Mixed Integer Programming on Transputers

Submitted by Peter Connard

in fulfilment of the requirements of the degree of PhD

School of Industrial and Business Studies

University of Warwick

December 1992

Summary

Mixed Integer Programming (MIP) problems occur in many industries and their practical solution can be challenging in terms of both time and effort. Although faster computer hardware has allowed the solution of more MIP problems in reasonable times, there will come a point when the hardware cannot be speeded up any more. One way of improving the solution times of MIP problems without further speeding up the hardware is to improve the effectiveness of the solution algorithm used.

The advent of accessible parallel processing technology and techniques provides the opportunity to exploit any parallelism within MIP solving algorithms in order to accelerate the solution of MIP problems. Many of the MIP problem solving algorithms in the literature contain a degree of exploitable parallelism. Several algorithms were considered as candidates for parallelisation within the constraints imposed by the currently available parallel hardware and techniques.

A parallel Branch and Bound algorithm was designed for and implemented on an array of transputers hosted by a PC. The parallel algorithm was designed to operate as a process farm, with a master passing work to various slave processors. A message-passing harness was developed to allow full control of the slaves and the work sent to them.

The effects of using various node selection techniques were studied and a default node selection strategy decided upon for the parallel algorithm. The parallel algorithm was also designed to take full advantage of the structure of MIP problems formulated using global entities such as general integers and special ordered sets. The presence of parallel processors makes practicable the idea of performing more than two branches on an unsatisfied global entity. Experiments were carried out using multiway branching strategies and a default branching strategy decided upon for appropriate types of MIP problem.

Acknowledgments

I would firstly like to acknowledge the support and guidance I received from my supervisor, Dr. Robert Ashford of the University of Warwick. His talent for keeping me on track when I began to stray from the point was especially appreciated, particularly during the writing of this thesis, which would have been even longer without his advice.

Many thanks are also due to my "step-supervisor", Professor Bob Daniel of the University of Buckingham, who invested numerous hours of his valuable time in various discussions with me. Much of the work for the thesis was carried out at the University of Buckingham, and the use of the facilities there was greatly appreciated.

Both Robert and Bob should also be acknowledged collectively under their "corporate identities" as Directors of Dash Associates Ltd. Dash provided me with the source programs for an early version of their commercial Branch and Bound code, without which the research would have been impossible.

The first two years of the research were funded by the Science and Engineering Research Council, after which Dash Associates provided me with consultancy work. I am grateful to all my paymasters for helping to keep body and soul together.

Lastly, and perhaps most importantly, I would like to thank my parents for not pressuring me too much to get a "proper job".

Table of Contents

List of Illustrations

List of Tables

1. Introduction

1.1. Background to Mixed Integer Programming

Since World War II, problems from a wide range of industries have been analyzed, attacked and solved using **Operational Research (O.R.)** techniques. Such techniques involve the creation and use of a model of the physical situation.

The most widely used and commercially successful O.R. techniques are those of **Mathematical Programming (MP).** Mathematical Programming models make the assumption that the controllable aspects of the physical situation are quantifiable. The situation is then modelled using mathematical **decision variables** which can take any value between some lower and upper bound (usually zero and infinity). The values of the decision variables in the final solution to the problem will provide a guide to the best course of action to take in the situation modelled.

A degree of simplification or an abstraction from reality is necessary in the creation of any model, and this must of course be reflected in the interpretation of the results obtained from using Mathematical Programming models.

Linear Programming (LP) is the most popular Mathematical Programming technique. LP models are concerned with the efficient allocation of some form of scarce resource to known activities in order to achieve a specified goal.

To this end, an LP model consists of:

- decision variables used to reflect decisions made as to the allocation of resources;

- constraints on the allowable (feasible) decisions; and

- some mechanism by which to measure the success of the solution.

The constraints of an LP problem must be <u>linear</u> (in)equations, consisting of a linear combination of decision variables, an equality or inequality and a constant term (possibly zero). Such an equation could be used to model the availability of a scarce resource for instance, with the linear combination of decision variables reflecting the actual resources used, and this being less than (or possibly equal to) a constant term which indicates the maximum availability of the resource.

An LP model also contains an **Objective Function** which acts as its measure of success. The objective function is a linear combination of decision variables which gives the actual quantified value of making the decision(s). This is usually expressed in terms of a total cost or profit, but any single objective may be used. Any allowed set of values of the decision variables will give a value to this objective function. The aim of using the LP model is usually to find an optimum value of this objective function (i.e. to maximise or minimise it), subject to satisfying the constraints on the problem.

The standard method used to solve LP problems, the Simplex Method (see [Dantzig, 1963]), essentially performs an ordered, although usually not exhaustive search through the possible combinations of decision variable values until the optimal solution is found. The search proceeds around the perimeter of the feasible region (which is imposed by the constraints on the problem) until the optimal point is found. This optimal point will always be at a vertex of the perimeter of the feasible region.

To summarise, the standard form of an LP problem consisting of m constraints and n variables is as follows:

Optimise	dy
Subject to	$Ey = b$
and	$y \geq 0$

where	$y = (y_n, s_m)$	is a vector consisting of n decision variables and m slack variables;
	$E = (E_n, I)$	is an m by (m+n) matrix consisting of constraint coefficients for the n variables and an m x m identity matrix for the slacks;
	$d = (d_n, 0)$	is a vector of the objective function coefficients for the variables; and
	b	is a vector of size m for the right hand sides of the m constraints.

LP models are used to solve problems from a great variety of industries, from agriculture and mining, to manufacturing and transportation, to chemical and petrochemical.

In certain instances however, it becomes apparent that a Linear Programming formulation of a problem is not sufficient to properly reflect reality. Choosing to carry out half each of two large projects needing capital investment and giving a return on completion for instance, would not maximise returns. Similarly, deciding to make two and a half multi-million pound jet aircraft using the facilities available would not be an acceptable solution. In either of these cases, solving the problem as an LP and then rounding off the numbers will not provide a provably optimal solution.

Thus the standard LP concept must be extended by the inclusion of 'discrete entities', i.e. variables (or sets of variables) whose values in an optimal solution must be ones taken from some discrete set. This extended problem is known as an **Integer Programming (IP)** problem. An IP problem may be further categorised as a **Pure Integer Programming (PIP)** problem if all decision variables must take values from a discrete set, or as a **Mixed Integer Programming (MIP)** problem if some, but not all, decision variables must take values from a discrete set.

