal solution is an integer feasible solution of the problem (if one exists) which in term of the objective function value, is either better or at least as good as other integer feasible solutions of that problem.

c) Following Beale's terminology, Beale (1985), in this report the term Global Entity is used to represent all binary, general integer, semi continuous and set variables (special ordered sets of type one and two). A discrete programming problem may have only set or semi continuous variables and no integer variables, in this case, it is inappropriate to address the model as an integer program.

d) An LP relaxed problem is a problem whose discrete restrictions (3.13) are not considered.

e) RIPBST, or cutoff value is initially fixed to a large negative number then it is updated by the integer feasible solution found during the tree search.

f) A good integer solution is an integer feasible (not necessarily optimum) solution whose objective function value x^{g0} lies within a user defined percentage of the original LP solution value x^{00}, that is $\theta X x^{00} \leq x^{g0}$ where $0< \theta \leq 1$. The value of x^{g0} can be used for the cutoff value before finding the first integer solution.

Branch and Bound (B&B) is a technique for solving certain constrained optimisation problems. It is particularly important for solving those problems whose solution by complete enumeration can often be prohibitively expensive. A wide variety of problems of this kind arise in Operational Research, Combinatorial Optimisation and Artificial Intelligence.

Taking into account practical computational experiences, the Branch and Bound method is considered to be the most efficient technique for attacking Discrete Programming (Combinatorial Optimisation problems). The Branch and Bound method carries out a progressive partitioned search of the solution space of a given problem. Starting with an LP relaxation, the space of all linear feasible solutions is repeatedly partitioned into smaller and smaller subsets, and an upper bound (in the case of maximisation) is calculated for the objective function value of the corresponding problems. After each partitioning, those subproblems with a bound that exceeds the solution value of a known integer feasible solution value are excluded from all further partitioning. The search continues until an integer feasible solution is found such that the solution value of its objective function is greater than the bound of all the remaining subproblems.

Depending on the optimisation requirement of a discrete model Branch and Bound algorithms may be designed in different ways. These algorithms may be of three main groups;

(i) Branch and Bound algorithm which computes all the optimal solutions,
(ii) Branch and Bound algorithm which provides a single optimal solution,
(iii) Branch and Bound algorithm which determines the k best integer feasible solution.
(iv) Branch and Bound algorithm which determines a good integer feasible solution.

These methods are generally characterised by their branching, bounding, solving, and selection techniques. In this chapter the branch and bound method for a single optimum solution (ii) of a given problem, (if a feasible solution of that problem exists) is considered, within this it is also possible to determine a good integer solution (iv). The algorithm for finding a single optimum solution (ii), can be easily extended to algorithm which computes all the integer optimal solutions (i), and the algorithm which determines the k best integer feasible solution (iii).

This Branch and Bound algorithm explores a search tree and in general, only search of a small proportion of the solution space may be required. The remainder of the search tree can be partly eliminated using the bound derived from a good integer feasible solution. When an integer solution is proven to be optimal the rest of the search tree is fully eliminated.

4.1 Statements of the Algorithm for Single Optimum Solution

We have developed and implemented a Branch and Bound algorithm which finds a single optimum solution. This algorithm uses the sparse simplex solver FortLP to process the LP relaxed subproblems. The algorithm first finds an integer feasible solution and then continues the tree search to find a better integer solution or prove the optimality of the current integer solution. For a detailed discussion of the algorithm see Mitra (1976), Mitra (1973), Benichou (1971), and Beale (1968). The steps of the algorithm can be stated as follows:

(0) INITIAL STEP: If an optimum solution to the LP problem does not exist Exit, else check the integer feasibility, if integer feasible Exit, else set the best integer solution value **(RIPBST)** to a large negative value, prepare a space for the list (stack) of subproblems of the search tree and continue.

(i) BRANCHING: Take a violated global entity from a current node P_k, create the two appropriate sub-problems, P_{k+1} and P_{k+2}, add them to the list of subproblems (see diagram 4.1) and store the associated basis

(ii) **SELECTION**: If the list of sub-problems is empty Exit, else select and remove a sub-problem from the list by node choice criterion.

(iii) SOLVING: Starting from a stored basis solve the selected sub-problem using a simplex algorithm. If the algorithm used is of the dual type the sub-problem may be discontinued should its objective function value become less than or equal to **Max (RIPBST)**; in this case go to (ii).

(iv) BOUNDING: If the sub-problem has no feasible solution go to (ii). If the objective function value is less than or equal to Max (RIPBST) go to (ii). If the solution meets all conditions of discreteness (3.13), of the sets and variables go to (v). Otherwise go to (i).

(v) Integer Solution: Set integer solution marker. If objective function value of this integer solution is equal to the original **LP** solution, Exit, else if x is greater than **Max (RIPBST),** up date the **Max (RIPBST)** Go to (ii).

If the integer solution marker is not set to any integer feasible solution and the list is empty then no feasible (integer) solution exists to the problem. Otherwise output the best integer solution.

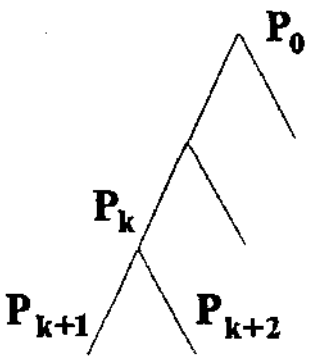


Diagram 4.1

4.2 Dealing With Different Classes of Variables.

A discrete programming problem as set out in section 2 is a problem which contains a number of discrete variables and set variables (generally global entities). If during the tree search the discreteness conditions of any discrete variable or set variables are not satisfied for a given subproblem, then the corresponding problem is analysed to choose a suitable unsatisfied global entity. Based on the result of this analysis two subproblems are proposed. The methods of dealing with different types of global entities are described in sections 4.3 to 4.7.

4.3 Zero-One and General Integer Variables.

The zero-one variables can be dealt with by generating two new sub-problems, setting the chosen variable to value 0.0 in one sub-problem and 1.0 in the other.

In the case of a general integer variable taking a non-integer value $x_{k=} [\bar{a}_{k0}] + f_{k'}\ [\bar{a}_{k0}] \equiv 0$ mod(l) and $0 < f_k < 1$ in the linear programming relaxation solution of a sub-problem. We create two sub-problems with new lower and upper bounds $x_k \geq [\bar{a}_{k0}] + 1$ and $x_k \leq [\bar{a}_{k0}]$.

4.4 Semi Continuous Variables.

Let x_i be an semi continuous (variable upper bound) variable. It is easily seen from the definition of an semi continuous variable (section 2) that in any of the following cases the semi continuous requirements is not satisfied.

Case i) $0 < x_j < l_j$,

or

Case ii) $u_j < x_j$

where l_i and u_i are the (conditional) lower and upper bound respectively. Case (ii) does not occur because the upper bound is imposed on the original **LPR**. If x_j is in the first range

case i) then, the semi continuous requirement is not satisfied and two new sub-problems are defined by setting the $x_i = 0$ in one and updating the lower .bound to l_i, in the other (diagram 4.2),

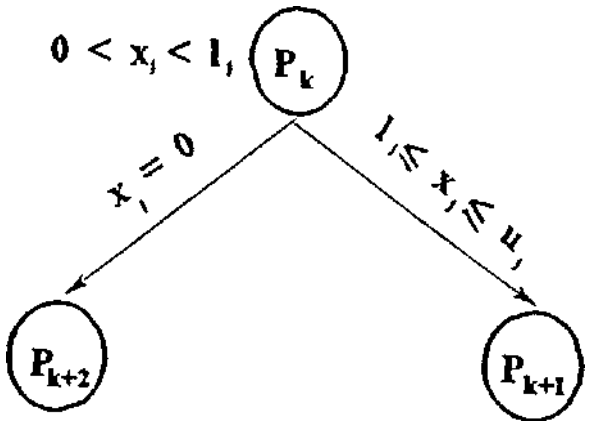


Diagram 4.2

4.5 Special Or dered Sets of Type One

Special ordered set of type one is a set of variables in which only one variable can take a value different from its lower bound typically zero see (section 2).
Consider the subproblem P_k which contains a special ordered set of type one (say $N_{6\ell}$) for which $x_j, \ldots, x_{j+p}$ Are the corresponding set variables, that is, $j,\ldots,j+p \in N_{6\ell}$. If in the optimum solution to P_k, the values of the set variables violate the special ordered set of type one requirement then two new subproblems are generated in the following way (diagram 4.3).

Find an index d, $j \leq d \leq j + p$ (methods of finding index d are discussed later in this chapter), and construct two subproblems P_{k+1}, P_{k+2} where,