Both the LP problem and the PIP problem can actually be considered as special cases of the MIP problem.

Consider the standard mathematical formulation of a Mixed Integer Programming problem, the category of problem on which this thesis will focus:

Optimise	$cx + dy$
Subject to	$Ax + Ey = b$
and	$x \geq 0$ and integer, $y \geq 0$

where x and y are vectors of decision variables and slack variables;

c and d are vectors of objective function coefficients;

A and E are matrices of constraint coefficients; and

b is a vector of right hand side values for the constraints.

An LP problem is simply the above formulation where the c vector and the A matrix are empty.

A PIP problem is the above formulation where the d vector and the E matrix are empty.

The mixture of discrete and non-discrete entities within a problem can be used to determine the most effective method of solution. For the purpose of this thesis, a MIP problem is considered to be one where there are a relatively low number of discrete entities within the decision variables.

Large hard MIPs are frequently encountered in real-world applications where the integer components often represent switching between radically different modes of operation. This is in contrast with problems of a **Combinatorial** nature, which contain a very high proportion of discrete entities, and which often have an underlying structure which may be exploited.

1.2. Computational Complexity of MIP Problems

The computational complexity of a problem gives an idea of how difficult the problem might be to solve. MIP problems have been shown to belong to a category of problems that exhibit a high degree of computational complexity. This means that MIP problems can be very difficult to solve in a reasonable amount of time.

The remainder of Section 1.2 will present background information relating to the theory of computational complexity. Various degrees of problem difficulty will be introduced, and an explanation given as to how problems are classified. This theoretical information is not central to an understanding of the rest of the thesis, but is presented for the interested reader.

Problem Complexity and Algorithm Efficiency

A problem can be said to consist of an infinite number of **instances** which are achieved by assigning numerical data to the problem parameters.

Since an LP or PIP problem is a special case of an MIP problem, it can be said that any instance of an LP or PIP is also an instance of an MIP. Thus, an algorithm which can solve all instances of an MIP problem can be used to solve all instances of the special case LP or PIP.

From this, we can conclude that MIP problems are **at least** as hard to solve as LP or PIP problems.

In order to decide on just how hard to solve MIP problems actually are, however, we must define some different potential categories of difficulty.

Problem complexity is usually measured in terms of the time taken to compute the result. Computation time is very often related to problem size (i.e. the number of variables and constraints of the problem), although this is not always the case (e.g. there are algorithms, such as the ellipsoid method for LPs ([Khachiyan, 1979, Gacs and Lovasz, 1981]), whose number of steps depend explicitly upon the magnitude of the numerical data).

To properly explain the concepts of computational complexity, the following definitions (from {Nemhauser and Wolsey, pp 118-119}) will be used.

Let the size of a problem instance be defined as the amount of information required to represent the instance.

An MIP is specified by data from the matrices c, d, A, E and b. This numerical data may be represented in a form close to the structure representing it when it is held on the computer by using a binary (0,1) alphabet.

In such a model, a positive integer x, where

$$2^n \le x < 2^{n+1}$$

is represented by the vector

$$(\delta_0, \delta_1, \ldots\ldots, \delta_n)$$

where

$$x = \sum_{i=0}^{n} \delta_i 2^i$$

and

$$\delta_i \in \{0,1\} \text{ for } i=0,\ldots\ldots,n$$

This representation of the data assumes that the initial data are integral or rational. Note that an extra digit is required to represent the sign of x and that rational numbers are represented by pairs of integers.

Let us now consider how to measure the computational time taken to solve a problem instance. The measure of computational time must be independent of the computer used, and so a good measure to use is a summation of all the basic operations carried out (e.g. addition, multiplication, comparison etc.). We assume that each basic operation is carried out in unit time.

A measure must also be found for the efficiency of the algorithm used to solve the problem. Consider an optimisation problem Z, consisting of an infinite number of instances (z_1, z_2 , ...) where the data for instance z_i is given by a binary string of length $l_i = l(z_i)$.

Let Q be an algorithm that can solve every instance of Z in finite time.

We assume that the running time of Q is specified by a function

$$g_Q : Z \to R_+^1 .$$

where R_+^1 is the set of non-negative real numbers.

We wish to express the running time of algorithm Q as a function of the length of the problem instance to be solved.

Two instances of a problem having the same length do not necessarily have the same running time of course. We must therefore use some statistic to aggregate the running times for all instances of the same length.

A common way to do this is to use a <u>worst case analysis</u>. Thus for all instances of size k, the running time is said to be

$$f_Q(k) = MAX\{g_Q(x_i) \mid l(x_i) = k\}$$

This approach gives an absolute upper bound on the running time, but can be misleading in certain cases (e.g. where only a small proportion of instances take a long time). Other measures requiring probability distributions of the instances could be used, but these would be more difficult to analyze and would require assumptions to be made about the underlying probability distribution.

Let us define f(k) to be O(g(k)) when there exists a positive constant w and a positive integer k' such that f(k) ≤ wg(k) for all integers k ≥ k'. This definition allows us to approximate f from above by a simpler function wg with w unspecified.

Using this definition, a polynomial

$$\sum_{i=0}^{p} v_i k^i \text{ is } O(k^p)$$

$$\text{since } \sum_{i=0}^{p} v_i k^i \leq wk^p \text{ for large integers } k$$

Algorithm Q is said to be a Polynomial Time Algorithm for problem Z if $f_Q(k)$ is $O(k^p)$ for some fixed p.

Any algorithm whose time complexity function cannot be bounded polynomially is regarded as an Exponential Time Algorithm.

The difference between these two types of algorithms can be seen quite dramatically when considering the solution of large problem instances (with large input length l). Examples of this difference are shown in Table 1.1 below (adapted from (Garey and Johnson, page 7)).

Note that the first three algorithms shown in Table 1.1 are polynomial, whereas the fourth and fifth algorithms are exponential.

Algorithm	Problem Size (input length l) 10	20	30	40	50
l	0.00001	0.00002	0.00003	0.00004	0.00005
l^2	0.0001	0.0004	0.0009	0.0016	0.0025
l^3	0.001	0.008	0.027	0.064	0.125
2^l	0.001	1.0	17.9 m	12.7 d	35.7 y
3^l	0.059	58 m	6.5 y	3855 c	2×10^8 c

(Times are in seconds unless followed by m = minutes, d = days, y = years, c = centuries).

Table 1.1: Comparison of Solution Times for Polynomial and Exponential Algorithms.

The effects of improved technology on the different types of algorithm are also of particular importance. Table 1.2 below (from (Garey and Johnson, page 8)) shows the largest problem that can be solved in one hour using the different types of algorithm if the hardware is speeded up.

N.B. X = size of problem solvable in 1 hour using current hardware.

	Hardware Speeded up	
Algorithm	100 times	1000 times
1	100X	1000X
1^2	10X	31.6X
1^3	4.64X	10X
2^1	X + 6.64	X + 9.97
3^1	X + 4.19	X + 6.29

Table 1.2: Relative Effects on Polynomial and Exponential Algorithms of Hardware Speedups.

It can be seen from Tables 1.1 and 1.2 why polynomial time algorithms are generally considered to be more desirable than exponential time algorithms. Indeed, many problems are not considered to have been "well-solved" unless a polynomial time algorithm has been found to solve them. Thus, for many theoretical purposes, a problem which cannot be solved by a polynomial time algorithm may be classified as intractable (although this is not a rigid rule in practice since, for instance, the Simplex algorithm for LP problems has been shown to have exponential time complexity [Klee and Minty, 1972]).

For theoretically tractable problems though, the first of the categories for problem complexity is defined as **P**, which is the class of problems that can be solved in polynomial time, (i.e. so that problem Z is in **P** only if there is a polynomial time algorithm for solving Z).

In order to introduce the next category of problem complexity, we shall now introduce the concepts of decision problems and deterministic and nondeterministic algorithms.

A decision problem is one which when solved will give either the answer "yes" or the answer "no".

A deterministic algorithm to solve an instance I of a decision problem, if given an input structure S will compute either the answer "yes" or the answer "no".

A nondeterministic algorithm to solve a decision problem can be thought of a having two stages, a "guessing" stage and a "checking" stage. Given a problem instance I, the first stage guesses some structure S. The instance I and the guessed structure S are then provided as inputs to the checking stage, which uses a deterministic algorithm to compute either the answer "yes" or the answer "no".

A nondeterministic algorithm is said to solve a decision problem Z if the following two properties hold for all instances $I \in D_Z$ (where D_Z is the set of all instances for problem Z):

If $I \in Y_Z$ (the set of instances of problem Z that give the answer "yes"), then there exists some structure S that, when guessed for input I, will lead the checking stage to respond "yes" for I and S.

If $I \notin Y_Z$ then there exists no structure S that, when guessed for input I, will lead the checking stage to respond "yes" for I and S.

As an example, a nondeterministic algorithm for the Travelling Salesman problem could be constructed using a guessing stage that, when given the data for the problem instance, gives an arbitrary sequence to the instance destinations. The checking stage would input the data for the instance and the guess, and verify whether the guess provided the best solution (e.g. by trying all combinations of routes).

A nondeterministic algorithm that solves a decision problem is said to operate in polynomial time if, for every instance $I \in Y_Z$ there is some guess S that leads the deterministic checking stage to respond "yes" for I and S within polynomial time.

We can now define a new category for problem complexity, i.e. the class **NP**. A problem Z is said to belong to the category **NP** if there is a nondeterministic algorithm that will solve it in polynomial time. By definition, all nondeterministic algorithms that operate in polynomial time will be members of **P**. Any deterministic algorithm that is in **P** can be used as the checking stage of a nondeterministic algorithm (although perhaps an entirely artificially created one).

Thus, the relationship between **P** and **NP** is probably such that **P** $\subseteq$ **NP** as shown in Fig. 1.1 below.

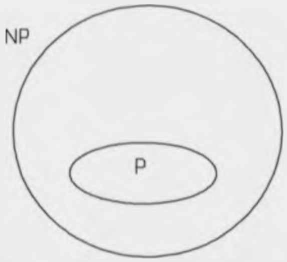


Fig. 1.1: The relationship between P and NP.

The class **NP** is especially important since those problems that are members of **NP** but not of **P** (if any such problems exist) are generally considered to be theoretically intractable.

Further categories of problem complexity can be introduced by considering the concept of problem reducibility. Two problems can be proved to be related to each other by "reducing" one problem to the other. This procedure involves providing a transformation process that maps any instance of the first problem into an equivalent instance of the second. Such a transformation provides the means for converting any algorithm that solves the second problem into a corresponding algorithm for solving the first problem.

As examples of such reductions, it may be shown that a number of combinatorial optimisation problems may be reduced to the general zero-one PIP problem ([Dantzig, 1960]), or that the Travelling Salesman problem may be reduced to the shortest path problem with negative edge lengths allowed ([Dantzig et al., 1966]).

Cook showed the importance of "polynomial time reducibility", i.e. a reduction for which the transformation is carried out in polynomial time ([Cook, 1971]). If the transformation process can be carried out in polynomial time, it can be proved that any polynomial algorithm to solve the second problem can be converted into a corresponding polynomial algorithm to solve the first.

In the same paper, Cook introduced a problem known as the "satisfiability" problem, which he proved has the property that every other problem in **NP** can be polynomially reduced to it.

Thus, if the satisfiability problem can be solved with a polynomial time algorithm, then so can every problem in **NP**, and if any problem in **NP** is intractable, then the satisfiability problem must also be intractable. He thus stated that the satisfiability problem is the "hardest" problem in **NP**, although other as yet undiscovered problems in **NP** might share this property.

Karp presented a collection of different combinatorial problems, including the Travelling Salesman problem, which when represented as decision problems were as "hard" as the satisfiability problem ([Karp, 1972]).

A class of problems consisting of the "hardest" problems in **NP** was thus created, and named **NP-complete**.

The relationship between the categories **P**, **NP** and **NP-complete** is shown in Fig. 1.2 below.