P_{k+1}: Same as P_k and $x_k = 0$ for $d+1 \leq k \leq j + p$

P_{k+2}: Same as P_K and $x_k = 0$ for $j \leq k \leq d$

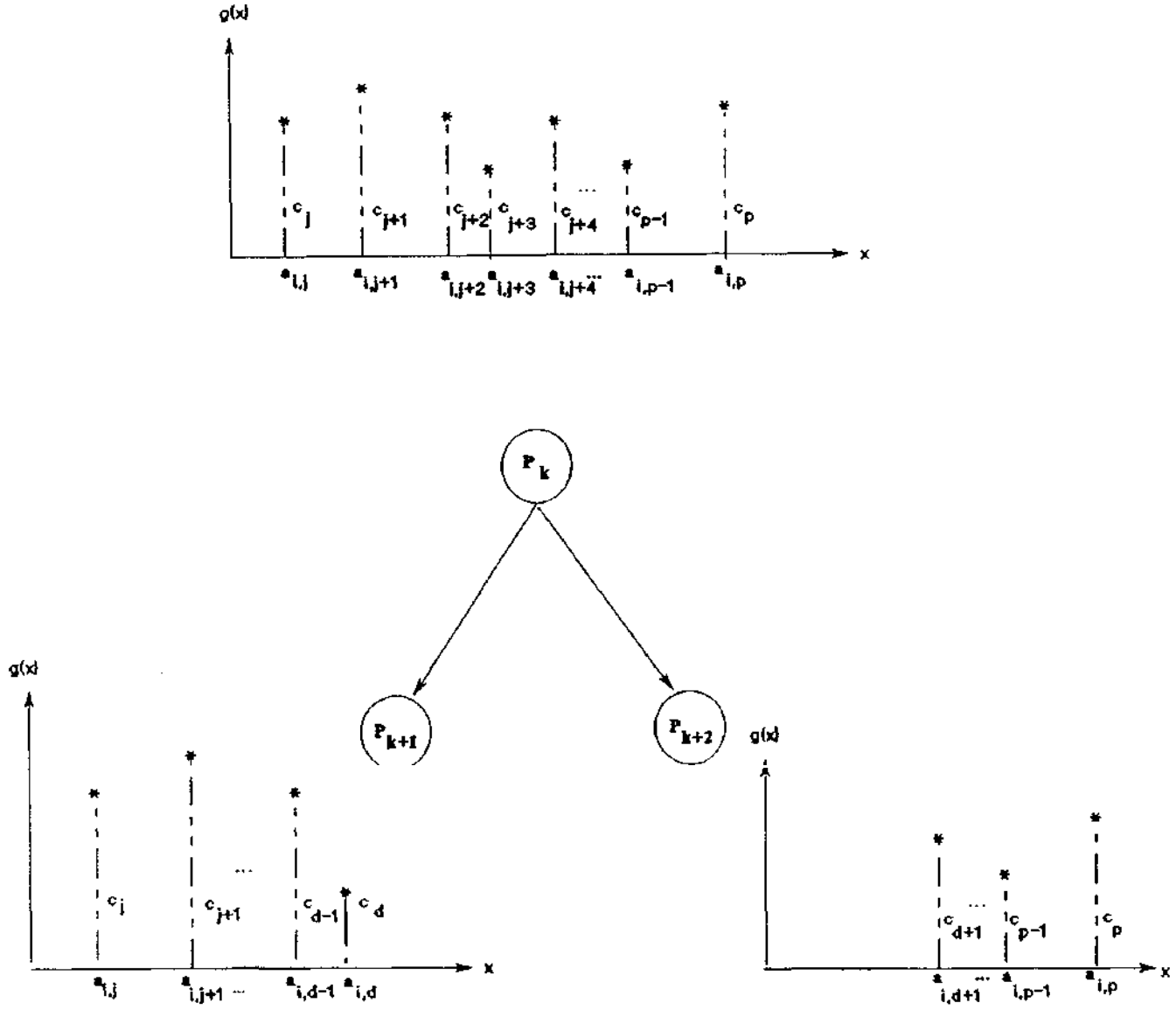


Diagram 4.3

4.6 Special Ordered Sets of TypeTwo.

Similarly consider subproblem P_k, which contains a special ordered set of type two (say N_{7k}) for which $x_j,...,x_{j+p}$ Are the corresponding members of the set variables that is, $j,...j+p \in N_{7k}$. If the solution values of these variables contain more than two nonzeros or there are two

nonzero set variables which are not adjacent then, two new subproblems are generated in the following way (diagram 4.4).

Find an index d, , $j \le d \le j+p$, $j \le d \le j+p$, (methods of finding index d are discussed later in this chapter), and construct two subproblems P_{k+1}, P_{k-} (diagram4.4) where,

P_{k+1}: Same as P_k and $x_k = 0$ for $d + l \le k \le j + p$

P_{k+2}: Same as P_k and $x_k = 0$ for $j \le k \le d-l$

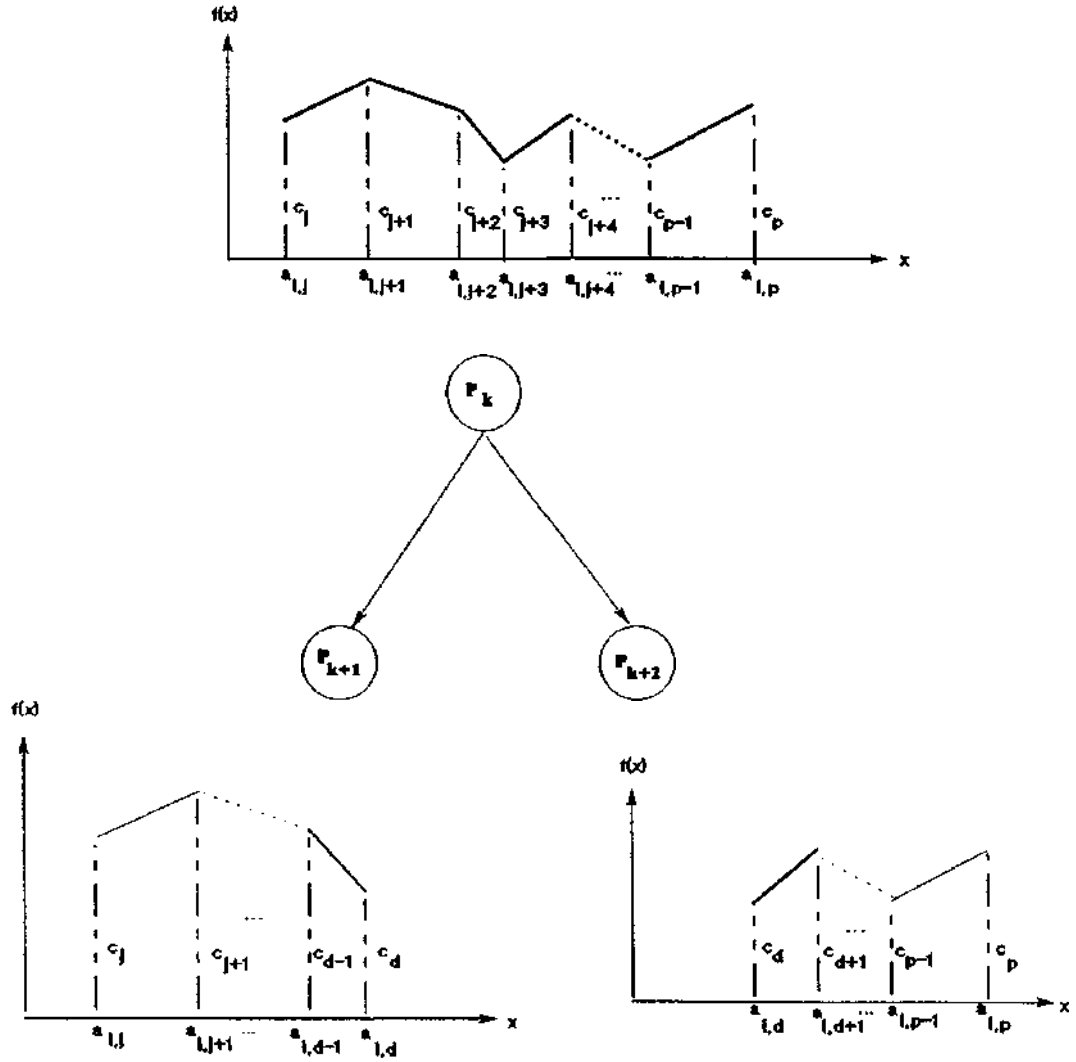


Diagram 4.4

4.7 Method of Partitioning Special Ordered Sets

A natural method for partitioning set variables which do not satisfy set conditions can be developed by using weighted average of their values in the following way (Tomlin 70). Let w_j be the weight associated with variables x_j, $j \in N_{6\ell}$ or $j \in N_{7k}$. These weights can be the coefficients of variables in the reference row (if one exists) of the set, that is, $w_j = a_{ij}$ where i is the index of the reference row of the set and j is the index number of the members of the set. If there is no reference row present, since the ordering of the set is defined implicitly within the problem then, one can simply consider the index number of each variable as its weight that is, $w_j = j$.

The weighted average of these variables is given by the expression,

$$\bar{w} = \sum_{j \in s} w_j \hat{x}_j \Big/ \sum_{j \in s} \hat{x}_j \ , \ \textit{where, } s \equiv N_{6\ell} \textit{ or } s \equiv N_{7k}, \tag{4.1}$$

The index d can be considered to be the partition indicator and is defined by the relation

$$w_d \leq \bar{w} < w_{d+1} \tag{4.2}$$

Choosing the biggest nonzero variable in a set as the marker can be another simple way of partitioning a set.

5. Tree Search Strategies.

The number of sub-problems which must be searched and solved is dependent on the variable and node choice strategies. Experiences with real life integer test problems indicate that applying a given branch and bound search strategy on different classes of integer programs lead to unpredictable results. For a given problem class, it is well known that a suitably tuned search heuristic of variable branching and node choice rules can often produce good discrete solution within a relatively small search tree. A variety of these methods has been investigated by researchers since the early 70s but none of these have shown dominating performance results and has become established as a standard technique. Therefore a discrete programming system which can provide a range of options for the variable and node choice is well suited for the investigation of different classes of integer programming applications.

5.1 Methods of Choosing an unsatisfied Global Entity

Choosing a global entity from the set of unsatisfied discrete variables and sets is an important step of the branch and bound procedure. This is because the choice of a less effective variable (global entity) can increase the size and complexity of the search tree with little progress towards finding the optimum solution.

The importance of this issue was observed by Land and Doig (1960) who first introduced the tree search algorithm to solve the discrete programming problems. Since then some of the researchers on this subject Beale (1965), Tomlin (1970), Benichou (1970), Mitra (1973), Gauthier (1977) and Beale (1985) have proposed different strategies for choosing the branching variable in order to reduce the size of the search tree.

In general these rules determine a ranked order (or priorities) for the global entities and

thereby identifies the branching variable or set of variables. If these rules lead to the same order of priority for the global entities during the search then these rules are considered to be static. On the other hand, if they lead to different orders of priority during the search then they are dynamic. The static priority orders are applied

a) when priorities are given by the modeller,

b) when priority order is computed from the deterioration of the functional value measured by pseudo-costs, and

c) by computing a decreasing order of absolute cost values of the discrete variables.

In contrast the dynamic priority orders can be determined by:

d) maximum fractional values of global entities at each node,

e) minimum fractional values of global entities at each node,

f) using the measure of non-discreteness for all global entities (see section 5.3), and

g) estimates of objective function value at each node for changing the bound on unsatisfied integer variables.

These estimates are obtained by alternative methods of penalty and bound calculations.

Let J be the set of indices of the non-basic variables such that if x_j is a non-basic variable then $j \in$. and let I be the set of the basic variables such that if x_i is a basic variable $i \in I$.

Now consider, the optimal solution of the 1st sub-problem represented as, Tomlin (1970),

$$x_0 = \bar{a}_{00} + \sum_{j \in J} \bar{a}_{0j}(-x_j)$$

$$x_i = \bar{a}_{i0} + \sum_{j \in J} \bar{a}_{ij}(-x_j)$$

$$\forall i \in I$$

where $x_0 = \bar{a}_{00}$ s the optimum (maximum) objective function value. If the solution is not integer feasible then there are one or more global entities with unsatisfied discrete

requirements. For example, consider x_k to be an integer variable with a non-integer value:

$$x_k = \bar{a}k_0, x_k = \beta_k + f_k \quad , \beta_k = [\bar{a}k_0\,] \; and \; 0 < + f_k < 1 \tag{5.1}$$

Land (1960), suggested that the most computationally convenient criterion for choice of the branching variable is to select a variable which is farthest from an integer, that is find an x_i such that,

$$\underset{j}{Max} \min \{1 - f_i, f_i\}, j \in N_3 \cup N_4 \tag{5.2}$$

In some circumstances, one may choose a variable X_j which is closest to an integer such that,

$$Min \min \{1 - f_i , f_i\} , j \in N_3 \cup N_4 \tag{5.3}$$

Beale (1965), introduced penalties as a criterion to choose the branching variable. These penalties are considered to be the deterioration of the objective function in one dual step due to the imposition of the new lower or upper bound on a variable.
The variable with the largest penalty is chosen and is used to generate two new sub-problems, one with the new lower bound of β_k + 1.0 and the other with the new upper bound of β_k on the variable x_k.