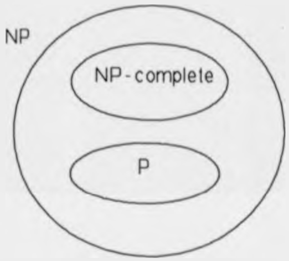


Fig. 1.2: The relationship between P, NP and NP-complete.

Note that any decision problem (whether it is a member of **NP** or not) to which we can transform an **NP-complete** problem will have the property that it cannot be solved in polynomial time unless **P**=**NP**.

There are of course, many problems which do not belong to **NP** at all. A further category of problem complexity may be considered to include such problems. The same techniques that are used to prove that a problem inside **NP** is **NP-complete** may be used to prove that a problem outside of **NP** is equally as hard.

Let **NP-hard** be the class of problems that are at least as hard as any member of **NP** since there is an **NP-complete** problem that can be polynomially reduced to it.

Having defined the different categories of problem according to computational complexity, we must now decide into which category MIP problems should be placed.

General and zero-one PIP problems have been proved to be **NP-complete** ([Garey and Johnson, 1979, Karp, 1972, Borosh and Treybig, 1976]). As stated previously, MIP problems are at least as hard to solve as PIP and LP problems. Thus it can be said that MIP problems are also **NP-complete**. Since MIP problems are **NP-complete**, the fact that solutions are being found in an acceptable time is as much due to luck in some cases as to a good algorithm. The algorithm commonly used to solve MIP problems (i.e. the Branch and Bound algorithm, [Land and Doig, 1960]) will, however, always eventually find the optimal solution, (although not necessarily in an acceptable time), since it enumerates all possible solutions.

1.1. Discussion of the Gist of the Thesis

Faster computer hardware has allowed the solution of more MIP problems in reasonable times, but there will come a point where the hardware cannot be speeded up any more. One way of improving the solution times of MIP problems without further speeding up the hardware is to improve the effectiveness of the solution algorithm used.

The arrival of accessible parallel processing facilities provides an opportunity to exploit any parallelism within the solution algorithm in order to provide an increase in solution speed for most MIP problems.

Effective algorithms for solving IP, and in particular MIP problems, can contain a considerable degree of exploitable parallelism. This has been demonstrated by the implementation of software used to solve pure integer programming (PIP) problems on a network of workstations ([Cannon and Hoffman, 1989]). The workstations used were loosely connected via Ethernet, but good performance was still achieved.

The practical solution of commercial MIPs can be challenging and expensive in terms of both time and effort. The development of a fast, inexpensive parallelised MIP-solving system would thus be of great benefit to many commercial users. To that end, a parallel algorithm has been developed, tested and implemented on a PC. This thesis discusses issues raised in the development and implementation of the algorithm, and reports computational results obtained from the solution of real IP problems.

The standard commercial codes for MIP (e.g. [IBM, 1988]) use a Branch and Bound approach ([Land and Doig, 1960]), although there are many different algorithms that can be used to solve MIP problems. Chapter Two introduces the different IP-solving algorithms that were considered for parallelisation and discusses their pros and cons when used on large MIP problems.

Before deciding on an algorithm to parallelise, it is necessary to consider the benefits and limitations of parallelisation. Chapter Three thus provides background on the development of parallel processing theory, algorithms and hardware. Different ways of exploiting the benefits of parallelism using appropriate parallel hardware are considered.

Chapter Four discusses the choice of MIP solving algorithm to be parallelised and the hardware chosen to implement the parallel algorithm. A full description is given of the initial parallelisation of the algorithm and of the process of its implementation using a small number of parallel processors. Computational results are reported for a number of test problems.

Chapter Five discusses the changes necessary to implement the parallel algorithm on a larger number of processors and reports computational results from a larger set of test problems.

Chapter Six reports computational results obtained by using the parallel algorithm to solve a set of test problems whilst using several different node selection strategies.

Chapter Seven discusses several theoretical extensions to the algorithm which are made to more fully exploit the parallel processing power available when attacking problems formulated using general integer variables or special ordered sets. Computational results are given using an appropriate set of test problems.

Chapter Eight draws conclusions from the results generated during previous chapters and makes several recommendations for future extensions to the research, including outlines of theoretical and implementational extensions to the present parallel algorithm for use on a very large number of parallel processors.

2. Methods of Solution for Mixed Integer Programming Problems

There are several different types of technique that can be used to solve MIP problems. These are categorised as:

(i) Cutting Plane Methods;
(ii) Partitioning Algorithms;
(iii) Group Theoretic Algorithms; and
(iv) Enumerative Methods.

This chapter will introduce various algorithms taken from these categories. The merits of each algorithm as a method for solving large MIP problems will be discussed.

It is worth noting that although several of the algorithms discussed in this chapter seem at first sight to be good candidates for parallelisation, this may not be the case in practice. Chapter Three thus considers the different parallel methods and hardware available for the implementation of a parallel algorithm, and Chapter Four discusses the final choice of algorithm and the method of parallelisation to be used.

2.1. Cutting Plane Methods

2.1.1. Background

The first cutting plane method for general use on any PIP was developed in 1958 ([Gomory, 1960]), although the idea had previously been proposed in 1954 for use in solving the Travelling Salesman category of PIP problem ([Dantzig et al., 1954]). Beale generalised the technique later in 1958 so that it could be applied to MIP problems ([Beale, 1958]).

The cutting plane concept is that the integrality constraints on the decision variables are removed, and additional linear constraints are generated and added to this LP relaxation. Each additional constraint added 'cuts off' part of the solution space of the LP, until eventually a solution can be found to the LP relaxation wherein all the decision variables that should take integer values do so.

As an example of this, in Fig. 2.1 below, we see the LP relaxation of a simple, two variable PIP problem. The optimal LP solution is such that one of the decision variables takes a continuous value (i.e. at point C, where $X_1=3$, $X_2=2.5$).

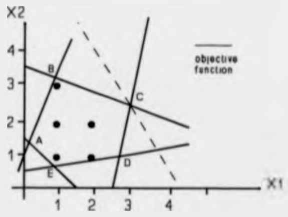


Fig. 2.1: Two variable PIP problem.

The solution space, at present shown by ABCDE, needs to have a section 'cut off' so that this answer is no longer feasible. The viable all-integer solution points of the LP are shown as dots. In order to produce an all-integer solution, one of these dots must become a vertex of the new solution space (since all optimum solutions to LP problems occur at a vertex of the solution space).