From the theory of cost ranging the imposition of the new lower bound β_k + 1.0 on x_k must decrease the objective function x_0 by the 'up penalty' P_U;

$$P_U \underset{j, \bar{a}_{kj} < 0}{\min} \left\{ (1 - f_{k0}) \bar{a}_{0j} / (\bar{a}_{kj}) \right\} \tag{5.4}$$

similarly the 'down penalty' or change of the objective function value by placing an upper bound β_k on x_k is given by P_D;

$$PD = \underset{j, \bar{a}_{kj} > 0}{\min} \{ f_{k0} \bar{a}_{0j} / \bar{a}_{kj} \} \tag{5.4}$$

Tomlin (1970) argues that because the penalties introduced previously are entirely based on the satisfaction of integer requirement of the basic variables and there may be some integer

Variables which are currently satisfied by being out of basis at their presumably integer upper or lower bounds, no use is made of the fact that this requirement* must be maintained when calculating the penalties. Therefore stronger penalties were introduced by considering separately the non-basic integer variables and the effect of changing their values by integer quantities. These penalties replace the above up and down penalties respectively;

$$P^*_U = \min_{j \bar{a}kj \prec 0} \begin{cases} \bar{a}_{0j}(1 - f_{k0})/(-\bar{a}_{kj}) & j \notin J \\ max\{\bar{a}_{0j}, \bar{a}_{0j}(1 - f_{p0})/(-\bar{a}_{kj})\} & j \in J \end{cases} \tag{5.5}$$

$$P^*_D = \min_{j \bar{a}kj \succ 0} \begin{cases} \bar{a}_{0j} f_{k0})/(-\bar{a}_{kj}) & j \notin J \\ max\{\bar{a}_{0j}, \bar{a}_{0j} f_{p0}/(-\bar{a}_{kj}\} & j \in J \end{cases} \tag{5.6}$$

where J is the set of non-basic integer variables.

Although these penalties are clearly stronger than P_U (5.3) and P_D (5.4) but, they can be still sharpened by considering the Gomory's mixed integer cutting plane algorithm Gomory (1960). Consider a non-integer value of an integer variable x_k (5.2), the following supplementary constraint must be satisfied by any integer solution obtained from the current subproblem:

$$s = -f_k - \sum_j f^*_{pj}(-x_j) \geq 0$$

where,

$$f^*_{pj} = \begin{cases} \bar{a}_{pj} & if\ a_{pj} \geq 0 and\ j \notin J \\ f_k(-\bar{a}_{pj})/(1 - f_k) & if\ a_{pj} < 0 and\ j \notin J \\ f_{pj} & if\ f_{pj} \leq f_k\ and\ j \notin J \\ f_k(1 - f_{pj})/(1 - f_k) & if\ f_{pj} > f_k\ and\ j \notin J \end{cases} \tag{5.7}$$

and f_{pj} *is the fractional part of* $\bar{a}_{pj}$

This currently unsatisfied constraint must be satisfied in any integer solution attainable from the current problem by dual simplex method. It may be observed* that the penalty for doing so is at least

$$p_G = \min_j \{ f_k \bar{a}_{oj} / f^*_{pj} \} \tag{5.8}$$

Benichou (1970) introduced Pseudo Costs. The concept of pseudo cost was presented to measure in a quantitative way the importance of the integer variables and to forecast the deterioration of the functional value when forcing an integer variable from a non integer to an integer value. Two quantities are attached to each integer variable x_i, they are called lower (PCL_i) and upper (PCU_i) pseudo cost. At the beginning of the search pseudo costs are generally not known, but they can be computed during tree scanning as follows:

$$PCL_j = \left| \frac{\overline{F}_{k+1} - \overline{F}_k}{f_j^k} \right|, \quad PCU_j = \left| \frac{\overline{F}_{k+2} - \overline{F}_k}{1 - f_j^k} \right|$$

where $\overline{F}_k$ is the functional value of node k, and $\overline{F}_{k+1}$, and $\overline{F}_{k+2}$ are the functional values of its successors.

These pseudo costs appear to be the deterioration of the functional value per unit of change in x_i one corresponding to a decrease and the other to an increase in x_i. Values of the pseudo cost of x_i depend on the node where they are computed. Although the value of the pseudo cost of x_i varies from node to node, Benichou (1970) say that they have the same order of magnitude in the most cases and therefore, they are assumed to be constant throughout the tree search. The disadvantage of this method is that at the beginning of the search pseudo costs are not known and one has to compute them after the exploring the node with the chosen variable x_j. To overcome this difficulty, Gauthier (1975) suggests the optimization of

the dummy subproblems at the beginning of the search for calculating the pseudo costs of each integer variable. The full optimisation of a dummy subproblem can take a large number of iterations therefore, some rules have been defined to stop the optimisation and use the premature solution to calculate the pseudo costs.

Beale (1985) contends that the above penalties are not always useful because many practical problems often happen to have several non-basic integer variables with zero reduced costs. Therefore, estimates which can be derived by a natural way based on the- Lagrangian relaxation discussed in Geofrrion (1974), may be more efficient towards finding the optimum solution. In the course of the tree search consider a subproblem stated as,

$$
\begin{aligned}
&\max \quad x_0 \\
&\textit{subject to:} \\
&\qquad x_0 + \sum_{\forall i} a_{0j} x_j = b_0 \\
&\qquad \sum_{\forall i} a_{ij} x_j = bi \qquad \forall I,j \\
&\qquad l_j \le x_j \le u_j \qquad \textit{for some } j
\end{aligned}
\tag{5.12}
$$

When considering the effect of imposing a change on the value of some integer variable x_k, it is natural to rewrite the constraints with this variable on the right hand side,

$$
\begin{aligned}
&\max \quad x_0 \\
&\textit{subject to:} \\
&\qquad x_0 + \sum_{i \neq k} a_{0j} x_j = b_0 - a_{0k} x_k \\
&\qquad \sum_{i \neq k} a_{ij} x_j = b_i - a_{ik} x_k \qquad \forall i \\
&\qquad l_j \le x_j \le u_j \qquad \textit{for some } j
\end{aligned}
\tag{5.13}
$$

If one decreases each b_i by $d_i=a_{ik}(1-f_k)$ simultaneously then, increasing the trial value of x_k by $(1-f_k)$ dose not change the value of x_0 or any other variable. The same argument applies

to decreasing x_k by f_k if, one considers $d_i = - a_{ik} f_k$.

Therefore, to evaluate the effect of changing x_k, the value of this variable is assumed to be constant while decreasing each b_i by d_i, for all rows. If π_i denotes the shadow price, or Lagrangian multiplier, on the i_{th} row, then if all d_i were small the original LP optimum functional value would be degraded by $\sum \pi_i x_i$ We then compute shadow prices π_i such that the optimum solution to the sub-problem:

maximizes x_0
subject to:

$$x_0 + \sum_j (\sum_i \pi_i \bar{a}_{ij}) x_j = \sum_i \pi_i b_i$$

$$l_j \le x_j \le u_j \qquad \textit{for some } j$$

where $\pi_0 = 1$.

The degradation $\Sigma \pi_i x_i$ is not often a realistic estimate of degradation D, but it defines a guaranteed lower bound. Note that $\Sigma \pi_i x_i = 0$ if x_k is a basic variable.

An upper bound on the degradation can be defined in terms of minimum and maximum shadow prices π_{MINi} and π_{MAXi}. Brearley et al (1975) show how to derive these bounds. In particular, note that, $\pi_{MIN0} = \pi_{MAX0} = 1$ and π_{MINi} is greater than or equal to zero if the i_{th} row is a less-than-or-equal-to type.

Therefore, $D = \sum_i \pi_i d_i + r_i$ Where;

$$r_i \begin{cases} (\min_i\{|\pi_i| + \theta, \pi_{max} - \pi_i\}) d_i & \text{for } d_i > 0 \\ (\max_i\{|\pi_i| + \theta), \pi_{mini} - \pi_i\}) d_i & \text{for } d_i < 0 \end{cases} \qquad (5.16)$$

and θ is a small positive tolerance. The explanation of using this tolerance in the formulae can be found in Beale (1985).

Since the value of D is dependent on value d_i and d_i is dependent on the direction of branching, two degradations can easily be estimated for each variable:

one from deriving the variable down to the integer part of its current value, and the other from deriving it up to this number plus one.

The largest estimated degradation may be used as a criterion for variable choice. There are some cases in which the formulator of the model can assess the relative importance of the variables. These degrees of importance can then be assigned as the priorities-and then the branching can take place according to the variable with the highest priority.

5.2 Node Choice

Likewise the variable choice discussed in section (5.1), choosing a subproblem from the list of unexplored subproblems is another important decision in the tree search method for solving an integer programming problem.

In the early Branch and Bound codes, the selection of the branching variable and the Selection of the next subproblem to solve were considered together. For instance a well Adopted methods was to branch on the variable with the greatest penalty and investigate the Branch in the other direction, Direbeek (1966), Beale and Small (1965).

However, many alternatives have been suggested since then to consider the node (subproblem) selection separately, Tomlin (1970), Mitra (1973), Gauthier (1976) and Beale (1985). In principle the node choice may be considered to be separate from the variable choice criterion. But not all such choices can be carried out independently of the information provided by the variable choice procedure. In a Branch and Bound code applying some of these rules are subject to availability of the information for example, an integer feasible solution and pseudo costs of integer variables.

The strategies which are long established in the literature and in practice are summarized below,

i) Last In First Out **(LIFO)** or, Depth First,

this is to select the last subproblem created provided that the upper bound on the objective function of this subproblem dose not exceed the value of **RIPBST** (in this case delete the subproblem from the list) otherwise, the most recently created subproblem with a bound that is better than **RIPBST** is selected. This strategy minimises the number of subproblems in the

list at any time. It also allows the subproblems in the list to be stored sequentially, which makes a fast access to the stack.

(ii) Choose a node by using Best Projection (BP) criterion due to Hirst (1969).

The BP criterion is to choose a node which maximises CB_k where,

$$CB_k = (F_k/\pi) - (F_0 - M(I))(S_k/S_0).$$

In this formulation $S_k = \sum \min\{ f_j, (1-f_j)\}$ is a measure of integer infeasibility of the solution

at node k, π is a user assigned parameter Mitra (1973), M(I) is a bound assigned to LP relaxed problem, and F_k is the objective function value of node k.

iii) Estimations of the integer solution values for each node due to Forrest et al (1974).

Choose a node which has the best estimated solution. There are different methods to estimate the solution value of a node, in general E_k the estimation of node k is given by the expression,

$$E_k = F_{(k-1)} + \sum D_j \, , \, j \in \{j \mid x_j \quad \text{unsatisfied global entity } \}$$

where, $D_j = \min \{DU_j , DL_j\}$, DU_j and DL_i a re the degradations of the objective function

value of node k, for assigning new lower and upper bounds to variable x_i respectively. It is obvious that the accuracy of this estimation depends on the accuracy of the degradation values. These degradation values can be derived by, a) calculation of pseudo cost of integer variables due to Gauthier (1977), b) calculation of penalties due to Tomlin (1970), and c) calculating pseudo shadow costs due to Beale (1985).

vi) Percentage error criterion due to Forrest et al (1974),

This is to choose a node with the smallest percentage error, P.E..

$$PE = \min_k \{ \frac{F_{best} - E_k}{F_{k-1} - F_{best}} \}$$

Where F_{best} is the best integer solution objective function value found so far and E_k is the estimation of the integer solution value at node k, gained by summing the estimated cost per unit change of the unsatisfied discrete variables.