As can be seen in Fig. 2.2, by adding two new constraints to this LP, the feasible region can be reduced to the points AFGHE, so that the optimum LP solution is achieved by giving integer values to the decision variables (i.e. at point G where $X_1=2$, $X_2=2$).

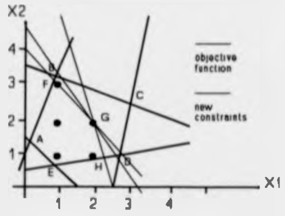


Fig. 2.2: Two variable PIP problem with cuts added.

It is important that the additional constraints, or 'cuts' do not remove any integer feasible solutions from the solution space.

2.1.2. The Gomory Fractional Cutting Plane Algorithm

The Gomory Fractional Cutting-Plane Algorithm (FCPA) makes use of Gomory cuts to reduce the solution space of IP problems without removing any integer feasible solutions. Appendix 1A shows that Gomory cuts are appropriate for this purpose.

A Gomory cut is of the form

$$\sum_{j \in R} f_{ij} x_j = g_i + s \qquad (6)$$

where, in the tableau for the optimal solution: R is the set of non-basic variables; x_j are the non-basic variables; f_{ij} is the fractional part of the coefficient of x_j in row i; g_i is the fractional part of the right hand side value of row i; and s is a positive slack variable for the new constraint.

The cut formula can be applied to any row of the LP solution to generate a new constraint row. It is only meaningful to apply the formula to a row which contains a basic variable not satisfying integrality.

The FCPA algorithm for solving a PIP is thus:

(1) Solve the LP relaxation of the PIP problem. If the solution is infeasible or all-integer, stop. Otherwise, go to (2).

(2) Choose a row from which to generate a new cut. Deduce a new cut and add it to the LP tableau, making the tableau infeasible. Go to (3).

(3) Reoptimise using the Dual Simplex method ([Lemke, 1954, Beale, 1954]). If the new solution is infeasible, the PIP has no solution - stop. If the new solution is integer feasible - stop. Otherwise go to (2).

This algorithm is finite as long as the <u>first</u> row with a non-integer constant component is used to generate cuts every finite number of iterations (see (Salkin p. 62)). The algorithm may be generalised for use with MIP problems, with the only change being that a more complicated cut formula must be used.

The tightest Gomory Mixed Integer Cut for MIP problems is of the form

$$\sum_{j \in R_1^+} f_{ij} x_j + \sum_{j \in R_1^-} \frac{g_i (1 - f_{ij}) x_j}{(1 - g_i)} + \sum_{j \in R_2^+} e_{ij} y_j - \sum_{j \in R_2^-} \frac{g_i e_{ij} y_j}{(1 - g_i)} \geq g_i$$

where

R_1	=	$\{j \mid j$ indexes the non-basics that are integers$\}$;
R_2	=	$\{j \mid j$ indexes the non-basics that are continuous$\}$;
x_{Bi}	=	basic variable for row i;
x_j	=	non-basic integer variables;
y_j	=	non-basic continuous variables;
a_{ij}	=	coefficient of integer variable x_j in row i;
e_{ij}	=	coefficient of continuous variable y_j in row i;
b_i	=	right hand side constant for row i;
gi	=	fractional part of b_i;
f_{ij}	=	fractional part of a_{ij};
R_1^+	=	$\{j \mid j$ is a member of set R_1, $f_{ij} \leq g_i\}$;
R_1-	=	$\{j \mid j$ is a member of set R_1 , $f_{ij} > g_i$ $\}$;
R_2^+	=	$\{j \mid j$ is a member of set R_2 , $e_{ij} > 0$ $\}$; and
R_2^-	=	$\{j \mid j$ is a member of set R_2 , $e_{ij} < 0$ $\}$.

Appendix 1B contains the derivation of this cut.

Various implementations of the FCPA algorithm are differentiated by the criteria they use when choosing a row on which a new cut will be based. Examples of popular criteria are the row containing the largest fractional variable, the row containing the first fractional variable, and a method where several different cuts are added and the Dual Simplex method is used to choose which cuts to keep and which to discard (see (Salkin) p. 58).

The FCPA algorithm has two main disadvantages when applied to MIP problems. Firstly, the FCPA algorithm cannot be proved to be finitely convergent unless the objective function only takes integer values (see (Nemhauser and Wolsey) p.374), which is by no means a obligatory feature of an MIP problem. The only way to ensure finite convergence is to scale the problem so that the objective function does only take integer values. This strategy is most unsatisfactory for computational purposes since either very large objective function coefficients will be required, unbalancing the matrix, or repeated scaling of the problem may be necessary.

Secondly, no feasible integer solution is obtained by the FCPA algorithm until the optimal solution is determined. There is no concept of a "reasonably good" solution associated with this algorithm. The Primal Cutting-Plane Algorithm (Ben-Israel and Charnes, 1962, Young 1965]) was designed to get around this problem. Unfortunately, due to the extremely large number of cuts necessary to solve a problem, only a few problems have ever been attacked using this method (e.g. [Padberg and Hong, 1980]).

2.2. Partitioning Algorithms

Another class of algorithms for solving IP problems is that of the partitioning algorithms. Such algorithms partition the variables or constraints of a problem into two or more categories and exploit the differences between the categories in order to solve the problem.

2.2.1. Benders' Decomposition

In 1962, Benders proposed an algorithm for use on any Mathematical Programming problem whose variables can be partitioned into two sets, so that if the variables in one set are given numerical values the overall problem reduces to an LP ([Benders, 1962]). One obvious usage of this is the MIP problem, where some variables may take continuous values but others must take integer values.

Benders' algorithm takes advantage of the fact that it can be shown, using duality theory, that any MIP problem can be rewritten as a PIP problem. The algorithm thus makes use of information from a reformulation of the MIP problem and from the dual of an LP relaxation of the MIP problem.

The MIP problem

$x_0^* = \text{MAX } x_0 = cx + dy$

$Ax + Ey \leq b$ (1)

$x \geq 0$ and integer, $y \geq 0$

can be relaxed, for any non-negative value, x_{val}, of the integer vector x, to give the LP:

$x_0^*(x_{val}) = cx_{val} + \text{MAX } dy$

$Ey \leq b - Ax_{val}$ (2)

$y \geq 0$

The dual of LP (2) is:

$u_0^*(x_{val}) = cx_{val} + \text{MIN } u(b - Ax_{val})$

$uE \geq d$ (3)

$u \geq 0$

The MIP (1) can also be rewritten as

MAX Z

$Z \leq cx + u^t(b - Ax)$ for every $u^t \in T$

$0 \leq v^q(b - Ax)$ for every $v^q \in Q$ (4)

$x \geq 0$ and integer

where $T = \{u^t | u^t$ is an extreme point of dual LP (3)}; and $Q = \{v^q | u^t + \theta v^q, \theta \geq 0$ is an extreme ray for some $u^t \in T\}$

This is Benders' reformulation of the original MIP problem (1). The derivation of this reformulation is discussed in Appendix 1C. Figure 2.3 below shows the extreme points and extreme rays of the dual LP (3).

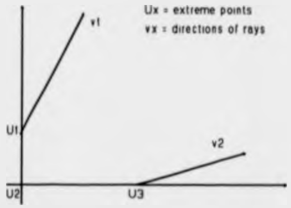


Fig. 2.3: The case where the Dual LP (3) is unbounded.

An apparent disadvantage of Benders' reformulation is that it generates a large number of constraints on the new PIP, since a constraint is generated for every extreme point and extreme ray of the problem. Benders' algorithm gets around this disadvantage by reformulating the MIP as a PIP and then solving the PIP using only a <u>subset</u> of the PIP constraints generated.

For any fixed non-negative value of x, x_{val} say, the dual LP (3) can be solved. Since (3) is more restricted than (1) (x being fixed), the value of its minimum solution can be used as a lower bound on the optimal solution of MIP (1). (If the dual LP (3) is unbounded, its value can be thought of as negative infinity). When dual LP (3) is solved, it provides an extreme point or ray, and thus an inequality for use in PIP (4) (the Benders' reformulation). If PIP (4) is solved using this single constraint, then its optimal solution is an upper bound on the best solution to the MIP (1). Solving PIP (4) also provides a new non-negative value for x_{val} which can be used to solve dual LP (3) again.

Thus, the dual LP problem (3) and the Benders' reformulation (4) can be repeatedly solved, providing better and better lower and upper bounds on the optimal solution to the MIP (1). When the lower bound on the MIP is equal to the upper bound on the MIP, the process terminates. The values of x are known, but to find the values of v, the LP problem (2) is solved with x taking their optimal values.

Benders' decomposition algorithm has been used successfully on MIPs ([Balinski and Wolfe, 1963], Childress, 1969, Geoffrion and Graves, 1974, Manne, 1971]).

2.2.2. Lagrangian Relaxation

Another useful partitioning algorithm is the Lagrangian relaxation method ([Everett, 1963]), which is again a method for obtaining bounds. The constraints of the problem are classified as either **simple** or **complicated**. The set of simple constraints is chosen so that it can easily be solved.

The complicated constraints are moved to the objective function (i.e. the complicated constraints are essentially dropped from the calculation process).

The Lagrangian relaxation method allows the solution of a PIP by enforcing only the non-negativity and integrality requirements on the variables.

Consider the PIP:

MAX	cx	
Subject to	$Ax \le b$	
and	$x \ge 0$ and integer	

This may be rewritten as

MAX	cx	
Subject to	$A^s x \le b^s$	(simple constraints)
	$A^c x \le b^c$	(complicated constraints)
and	$x \ge 0$ and integer	

Let λ be a column of non-negative "multipliers".

Then if $x=x^0$ solves

$$\max_{x} \quad cx + \lambda(b^c - A^c x)$$

Subject to $\quad A^e x \leq b^e$

$\quad x \geq 0$ and integer

it also solves the PIP with b^c replaced by $A^c x^0$. Thus, if λ is chosen so that the optimal solution x^0 gives $b^c = A^c x^0$, the original PIP has been solved.

The proof of this, (which is also applicable to MIP problems), adapted from (Salkin, pp 416-419) is as follows:

As x^0 gives the maximum value to the objective function, we can say that

$$cx + \lambda(b^c - A^c x) \leq cx^0 + \lambda(b^c - A^c x^0)$$

Therefore, for all $x \geq 0$ and integer,

$$cx + \lambda(A^c x^0 - A^c x) \leq cx^0$$

So for all non-negative integer solutions to $A^c x \leq A^c x^0$, the previous inequality is true. Thus, since $A^c x^0 - A^c x \geq 0$ and $\lambda \geq 0$, this implies that $cx \leq cx^0$.

Hence x^0 solves the PIP

MAX	cx
Subject to	$A^c x \leq A^c x^0$
	$A^e x \leq b^e$
and	$x \geq 0$ and integer

Thus, if the multipliers (i.e. the λ), are chosen such that $b^c = A^c x^0$, the IP can be solved without the inequality constraints. The difficulty obviously comes in finding λ such that $b^c = A^c x^0$.

The overall Lagrangian Relaxation Algorithm works as follows:

(1) Select λ and find x^0 and hence $A^c x^0$.
(2) If $A^c x^0$ is close enough to b^c, stop. Else, go to (1).

Many different methods have appeared in the literature for choosing the λ values (e.g. [Brooks and Geoffrion, 1966, Fox and Landi, 1970, Nemhauser and Widhelm, 1971, Geoffrion, 1972]).

Lagrangian relaxation has been a successful way of solving combinatorial and MIP problems, including the Travelling Salesman Problem ([Christofides, 1970, Held and Karp, 1970, 1971]), vehicle routing problems ([Stewart and Golden, 1984]) and plant location problems ([Sridharan, 1991]).

2.3. Group Theoretic Algorithms

Gomory, as well as developing the fractional cutting plane algorithm, also proposed the use of Group Theoretical Algorithms to solve IPs ([Gomory, 1965, 1967, 1969]). He showed that by relaxing the non-negativity (but not integrality) constraints on certain variables, any PIP can be represented by a minimisation problem defined on a group.