5.3 Measure of Non-Discreteness

Integer infeasibility measure is used in most of the variable and node choice strategies adopted in Branch and Bound methods. The measure of integer infeasibility for a binary or general integer variable x_j in the solution of a given subproblem is often shown by t_j where,

$$t_j = \min \{f_j , (1-f_j)\},$$

and f_j is the fractional part of the solution value of the variable x_j (see section 5.1).

In an extended Branch and Bound algorithm for solving discrete programming problems with semi continuous and SOS type variables, some measure comparable to this integer infeasibility measure needs to be defined for these types of variables and sets .

An unsatisfied semi continuous variable x_i with solution value of, $0 < x_i < l_i$ in a given sub-problem has,

$$t_j = \min \{x_j/l_j , 1-(x_j/l_j)\}.$$

In the case of special ordered sets some analogy has been given by Forrest (1974), using the general upper bound of the convexity row $\sum x_j = 1$, that is,

$$t_\ell = 1 - \max(x_j), \quad j \in N_{6\ell}$$

for special ordered sets of type one and similarly,

$$t_k = 1 - \max(x_j), \quad j \in N_{6k}$$

for special ordered sets of type two.
The above expressions defined for special ordered sets do not properly reflect a discreteness measure since it is by no means certain that, the largest variable (or sum of adjacent variables) in a set on any branch will become unity in any feasible solution.

We have therefore introduced the following measure for an unsatisfied special ordered set of type one as,

$$t_\ell = \frac{\iota_\ell - k_\ell}{\rho_\ell},$$

where,
l_ℓ is the indexof the last nonzero in the set ℓ,
k_ℓ is the first nonzero index in the set ℓ, and
ρ_ℓ is the number of variables in the set ℓ.

Similarly for an unsatisfied special ordered set of type two, this measure is,

$$t_k = \max\{0, \frac{\iota_k - k_k - 1}{\rho_k - 1}\},$$

When the type one and type two sets conditions are satisfied t_ℓ, t_k assume zero values. Otherwise $0 < t_\ell, t_k < 1$ compare with the fractional values of zero-one integer variables. These measures can be also used for calculating the sum of fractions or integer infeasibility of a subproblem.

5.4 Bounding the Tree Search

Once an integer solution x_{0c} (the objective function value of the incumbent) has been obtained, it can be used to restrict the tree search in the following way.
A subproblem can be set to terminal (leaf) node of the tree and excluded from the further expansion by one of the following conditions:
i) eliminate a subproblem k with the upper bound value of U_k from the further expansion if $U_k \leq x_{0C}$.
ii) If a dual method is used to solve a subproblem then it always provides an upper bound on the objective function value of the subproblem after the dual feasibility is gained. If this upper bound U_k, reaches or passes the incumbent value $(U_k \leq x_{0C)}$, then the dual iteration can be abandoned together with the subproblem (node).
iii) If there is no feasible solution to the subproblem then again it can be abandoned.

Postponement of a Node in the Tree Search

The Lagrangean Relaxation method first proposed by Geoffrion (1974), often provides a good bound estimate for the objective function of the integer solution that may be found at a given node. Let E_k denote such a bound estimate for the k_{th} node. In theses cases if $E_k < x_{0C}$ then the search beyond node k can be postponed. This postponement strategy suggested by Beale (1985), can also influence the size of the search tree.

5.5 The Generalized Tree Search Strategies.

Experiments on real life test problems using different strategies show that each individual strategy behaves differently on different test problems and therefore, it is worthwhile to have a system which supports different strategies for the users.
In principle the node choice may be considered separately from the variable choice criterion.

But not all such choices can be carried out independently of the information provided by the variable choice procedure. In a Branch and Bound code applying some of these rules is subject to the availability of the information such as integer feasible solution and pseudo costs of integer variables.

A complete tree search strategy **STR(V,N)** is defined by a unique combination V,N of variable and node choice rules where, $V \in \{VC_1,..,VC_6\}$, $N \in \{NC_1,...,NC_6\}$, and

NC1: Last in first out **(LIFO)**

NC2: **BP** Criterion

NC3: Node Estimation using Penalties

NC4: Node Estimation using Pseudo-Costs

NC5: Node Estimation using Pseudo Shadow Costs

NC6: Percentage Error Criterion

VC1: Maximum Fraction

VC2: Minimum Fraction

VC3: Minimum Pseudo-Cost

VC4: Estimated Degradation

VC5: Minimum Penalty

VC6: Given Priority Order

The table below shows the possible combinations of node and variable choices:

	VC1	VC2	VC3	VC4	VC5	VC6
NC1	√	√	√	√	√	√
NC2	√	√	√	√	√	√
NC3	√	√			√	
NC4	√	√	√			
NC5	√	√		√		
NC6	√	√		√		

Table5.1

We observe that the combination of some strategies are not included, as they require an Excessive amount of computational effort.

6. Data Structures for representing Discrete variables, Set variables and the Branch and Bound Tree.

6.1 Data Structure for Representing Discrete and Set variables

In order to introduce discrete and set variables into our linear programming optimiser, FortLP, the existing data structure has been extended in the following way.

FortLP data structure uses a sparse column storage scheme which is described by Tamiz(1986). The following diagram gives the representation of the column j of the A matrix in a compact form,

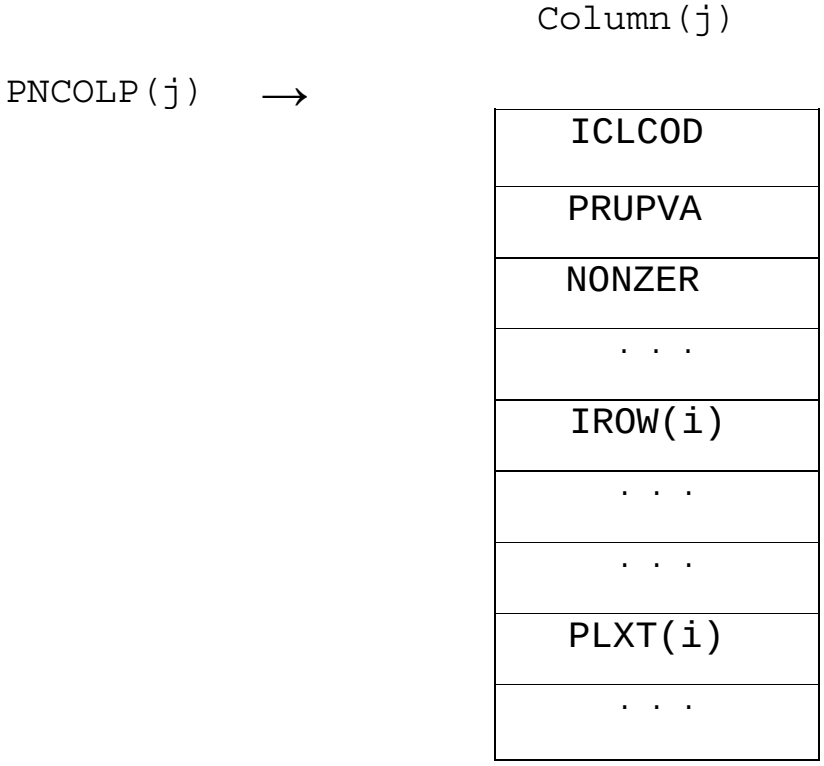


Diagram 6.1 - Data Structure for representing Discrete variables

where,

PNCOLP(j), is the pointer to the beginning of column j of the A matrix,
ICLCOD, is the column code,
PRUPVA, is the pointer to the upper bound value,
NONZER, is the number of non-zero entries in column j,
IROW(i), is the row position of th e i_{th} non-zero entry in column j, and
PLXT(i), is the address in the distinct non-zero pool of its corresponding value.

ICLCOD was originally used to identify ordinary, bounded, and fixed variables. Here we extend ICLCOD to define binary, general integer, and semi-continuous variables as well as ordinary, bounded, and fixed variables. The lower bound on continuous and semi-continuous

variables are stored separately in an array called **RLOFXV(NCOL).**

To represent the variables of a given **SOS1** or **SOS2** the following items of information are introduced. The first column **SETBEG**, the last column **SETEND**, the reference row **REFROW**, the convexity row **CONROW**, and the function row **FUNROW**. In order to incorporate this information into **FortLP**, a compatible data structure is introduced which extends the existing data structure such that two new locations added to the end of each packed column belonging to a given set. The first location **SETNTP** presents the set number and type (if negative then **SOS1** otherwise **SOS2)**, and the second location **SOLVAL** presents the current solution value of the column. These solution values are refreshed during the tree development process by storing this information with the column, the average weighted reference row value is easily computed and utilised in the partitioning of the set. The pointers and the contents of the storage structure are displayed in diagram 6.2.

6.2 Data structure for representing the Search Tree

The stages of Branch and Bound algorithm are conceptually represented as a search tree in which each node of the tree represents a subproblem. The root of the tree corresponds to the original linear programming relaxation of the problem. When any node or subproblem is explored, either two new branches with corresponding sub-problems are created from it or the node is bounded from further expansion (see section 5.4). In either case the processed node is eliminated from the list of unexplored subproblems. Therefore, only the alternative subproblems (unexplored nodes) which need to be solved are stored. It is easily seen that the information concerning the leaf nodes of the tree are sufficient to represent the entire search

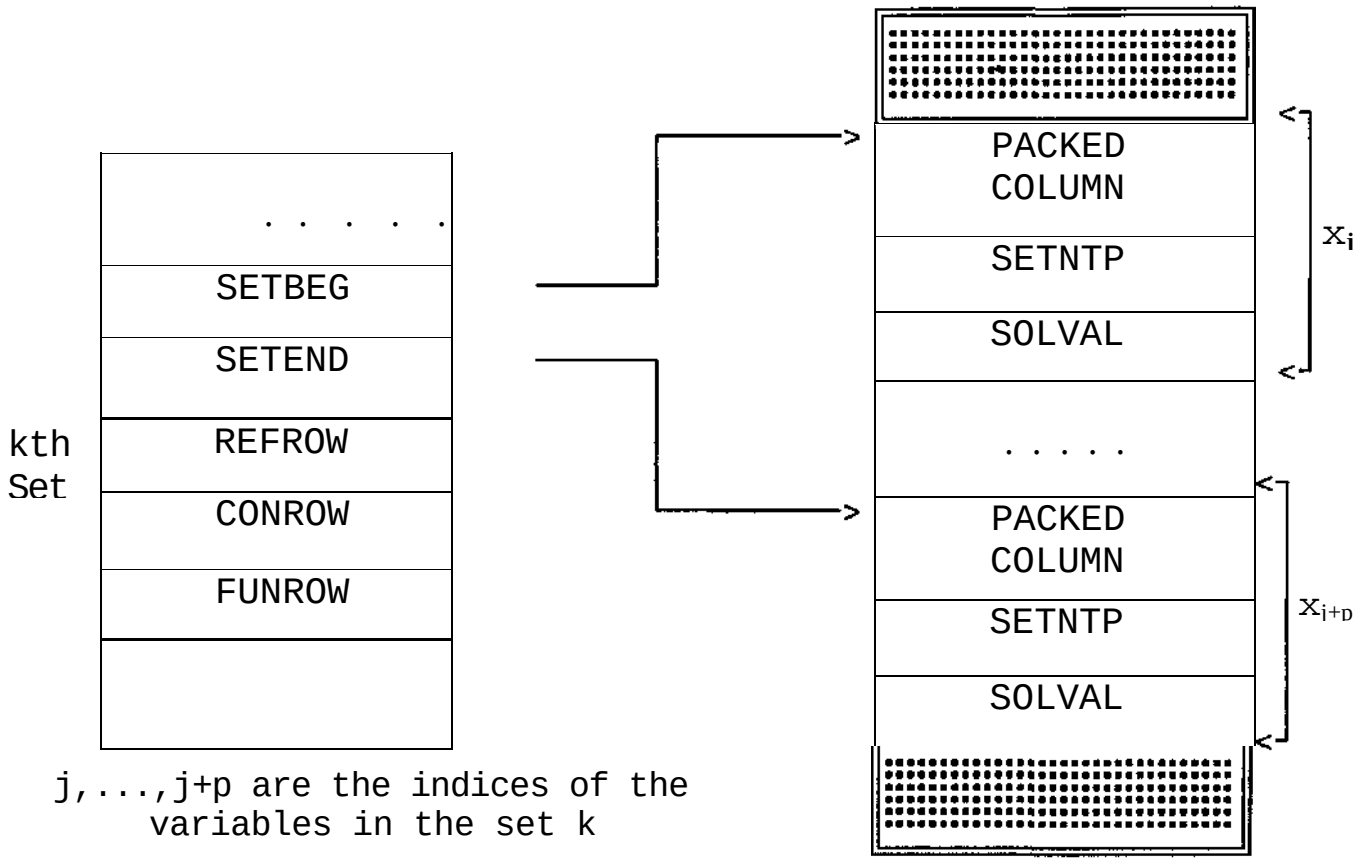


Diagram 6.2 - Data structure for representing the k_{th} Special Ordered Set

tree. Following the sequence in which the leaf nodes are generated, they are put in a list linearly, overwriting a processed node when appropriate. This data structure is well known as the stack mechanism.

In defining the subproblem only the bound information are altered thus, size of the linear programming subproblems are constant . From the point of view of storage it is a considerable advantage that the A matrix is stored only once.
In this approach, only the differences between the parent and the subproblem need to be stored. These differences are in the bounds on the discrete variables, the upper bound on the objective function value, some estimate of the objective function value and the optimum basis parent from which the new subproblem is generated. The storage requirement for

representing a node may be computed in the following way.

Let,

NCOL be the number of columns, and
MROW be the number of rows of a model including the objective function.

For each node we store the following vector array,

ICOLPT(NCOL), the array of pointers to star of each column, **INTEGER*4,**
ICOLBT(NCOL), the array of column types, this covers binary, general integer and semi-continuous variables, **INTEGER*2,**
ICOLVP(NCOL),the array of pointers to the current bound values, **INTEGER*2,**and
IBSIND(MROW), the array of basic columns for the node, **INTEGER*4.**

We also store the following scaler information for each node,

PARNOD, the parent node number,
IDEPTH, the depth of the node in the tree,
SUMINF, the non-discreteness measure of the node,
NONDIS, the number of unsatisfied discrete variables and sets,
RPAROB, the upper bound on the objective function value of the node,
RESTIM, an estimate of the integer feasible solution deriving from the node,
VARCHO, the chosen branching variable or set variables marker, and
SETCHO, the set chosen for branching.

Diagram 6.3 contains an explanation of the storage structure.

The storage requirement S to represent a node is computed as,

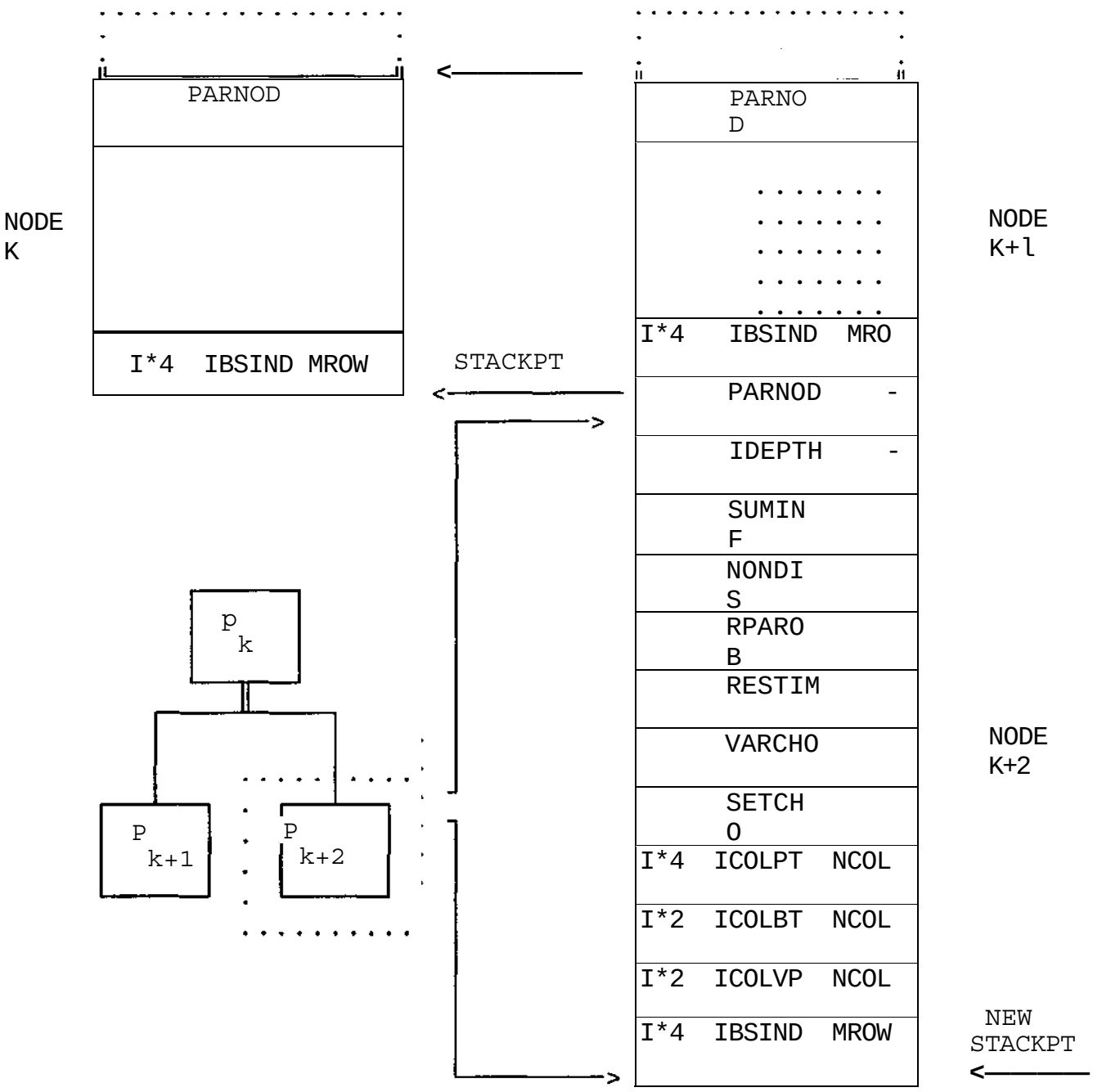
$$S \text{ (full word)} = 2*\mathbf{NCOL} + \mathbf{MROW} + 8,$$

And the total storage requirement **TS** is equal to,

$$\mathbf{TS} = \mathbf{S}*\mathbf{MAXSTK},$$

where **MAXSTK** is the user assigned parameter for the maximum number of waiting nodes (stack).

If the problem size is large and the total storage requirement TS is larger than the available



Note that, by the stack mechanism node k is overwritten by node k + l followed by k+2 which extends the stack.

Diagram 6.3 – Illustrates the Node Representation.

memory then this information is stored in a direct access file. The list of unexplored node is stored in an array with **MAXSTK** size which provides the pointers to the beginning of each node where is kept or gives the record number on the file.

7. Experimental Results.

As a part of our investigation a number of test problems were collected. These test problems are listed in appendix 2 of this report. Majority of these test problems are real life models which arise in different application covering many industry sectors and MIPLIB (Mixed Integer Programming Library of test problems). Given the wide range of problem types and fairly erratic behaviour of search trees, we have decided to investigate only six strategies (table 7.1) out of twenty four possible strategies (table (5.1) over a number of models listed in table 7.2.

The preliminary investigation of other strategies over a few chosen test problems showed a wide variation of the search tree.

Variable Choice Strategy (VARCHO)		
VARCHO= 1	VC1	Minimum Fraction, (Minmin)
2	VC2	Maximum Fraction, (Minmax)
3	VC3	Given Priority Order
NODE Choice Strategy (FNODCH & SNODCH)		
FNODCH& SNODCH 1	NC1	Last In First Out, (LIFO)
2	NC2	~ Extended Last In First Out, (E-LIFO)
3	NC3	Best Projection, (BP)

~ NC2: an extended LIFO criterion which first fixes the chosen variable to the integer value farthest from its fractional value and then process the corresponding node.

Table 7.1 - Explanations of VC# and NC#

Table 7.2 represents the list of chosen models and includes the following items of information:-

NAME - name of the problem
ROWS - number of constraints in the problem, not including free rows

COLS - total number of variables in the problem
BV - number of variables that are binary
UI - number of general integer variables
SC - number of semi-continuous variables
S1, S2 - number of special ordered sets of type one and two
IP obj. - Optimum integer solution to the problem,
LP obj. - optimal solution to the linear relaxation of the problem

NAME	ROWS	COLS	BV	UI	SC	S1	S2	IP obj.	LP obj.
MODGLOB	291	422	98	---	---	---	---	20740508.0	20430947.0
EGOUT	98	141	55	---	---	---	---	568.10	149.58
KHB05250	101	1350	24	---	---	---	---	106940226.0	95919464.0
FIXNET3	478	878	378	---	---	---	---	51973.0	40717.01 -
P0040	23	40	40	---	---	---	---	62027.0	61796.5
P0201	133	201	201	---	---	---	---	7615.0	6875.0
P0291	252	291	291	---	---	---	---	5223.7	1705.1
MAT3B	96	710	10	---	---	70	---	72023.76	60728.3
PROBLEM 1	534	632	7	1	---	2	1	53.36	54.40
PROBLEM 2	652	643	18	1	---	2	1	53.36	54.40
MOZIGEN	11	19	1	1	1	2	1	95.16	93.43
BEALE	171	303	38	---	---	11	---	100.00	100.00
S2MATRIX	51	450	—	—	---	---	25	157.9	225.88
GAS1	211	75	12	---	---	---	---	-6.65	-6.79
MICHSC	90	127	64	---	63	---	---	5177.8	5167.0
HYPERPL	128	68	63	---	---	---	---	1336.5	333.33
AIR02	50	6774	6774	---	---	---	---	7810.0	7640.0
STEIN 15	36	15	15	---	---	---	---	9.0	7.0
GRAY2IN	35	48	24	---	---	---	---	202.3	185.5
SAMPLE2	46	67	21	---	---	---	---	375.0	247.0
GRAY9IN	63	96	48	---	---	---	---	280.95	256.01

Table 7.2 - Model statistics

The experimental results for these test problems are presented in tables 7.3 to 7.8. In these tables of test results we have decided not to include time taken for solving these problems since the performance of different Branch and Bound algorithms are closely connected with the number of sub-problems (nodes) which are proposed and investigated by the search.