Appendix 1D shows that the PIP problem

Maximise	cx
subject to	Ax = b
and	x ≥ 0 and integer

may be rewritten as

Minimise $$\sum_{j=1}^{n} \sigma_j x_{J(j)}$$

subject to $$\sum_{j=1}^{n} \boldsymbol{\alpha}_j x_{J(j)} \equiv \boldsymbol{\alpha}_0 \quad \text{(mod1)}$$

and $x_{J(j)} \geq 0$ and integer (j=1,....,n)

where $x_{J(j)}$ is the jth ($1 \leq j \leq n$) non-basic variable, $\sigma_j \geq 0$ are the costs and each column $\boldsymbol{\alpha}_j$ ($j=0,1,\ldots,n$) satisfies $0 \leq \boldsymbol{\alpha}_j < \boldsymbol{\pi}$ (where $\boldsymbol{\pi}$ is a column of ones).

This is referred to as the **Group Minimisation Problem**. By solving this problem, we may also be able to solve the original PIP problem.

Several algorithms have appeared in the literature for use in solving the Group Minimisation Problem (GMP). These algorithms include a dynamic programming algorithm, an enumeration method and a shortest route algorithm used for attacking the problem as a network.

These algorithms have a common basic structure. The GMP problem is solved and the minimum value for the objective function is found. If the basic variables happen to take non-negative values then the original PIP problem has also been solved. If one or more of the basic variables take negative values, the GMP has to be solved again to find the solution which gives the smallest value of the objective function whilst the basic variables take non-zero values.

The Dynamic Programming approach to this situation is to describe a set of recursive relationships to find the rth best solution to the Group Minimisation Problem (see [White, 1966]). By proceeding through these solutions, it is hoped that eventually a solution will be found which will solve the original PIP problem. The recursion expressions are very complicated however, and difficult to implement.

The enumerative approach to the situation is to explicitly or implicitly examine all possible integer solutions to the Group Minimisation Problem ([Shapiro, 1968]). This is achieved by successively adding bounds of the form $x_{j(i)} \geq K$ $(j=1,\ldots,n)$, where K starts at zero and is incremented by one until it reaches any upper bound on the variable $x_{j(i)}$. This process will eventually generate all possible integer solutions to the Group Minimisation Problem, some of which it is hoped will contain non-zero basic variables and thus satisfy the original PIP problem. In order to find the optimum solution to the original PIP, it is necessary to enumerate all the possible integer solutions of the GMP so that the allowable solution to the GMP with the minimum objective function value can be found. It is hoped that much of this enumeration will be done implicitly once an allowable solution has been found. Each allowable solution found can be used to place a bound on any future candidate solutions, thus allowing the implicit enumeration of certain situations which cannot possibly give a solution to better the present best.

This enumeration method is very similar to the Branch and Bound enumeration algorithm described in section 2.4.1., although its search process is less well directed since there is no guide as to whether the individual GMP situations examined will or will not give solutions with non-zero basic variables.

The shortest path approach makes use of the enumerative algorithm previously mentioned along with the fact that the GMP may be represented as a shortest route problem. (Salkin, pp 364-368) discusses this representation as follows:

Let the set of vectors generated by additions (modulo 1) of the n $\tilde{\alpha}_j$s in the Group Minimisation Problem be denoted by $G(\tilde{\alpha})$.

Each element in the group $G(\tilde{\alpha})$ corresponds to a distinct node. Let a directed arc (which may only be traversed in the direction it indicates) (i,k) join two nodes representing group elements g_i and g_k whenever $g_k - g_i$ (mod 1) is equal to some $\tilde{\alpha}_j$ $(j \geq 1)$.

Traversing an arc from node i to node k therefore corresponds to incrementing $x_{J(j)}$ by one, and so the cost σ_j is assigned to arc (i,k).

The Group Minimisation Problem may be reduced to the problem of finding the cheapest (or shortest) route from the node representing the zero element of the group (denoted by g_0) to the one representing the right hand side, $\tilde{\alpha}_0$.

A route is a sequence of directed arcs originating from g_0, and if an arc (i,k) is used in the route it means that $x_{J(j)}$ is increased by one, where $\alpha_j \equiv g_k - g_i \pmod 1$. Initially (at node g_0) all $x_{J(j)}$ are 0).

Thus, since it can be shown that the GMP can be represented as a shortest route problem, each of the separate GMPs constructed by adding bounds of the form $x_{J(j)} \geq K$ $(j=1,\ldots,n)$ can be solved using a shortest route algorithm (e.g. [Dijkstra, 1959]).

All of the above methods may be extended to the MIP case by only considering the solutions to the GMP that give integer values to the discrete variables.

None of these Group Theoretic methods however is particularly efficient in solving MIP or PIP problems, since the required search through the many different Group Minimisation Problems is either conducted by complex and computationally difficult recursion relations or by a semi-blind search through all the possible problems.

2.4. Enumerative Techniques

Enumeration algorithms for solving IP problems attempt to enumerate, either explicitly or implicitly, all the possible solutions to the IP problem. The optimal solution is defined as the feasible solution from the list of possible solutions found that maximises or minimises the objective function.

All enumeration algorithms consist of a method of keeping track of the solutions considered so far and of "point criteria" which indicate situations where certain related integer points cannot yield better solutions than the present best (incumbent) solution. Such point criteria make use of the integrality and constraint requirements of the IP problem to implicitly enumerate large numbers of points at once. The efficiency of an enumerative algorithm will depend very heavily upon the effectiveness of its point criteria.

The two major categories of enumerative algorithm are those of Branch and Bound Enumeration and Direct Search Enumeration.

2.4.1. Branch and Bound Enumeration

The most commonly used algorithm for solving IP problems is the Branch and Bound Enumeration Algorithm ([Land and Doig, 1960, Dakin, 1965]). The enumeration process is directed by criteria for **separation**, **relaxation** and **fathoming**.

The integrality constraints on the decision variables are initially ignored (i.e. they are **relaxed**) and the IP problem is solved as an LP (usually by using the Simplex Method [Dantzig, 1951]). The LP relaxation may prove to be infeasible or unbounded, in which case the original IP problem is also infeasible or unbounded respectively. Otherwise, if there is an optimal, (and therefore feasible) solution to the LP relaxation, a check is made to see if the supposedly-integer decision variables in this solution all take integer values. If so, then the optimal solution has also been found to the IP problem. If one or more of the supposedly-integer decision variables take fractional values in the optimal LP solution, then further work (in the form of **separation** and **fathoming**) must be done to find the optimal IP solution.