The first row of these tables indicates the strategy used for solving the problems, (NC#, VC#). Column marked with "Name" gives the name of the models. Columns marked with "First Integer Solution", present the number of nodes processed to find the first integer solution and the corresponding solution. Similarly columns with the heading "Best Integer Solution" present the Best integer solution found so far. Number of integer solutions found during the course of a search is reported in a column with the heading "no. of int. feas sols". Finally, we report the total no of nodes investigated in the column marked "Total no. of nodes".

column with the heading "no. of int. feas sols". Finally, we report the total no of nodes investigated in the column marked "Total no.of nodes".

Search Strategy, STR (NC1, VC1)						
Name	First Integer Solution		Best Integer Solution		no. of int feas sols.	Total, no. of nodes
	no. nodes	obj. value	no. nodes	obj. value		
MODGLOB	98	36180511.5	46967 ~	26467978.1	184	49801
EGOUT	94	626.8	1333	568.1	12	5240
KHB05250	23	108684849.0	15769	106940226.0	6	24388
FIXNET3	183	100073.1	45849 ~	61986.0	94	49801
P0040	2	62166.0	8	62027.0	3	144
P0201	19	8635.6	533	7615.0	9	4558
P0291	16	25964.4	614	5223.7	15	2164
MAT3B	627	85857.0	33583 ~	81807.0	43	49801
PROBLEM 1	7	48.98	35	53.36	6	36
PROBLEM2	6	48.98	39	53.36	7	40
MOZIGEN	5	111.5	9	95.16	3	12
BEALE	528	100.0	528	100.0	1	528
S2MATRIX	38	157.9	36838	157.9	5	40301
GAS1	9	-6.22	39	-6.65	4	40
MICHSC	43	5241.27	4137	5177.8	24	4800
HYPERPL	207	1336.5	207	1336.5	1	300
AIR02	19	7984.0	49	7810.0	3	58
STEIN15	6	9.0	6	9.00	1	300
GRAY2IN	16	238.89	134	203.34	9	162
SAMPLE2	21	445.0	196	375.0	3	380
GRAY9IN	12	285.74	683	280.94	7	782

Table 7.3 - LIFO and Minmin

In the following tables solutions marked with the sign " ~ "are the best (not necessarily optimal) integer solutions found during the course of a search. We could not terminate the search due to a long performance time or lack of space for storing the sub-problems.

Search Strategy, STR(NC1, VC2)						
Name	**First Integer Solution**		**Best Integer Solution**		**no. of int feas sols.**	**Total, no. of nodes**
	no. nodes	**obj. value**	**no. nodes**	**obj. value**		
MODGLOB	**41**	**29764906.2**	**30326 ~**	**26007495.0**	**336**	**49801**
EGOUT	**94**	**670.0**	**10046**	**568.1**	**20**	**26236**
KHB05250	**19**	**111632934.0**	**10002**	**106940226.0**	**9**	**13524**
FIXNET3	**180**	**232184.7**	**10119~**	**208771.7**	**86**	**49801**
P0040	**11**	**62597.0**	**56**	**62027.0**	**8**	**120**
P0201	**18**	**8595.0**	**219**	**7615.0**	**13**	**938**
P0291	**36**	**16342.8**	**43131~**	**14905.0**	**8**	**49801**
MAT3B	**125**	**82705.0**	**39681~**	**78855.0**	**14**	**49801**
PROBLEM 1	**7**	**48.98**	**27**	**53.36**	**6**	**28**
PROBLEM2	**6**	**48.98**	**35**	**53.36**	**7**	**36**
MOZIGEN	**7**	**111.5**	**9**	**95.16**	**3**	**12**
BEALE	**28**	**100.0**	**28**	**100.0**	**1**	**28**
S2MATRIX	**38**	**157.9**	**38**	**157.9**	**1**	**36050**
GAS1	**2**	**-6.65**	**2**	**-6.65**	**1**	**12**
MICHSC	**80**	**5308.4**	**29855~**	**5279.1**	**8**	**49801**
HYPERPL	**105**	**1336.5**	**105**	**1336.5**	**1**	**132**
AIR02	**13**	**26740.0**	**921**	**7810.0**	**8**	**922**
STEIN 15	**6**	**9.0**	**6**	**9.0**	**1**	**264**
GRAY2IN	**9**	**232.9**	**101**	**202.34**	**4**	**142**
SAMPLE2	**43**	**565.0**	**297**	**375.0**	**7**	**404**
GRAY9IN	**17**	**371.3**	**368**	**280.9**	**17**	**1044**

Table 7.4 - LIFO and Minmax

In our investigation of the LIFO criterion we have set the maximum number of nodes to 49801 and for the BP criterion we have set the maximum number of nodes to 9980 since BP criterion creates a bushier tree than **LIFO** and requires more storage.

Search Strategy, STR(NC2, VCl)						
Name	**First Integer Solution**		**Best Integer Solution**		**no. of int feas sols.**	**Total. no. of nodes**
	no. nodes	**obj. value**	**no. nodes**	**obj. value**		
MODGLOB	**35**	**20880067.2**	**30204**	**20740508.0**	**13**	**35288**
EGOUT	**39**	**601.4**	**5470**	**568.1**	**9**	**8300**
KHB05250	**21**	**129159364.0**	**17783~**	**114538541.0**	**66**	**49801**
FIXNET3	**105**	**86408.0**	**12503 ~**	**80309.0**	**13**	**49801**
P0040	**11**	**62134.0**	**131**	**62027.0**	**4**	**134**
P0201	**45**	**11050.0**	**15657**	**7615.0**	**76**	**17564**
P0291	**31**	**110611.3**	**46104**	**5223.0**	**194**	**46336**
MAT3B	**—**	**—**	**—**	**—**	**0**	**49801**
PROBLEM 1	**12**	**-574.33**	**42**	**53.36**	**10**	**44**
PROBLEM2	**12**	**-574.33**	**64**	**53.36**	**11**	**66**
MOZIGEN	**3**	**96.16**	**7**	**95.16**	**3**	**8**
BEALE	**17**	**100.00**	**17**	**100.00**	**1**	**17**
S2MATRIX	**38**	**157.9**	**38**	**157.9**	**1**	**34060**
GAS1	**4**	**-6.49**	**9**	**-6.65**	**2**	**22**
MICHSC	**18**	**5274.5**	**8260**	**5177.8**	**34**	**8746**
HYPERPL	**92**	**1336.5**	**92**	**1336.5**	**1**	**92**
AIR02	**3**	**8026.0**	**16**	**7810.0**	**2**	**18**
STEIN15	**6**	**9.0**	**6**	**9.0**	**1**	**286**
GRAY2IN	**11**	**234.64**	**224**	**202.34**	**9**	**242**
SAMPLE2	**13**	**475.0**	**119**	**375.0**	**4**	**264**
GRAY9IN	**7**	**300.1**	**331**	**280.94**	**9**	**754**

Table 7.5 - E-LIFO and Minmin

Search Strategy, STR(NC2, VC2)						
Name	First Integer Solution		Best Integer Solution		no. of int feas sols.	Total, no. of nodes
	no. nodes	obj. value	no. nodes	obj. value		
MODGLOB	45	28971733.9	28365 ~	20740508.0	12	49801
EGOUT	38	631.5	41519~	568.4	15	49801
KHB05250	19	122690022.0	5568	106940266.0	21	9566
FIXNET3	59	76085.0	16902~	60213.0	15	49801
P0040	11	62134.0	91	62027.0	8	110
P0201	50	11370.0	2153	7615.0	33	2520
P0291	25	61361.2	15614~	41361.0	7	49801
MAT3B	115	81713.0	1603~	76313.0	71	49801
PROBLEM1	12	-547.3	36	53.36	10	38
PROBLEM2	12	-547.3	40	53.36	11	42
MOZIGEN	3	96.16	7	95.16	3	8
BEALE	22	100.0	22	100.0	1	22
S2MATRIX	39	157.9	39	157.9	1	3805
GAS1	5	-6.31	30	-6.65	3	30
MICHSC	26	5322.7	4616-	5285.92	9	49801
HYPERPL	136	1336.5	136	1336.5	1	140
AIR02	10	9404.0	1805	7810.0	10	4650
STEIN 15	7	9.0	7	9.0	1	262
GRAY2IN	14	222.09	123	202.34	6	160
SAMPLE2	15	415.0	29	375.0	2	234
GRAY9IN	14	340.3	925	280.9	14	1282

Table 7.6 - E-LIFO and Minmax

Search Strategy, STR(NC3, VC1)						
Name	First Integer Solution		Best Integer Solution		no. of int feas sols.	Total. no. of nodes
	no. nodes	obj. value	no. nodes	obj. value		
MODGLOB	44	21464525.0	2202~	20763860.0	21	9980
EGOUT	46	601.44	8559	568.1	9	9250
KHB05250	23	129115788.0	9012~	116836089.0	50	3980
FIXNET3	70	55845.0	8305~	55845.0	1	9980
P0040	6	62119.0	27	62027.0	4	134
P0201	29	7735.0	38	7615.0	3	3692
P0291	78	59946.2	5223	5223.74	44	6779
MAT3B	170	79646.0	7806	72023.7	43	9876
PROBLEM 1	24	52.27	36	53.36	4	43
PROBLEM2	24	52.28	36	53.36	5	43
MOZIGEN	4	95.16	4	95.16	1	11
BEALE	1305	100.0	1305	100.0	1	9767
S2MATRIX	---	---	---	---	0	9980
GAS1	5	-6.49	9	-6.65	2	23
MICHSC	918	5209.0	4043	5177.86	12	4171
HYPERPL	280	1336.5	280	1336.5	1	323
AIR02	3	8416.0	29	7810.0	3	47
STEIN 15	9	9.0	9	9.0	1	277
GRAY2IN	10	212.39	166	202.34	5	192
SAMPLE2	20	505.0	293	375.0	5	333
GRAY9IN	19	300.19	168	280.9	7	665

Table 7.7 - BP and Minmin

Search Strategy, STR(NC3, VC2)						
Name	First Integer Solution		Best Integer Solution		no. of int feas sols.	Total, no. of nodes
	no. nodes	obj. value	no. nodes	obj. value		
MODGLOB	31	20751102.1	8243~	20745508.0	18	9980
EGOUT	39	631.54	672~	602.32	4	9980
KHB05250	20	116800424.0	5158	106940226.0	13	3322
FIXNET3	59	76085.0	5931~	75213.0	13	9980
P0040	7	62119.0	68	62027.0	3	122
P0201	18	7805.0	47	7615.0	3	735
P0291	30	35072.0	327	26869.0	3	9980
MAT3B	83	76313.0	4303~	72986.6	16	9980
PROBLEM1	24	52.27	34	53.36	4	47
PROBLEM2	24	52.28	34	53.36	4	47
MOZIGEN	4	95.16	4	95.16	1	11
BEALE	1296	100.0	7697~	100.0	8	9980
S2MATRIX	894	197.3	6305~	187.9	17	9980
GAS1	6	-6.49	12	-6.65	2	19
MICHSC	14	5189.13	16	5177.8	2	4759
HYPERPL	81	1336.4	81	1336.4	1	134
AIR02	4	26048.0	59	7810.0	4	63
STEIN15	9	9.0	9	9.0	1	273
GRAY2IN	11	224.3	100	202.3	5	145
SAMPLE2	16	415.0	66	375.0	1	235
GRAY9IN	10	298.2	618	280.9	6	1041

Table 7.8 – BP and Minmax

A Discussion of Test Results

In this section we discuss our results reported in the previous section. We have prepared table (7.9) which presents the best performance and the worst performance of the strategies used for solving each test problem. We have compiled this table by considering each test problem and looking up tables (7.3) to (7.8) and extracting the best and the worst performing heuristics. Our measure of search performance is based on the number of nodes processed and the difference between the optimal integer solution to the problem and the best solution found so far.

Table (7.9) together with tables (7.3) to (7.8) demonstrate the erratic behaviour of different variable choice and node choice criterion. For example problem **MODGLOB** is solved only by **STR(NC2,VC1)** which scores the maximum number of worst performances (6 cases) in the table. **STR(NC3, VC2)** seems to produce the best result although it cannot produce any solution for problem **BEALE** which was solved by other strategies.

The most difficult problems amongst our test problems are the first four test problems. This includes **FIXNET3** which was not solved by any of the above strategies however, **STR(NC3, VC1)** finds a very close integer feasible solution to its integer optimal solution. The least difficult problem is **AIR02** which is a scheduling problem with 6774 binary variables.

Worst and Best Performing Strategies						
Name	Worst Performance			Best Performance		
	Strategy	no. nodes	obj. value	Strategy	no. nodes	obj. value
MODGLOB	NC1-VC1	46967~	264679978.0	NC2-VC1	30204	20740508.0
EGOUT	NC2-VC2	41519~	568.4	NC1-VC1	1333	568.1
KHB05250	NC2-VC1	17783~	114538541.0	NC3-VC2	5158	106940226.0
FIXNET3	NC1-VC2	10119~	208771.7	NC3-VC1	8305~	55845.0
P0040	NC2-VC1	131	62027.0	NC1-VC1	8	62027.0
P0201	NC2-VC2	15657	7615.0	NC3-VC1	38	7615.0
P0291	NC2-VC2	15614~	41361.0	NC1-VC1	614	5223.0
MAT3B	NC2-VC1	49801 ~	—	NC3-VC1	7806	72023.7
PROBLEM 1	NC2-VC1	42	53.36	NC1-VC2	27	53.36
PROBLEM2	NC2-VC1	64	53.36	NC3-VC2	34	53.36
MOZIGEN	NC1-VC2	9	95.16	NC3-VC1	4	95.16
BEALE	NC3-VC2	7697~	100.00	NC2-VC1	17	100.00
S2MATRIX	NC3-VC1	9980~	...	NC2-VC2	39	157.9
GAS1	NC1-VC1	39	-6.65	NC1-VC2	2	-6.65
MICHSC	NC1-VC2	29855~	5279.1	NC3-VC2	16	5177.8
HYPERPL	NC3-VC1	1280	1336.5	NC3-VC2	81	1336.5
AIR02	NC2-VC2	1850	7810.0	NC2-VC1	16	7810.0
STEIN15	NC3-VC1	9	9.0	NC2-VC2	7	9.0
GRAY2IN	NC2-VC1	224	202.34	NC3-VC2	100	203.3
SAMPLE2	NC1-VC2	297	375.0	NC2-VC2	29	375.0
GRAY9IN	NC2-VC2	925	280.9	NC3-VC2	168	280.9

Table 6.4.1 - Performance Table

8. References

Beale, E.M.L. (1980), "Some Features of Integer Programming in SCICONIC" *Special Issue, Mixed Integer Programming in Mathematical Programming Systems, By: Jackson R.H.F. and O'Neil R.P., A joint Publication of the Computer Science Technical Section of ORSA.*

Beale, E.M.L., (1970), "Advanced Algorithmic Features for General Mathematical Programming Systems," *Integer and Nonlinear Programming J. Abadie North Holland P&C.*

Beale, E.M.L. (1985), "Integer Programming", *NATO ASI Series, VOL.F15, Computational Mathematical Programming by K. Schittkowski.*

Beale, E.M.L. and Forrest, J.J.H., (1976), "Global Optimisation Using Special Ordered Sets," *Mathematical Programming Study, 1976(10), pp 52-69.*

Beale, E.M.L. and Tomlin, J.a. (1969), "Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables," *Proc. 5th IFORS Conf. (Wiely, Newyork).*

Benichou, M., et al., (1971), "Experiments in Mixed-Integer Linear Programming, *Mathematical Programming 1971(1), pp 76-94.*

Dantzig, G.B., (1960), "On the Significance of Solving Linear Programming Problems with Some Integer Variables", *Econometrica 28 (1960) 30-44.*

Despain G.L., (1980), "Mixed Integer Programming with Honeywell's MPS," *Special Issue, Mixed Integer Programming in Mathematical Programming Systems, By: Jackson R.H.F. and O'Neil R.P., A joint Publication of the Computer Science Technical Section of ORSA.*

Driebeek N.J., (1966), "An Algorithm for the Solution of Mixed Integer Programming Problems",*Management Science 12 (1966) 576-587.*

Falk, J.E. and Soland, R.M., (1968), "An Algorithm for Separable Non-Convex Programming Problems," *Research Analysis Corporation, Maclean, Virginia, USA.*

Forrest, J.J.H., Hirst, J.P.H., and Tomlin, J. A., (1974), "Practical Solution of Large Mixed Integer Programming Problems with UMPIRE," *Management Science, 20(51, (1974), pp 736-773.*

Garfinkel, R.S., Nemhauser, G.L., (1972), "Integer Programming", *John Wiley and Sons, Inc.*

Gauthier, J.M. and Ribiere, G., (1977), "Experiments in Mixed Integer Programming Using Pseudo-Costs," *Mathematical Programming, 12(1977), pp 26-47.*

Geoffrion, A.M., (1974), "Lagrangian Relaxation for Integer Programming", *Mathematical Programming Study 2 pp 82-114.*

Hajian, M., T., (1992), "Investigation of Integer Programming Solution Methods", *PhD thesis, Maths Department, Brunei University (1992).*

Hirst, J.P.H., (1969), "Features Required in Branch and Bound Algorithms for Zero-One Mixed Integer Linear Programming", *Privately Circulated Manuscript, December 1969.*

Hoffman, K. and Padberg, M., (1984), "LP Based Combinatorial Problem Solving," *Computational Mathematical Programming, By, Klaus Schittkowski, NATO AS1 Series, Vol. F15, pp 65-123.*

Hummeltenberg, W., (1984), "Implementations of Special Ordered Sets in MP Software", *European Journal of Operational Research 17 (1984) 1-15*

Ibaraki, T., (1987), "Enumerative Approaches to Combinatorial Optimization", *Annals of Operations Research (1987), Vol(10-11), By, J.C. Baltzer AG Publishing Co.*

IBM Corporation, (1991), "Optimisation Subroutine Library, Guide and Reference Release 2", *IBM Publication (SC23-0519-02).*

Lawler, L.E., Wood, D.E. (1966), "Branch and Bound Methods - A Survey", *Operations Research 14, 699-719.*

Little, J.D.C., Murty, K.C., Sweeney, D.W., and Karel, C, (1963), "An Algorithm for Travelling Salesman Problem", *Operations Research 11 pp 972-989.*

Miller, C.E., (1963), "The Simplex Method for Local Separable Programming", *Recent Advances in Mathematical Programming, By, R. L. Graves and P. Wolf (Mc Grow Hill), pp 89-100.*

Mitra, G., (1973), "Investigation of Some Branch and Bound Strategies for the Solution of Mixed Integer Linear Programs," *Mathematical Programming 4 (1973), pp 155-170.*

Mitra, G., (1976), "Theory and Application of Mathematical Programming", *Academic Press Inc. London*

NAG Ltd., (1990), "FortLP, User Guide and Reference Manual", *NAG Ltd, Wilkinson House, Jordan Hill Road, Oxford, U.K.*

Nemhauser, G.L., et al, (1989), "Optimization", *Handbooks in Operations Research and*

Management Science Vol. 1 North-Holland Publishing co.

Nygreen B., (1991), "Branch and Bound with Estimation based on Pseudo Shadow Prices" *Mathematical Programming 59 (1991) 59-69.*

Savelsbergh, M.W.P., Sigismondi G.C., Nemhauser G.L., "Functional Description of MINTO, a Mixed Integer Optimiser", *Georgia Institute of Technology, Atlanta, Gorgia 30332.*

SCICON Ltd., (1989), "SCICONIC User Guide Version 1.40", *Scicon Ltd, Milton Keynes, U.K.*

Tomlin J.A., (1970), "Branch and Bound Methods for Integer and Non-Convex Programming," *Integer and Nonlinear Programming J. Abadie North Holland P&C.*

Tamiz, M., (1986), "Design, Implementation and Testing of a General Linear Programming System Exploiting Sparsity", *PhD thesis, Maths Department, Brunei University.*

Van Roy, T.J., Wolsey, L.A., (1987), "Solving Mixed Integer Programs by Automatic Reformulation", *Operational Research Vol35 Nol pp 45-57 1987.*

Appendix 1. Input Data Definition for The General Discrete Programming Problems.

The Input data definition for the general discrete programming problems is shown by an example, consider the following model,

$$\textit{Min:}\quad X_0 = 3X_1 + 4X_2 + 5X_3 - X_4 + 2X_5 + 3X_6 + 4X_7 + 3X_8 + 2X_9 + 2X_{11} + 2X_{12} + X_{13} + 10X_{14} + 25X_{15} - 7.5X_{16}$$

subject to:

$$2X_1 + 3X_2 - 4X_3 + X_4 + X_{17} + 2X_{18} + 3X_{19} \leq 25$$

$$5X_2 + 3X_3 - X_4 + 3X_{18} \geq 50$$

$$6X_1 + 3X_2 + 2X_{17} + X_{19} = 100$$

$$X_1 - 100\,X_4 \leq 0$$

$$X_6 + 2X_7 + 3X_8 + 4X_9 - X_{17} = 0$$

$$2X_{11} + 3X_{12} - X_{18} = 0$$

$$X_{13} + 5X_{14} + 10X_{15} + 15X_{16} - X_{19} = 0$$

$$X_5 + X_6 + X_7 + X_8 + X_9 = 1$$

$$X_{10} + X_{11} + X_{12} = 1$$

$$X_{13} + X_{14} + X_{15} + X_{16} = \mathbf{1}$$

$$0 \leq X_1 \leq 100$$

$$0 \leq X_2 \leq 20\text{, } \textit{and integer}$$

$$\begin{array}{c} either \\ or \end{array} \begin{cases} 1 \le X_3 \le 10 \\ X_3 = 0 \end{cases}$$

$$X_4 = 0 \text{ or } 1$$

$$X_{17}, X_{18}, X_{19} \ge 0$$

Only one of the set variables $X_5,...,X_9$ can be non-zero

Not more than 2 adjacent variables from set $\{X_{10},...,X_{12}\}$ and set $\{X_{13},...,X_{16}\}$ can be non-zero.

The input data definition of the above model in the extended mathematical programming format (MPSX) is as following,

```
NAME          MGINT
ROWS
 N  OBJ
 N  'MARKER'
 L  COS1
 G  COS2
 E  COS3
 E  CON1
 E  CON2
 E  CON3
 E  REF1
 E  REF2
 E  REF3
COLUMNS
    X1        OBJ       3.0
    X1        COS1      2.0
    X1        COS3      6.0
    X2        OBJ       4.0
    X2        COS1      3.0
    X2        COS2      5.0
    X2        COS3      3.0
    X3        OBJ       5.0
    X3        COS1      -4.0
    X3        COS2      3.0
    X4        OBJ       -1.0
    X4        COS1      1.0
    X4        COS2      -1.0
    X17       COS1      1.0
    X17       COS3      2 . 0
    X17       REF1      -1.0
    X18       COS1      2.0
    X18       COS2      3.0
    X18       REF2      -1.0
    X19       COS1      3.0
    X19       COS3      1.0
    X19       REF3      -1.0
 S1 S1SET1    'MARKER'  'SETORG
    X5        OBJ       2.0          REF1      1.0
    X5        CON1      1.0
    X6        OBJ       3.0          REF1      1.0
    X6        CON1      1.0
    X7        OBJ       4.0          REF1      2.0
    X7        CON1      1.0
    X8        OBJ       3.0
    X8        CON1      1.0          REF1      3.0
    X9        OBJ       2.0
                                     REF1      4.0
```

```
    X9          CON1        1.0
    S1SET1E     'MARKER'    'SETEND'
 S1 SISET2      'MARKER'    'SETORG'          REF2        1.0
    X10         CON2        1.0
    Xll         OBJ         2.0               REF2        2.0
    Xll         CON2        1.0
    X12         OBJ         1.0               REF2        3.0
    X12         CON2        1.0
    S1SET2E     'MARKER'    'SETEND'                      'E'
 S1 SISET3      'MARKER'    'SETORG'          REF3        1.0
    X13         OBJ         1.0               REF3        1.0
    X13         CON3        1.0
    X14         OBJ         10.0              REF3        5.0
    X14         CON3        1.0
    X15         OBJ         25.0              REF3        10.0
    X15         CON3        1.0
    X16         OBJ         -7.5              REF3        15.0
    X16         CON3        1.0
    S1SET3E     'MARKER'    'SETEND'                      'E'
RHS
    M0Z1        COS1        25.0
    MOZ1        COS2        50.0
    MOZ1        COS3        100.0
BOUNDS
  UP MOZ2       X1          100.0
  UI MOZ2       X2          20.0
  SC MOZ2       X3          10.0
  BV MOZ2       X4          1.0
ENDATA
```

where UP presents the continuous variable X1 with upper bound of 100.0, UI is presenting the general integer variable X2 with upper bound of 20.0, SC is the semi-continuous variable X3 with upper bound of 10.0 and default lower bound of 1.0, and X4 is the binary variable.

Appendix 2. Table of Test Problems

NAME	ROWS	COLS	BV	UI	SC	S1,2	IP obj.	LP obj.
EGOUT	98	141	55	—	—	—	568.1	149.58
MODGLOB	291	422	98	—	—	—	20740508.0	20430947.0
BEALE	171	303	38	—	—	11	100.00	100.00
MOZIGEN	11	19	1	1	1	3	95.16	93.43
S2MATRIX	51	450	—	—	—	25	157.9	225.88
SOUZA	191	194	11		—	11	100.00	100.00
KHB05250	101	1350	24	—	—	—	106940226.0	95919464.0
MICHA	90	127	64	—	63	—	5177.8	5167.0
BPGAS1	211	75	12	—	—	—	-6.65	6.79
GRAY2IN	35	48	24	—	—	—	202.30	185.5
GRAY9IN	63	96	48	—	—	—	280.95	256.01
HYPERPL	128	68	63	—	—	—	1336.5	333.33
SAMPLE2	46	67	21	—	—	—	375.0	247.0
MAT1A	176	202	10	—	—	22	5.68	0.0
MAT1B	262	435	16	—	—	25	6.7	0.0
MAT1C	615	676	38	—	—	38	9.7	0.0
MAT2A	102	1092	12	—	—	90	72037.8	66150.5
MAT2B	83	710	10	—	—	70	60733.0	55500.5
MAT2C	118	1515	15	—	—	100	76843.1	71950.5
MAT2D	70	714	14	—	—	50	41360.2	39019.1
MAT2E	66	411	11	—	—	40	34437.9	33209.2
MAT3A	114	1134	14	—	—	80	73864.5	64566.9
MAT3B	96	1515	10	—	—	70	72023.76	60728.3
MAT3C	92	732	6	—	—	60	56274.1	50199.9
PROBLEM1	534	632	7	—	—	3	53.36	54.40
PROBLEM2	652	643	18	—	—	3	53.36	54.40
PROBLEM3	652	643	18		—	3	51.02	52.45
PROBLEM4	652	643	144		—	3	53.36	54.40
PROBLEM5	700	678	18		—	4	-5.93	11.08
PROBLEM6	652	643	144		—	3	50.05 F	52.45
PROBLEM7	1500	1129	2		—	11	28.46	2934
AIR01	23	771	771	—	—	—	6796	6743.0
AIR02	50	6774	6774	—	—	—	7810	7640.0
AIR03	124	10757	10757	—	—	—	340160	338864.25
AIR04	8223	8904	8904	—	—	—	56138	55535.43

NAME	ROWS	COLS	BV	UI	SC	S1.2	IP obj.	LP obj.
AIR05	426	7195	7195	—	—	...	26402	25877.6
AIR06	825	8627	8627	—	—	—	49649	49616.3
BELL3A	123	133	39	32	—	—	878430.3	862578.6
BELL3B	123	133	39	32	—	—	11786160.6	11404143.8
BELL4	105	117	34	30	—	—	18541484.2	17984775.9
BELL5	91	104	30	28	—	—	8966406.4	8608417.9
BM23	20	27	27	—	—	—	34	20.57
CRACPB1	143	572	572	—	—	—	22199	22199.0
DIAMOND	4	2	2	—	—	—	INF	- 1.0
DSBMIP	1182	1886	160	32	—	—	-305.19	-305.1
EGOUT	*98*	141	55	—	—	—	568.10	149.58
ENIGMA	21	100	100	—	...	—	0.0	0.0
FIXNET3	478	878	378	—	—	—	51973	40717.01
FIXNET4	479	878	378	...	—	...	8936	4257.97
FIXNET6	479	878	378	—	—	—	3983	1200.0
FLUGPL	18	18	—	11	—	...	1201500	1167185.7
GEN	780	870	144	6	—	—	112313	112130.0
KHB05250	101	1350	24	—	—	...	106940226	95919464.0
L152LAV	97	1989	1989	—	—	—	4750 F	4656.3
LP41	85	1086	1086	—	—	...	2967	2942.5
ISEU	28	89	89	—	—	——	1120	834.68
MODGLOB	291	422	98	—	—	—	20740508	20430947.0
MISC01	54	83	82	—	—	—	563.5	5 7.0
MISC02	39	59	58	—	—	—	1690	1010.0
MISC03	96	160	159	—	—	—	3360	1910.0
MISC04	1725	4897	30	—	—	—	2666.69	2656.42
MISC05	300	136	74	—	—	—	2984.5	2930.9
MISC06	820	1808	112	—	—	—	12850.8	12841.6
MISC07	212	260	259	—	—	—	2810	1415.0
MOD008	6	319	319	—	—	—	307	290.9
MOD010	146	2655	2655	—	—	—	6548	6532.08
MOD011	4480	10958	96	—	—	—	-54558535	- 62121982.5
MOD013	62	96	48	—	—	—	280.9	256.02
NOSWOT	182	128	75	25	—	—	-43	-43.0
P0033	16	33	33	—	—	—	3089	2520.5

NAME	ROWS	COLS	BV	UI	SC	S1.2	IP obj.	LP obj.
P0040	23	40	40	—	—	—	62027	61796.5
P0201	133	201	201	—	—	—	7615	6875.0
P0282	241	282	282	—	—	._	258411	176867.5
P0291	252	291	291	—	—	—	5223.7	1705.1
P0548	176	548	548	—	—	—	8691	315.2
P2756	755	2756	2756	—	—	—	3124	2688.7
P6000	2095	5872	5872	—	—	___	-2350544	-2351871.3
PIPEX	25	48	48	—	—	—	788.2	773.75
RENTACAR	6803	9557	55	—	—	—	30356761	28806137.6
RGN	24	180	100	—	—	—	82.19	48.79
SAMPLE2	45	67	21	—	—	—	375	247.0
SENTOY	30	60	60	—	—	—	-7772	-7839.2
SET1AL	493	712	240	—	—	—	15869.7	11145.6
SET1CH	493	712	240	—	—	—	54537.7	32007.7
SET1CL	493	712	240	—	—	—	6484.25	1671.96
STEIN 15	36	15	15	—	—	—	9	7.0
STEIN27	118	27	27	—	—	—	18	13.0
STEIN45	331	45	45	—	—	—	30	22.0
STEIN9	13	9	9	—	—	—	5	4.0
VPM1	234	378	168	—	—	—	20	15.4

Explanation of columns:

NAME - name of the problem
ROWS - number of constraints in the problem, not including free rows
COLS - total number of variables in the problem
BV - number of variables that are binary
UI - number of general integer variables
SC - number of semi-continuous variables
S1,2 - number of special ordered sets of type one and two
IP obj. - best known integer solution to the problem,
INF, no integer feasible solution found
F, indicates that the given solution is not integer optimal
if there is no qualifier defined the given solution is optimum
LP obj. - optimal solution to the linear relaxation of the problem