The Branching Process

The first step is to **separate** one of the supposedly-integer decision variables that is taking a continuous value, and by doing so effectively divide the original IP problem's solution space into a number of sections (thus creating several new IP problems to be solved).

The process of separating a variable is called **branching** upon that variable.

As an example of this, shown in Fig 2.4, if the solution given to an LP relaxation of an IP problem is such that a supposedly-integer decision variable, x_1 say, takes a continuous value C, then the integer feasible answer must lie in the solution space of one of two LPs where in the first, the simple bound

$x_1 \geq |C| + 1$

is placed on decision variable x_1, and in the second the bound

$x_1 \leq |C|$

is placed on decision variable x_1.

The solution space removed (i.e. where $|C| < x_1 < |C| + 1$) would not have produced an integer feasible answer anyway.

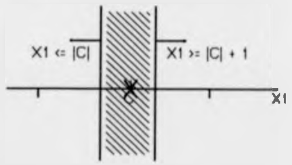


Fig. 2.4: Branching upon a variable.

The next step in the algorithm is to add the two new IPs created by the separation to a list of candidates for solution. A choice is then made from the candidate list, and the appropriate LP relaxation of the original IP (with the new bounds indicated by the separation) is solved. The solution to this LP relaxation will indicate the next step of the algorithm. Either the separation of a different variable (i.e. further enumeration of the original IP solution space) will be indicated, or the LP relaxation will be said to be **fathomed**. A fathomed LP relaxation will either provide an integer feasible solution, or will be infeasible, or will provide a solution that can be shown to be worse than the overall optimum).

The Branch and Bound algorithm is convergent, as it will (eventually) fathom all the IP problem's solution space. This is because, in the process of separating variables, (and thus further enumerating the original solution space of the IP problem), more and more useless parts of the solution space are removed. The unenumerated areas that remain will be further and further constrained by bounds on the variables as the search continues (and more and more LPs with more and more bounds on their decision variables are generated). Thus, as the algorithm leaves smaller and smaller areas of the solution space unenumerated, a particular area must eventually yield either an integer feasible answer or indicate that this part of the solution space is invalid (i.e. there are no feasible answers to be found within it that are better than the current best feasible answer). In either of these cases, that part of the solution space is said to be **fathomed**, since we no longer need to consider it or any further enumeration of it.

Of course, if several decision variables are taking continuous values in an optimal LP solution that has just been obtained, an intelligent choice of **branching variable** must be made.

Branching Variable Selection

Good branching variable selection (i.e. effective IP problem solution space partitioning) is important if the search through the solution space is to be carried out efficiently.

A common way of choosing branching variables in practice is to use **user-specified priorities**, i.e. to order the variables as part of the input to the algorithm. This method can be very powerful in practice, especially in cases where the IP problem is highly structured. For example, a variable indicating whether or not a project is to be done or not should obviously be branched upon before variables reflecting details of the project if it is undertaken.

Other methods for choosing the branching variable make use of **estimated degradations** which are calculated when an LP relaxation has just been solved and further branching is indicated. An estimate is made of the degradation to the LP solution caused by making branches on each of the integer-constrained variables that are taking continuous values. The choice of branching variable is then made based on a comparison of the degradations the various branches will cause to the LP solution.

If a supposedly-integer decision variable x_i takes a continuous value C_i, then two branches are usually considered whereby either a lower bound of the form

$x_i \geq |C_i| + 1$

or an upper bound of the form

$x_i \leq |C_i|$

will be placed on decision variable x_i.

Let imposing a new lower bound of $|C_i| + 1$ on x_i decrease the objective function by the "up penalty" of p_i for every unit increase from the current value of x_i. (This is an "up penalty" since the variable value must be moved up to the new lower bound).

Similarly, let the "down penalty" incurred by placing an upper bound of $|C_i|$ on x_i be p_i^+ for every unit decrease from the current value of x_i.

If we define

$C_i = |C_i| + f_i$

then an estimated decrease of

$D_i^+ = p_i^+ f_i$

would be expected if an upper bound was imposed and a decrease of

$D_i^- = p_i^-(1-f_i)$

expected if a lower bound was imposed.

Forrest, Hirst and Tomlin give a method for estimating penalties p_i^- and p_i^+, but state that this method is only useful for attacking small problems (see [Forrest et al., 1974]). A more useful method, as used by the commercial optimiser XPRESS-MP ([Dash Associates, 1989]), is summarised in Appendix 1G.

Once the estimated degradations D_i^- and D_i^+ have been calculated for all the candidate variables, a choice must be made as to which one to branch upon based upon the degradation values.

The most common way of choosing the branching variable is to use the **maximum integer infeasibility criteria**

$$\underset{j \in N^1}{MAX} \; MIN \; (D_j^+, D_j^-)$$

where N^1 is the set of all the candidate variables for branching, x_i.

The idea behind this method is that a variable whose smallest degradation is the largest degradation overall is the most important for achieving integrality (i.e. it is the variable whose value has the most drastic effect).

Node Selection

A selection must also be made at each iteration of the algorithm from the growing list of candidate LPs with appropriate bounds on the decision variables. Such a candidate LP problem may be represented as a **node** on a **solution tree** (see Fig 2.5).

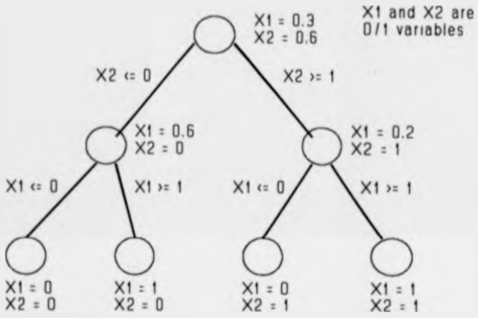


Fig. 2.5: Solution tree for a PIP problem.

If the algorithm is to proceed efficiently, **Node Selection** (i.e. the selection of which part of the solution space to consider next) must also be considered very carefully.

There are two options for node selection: either the strategy is worked out before the problem is attacked (using a **set of a priori rules**); or an **adaptive strategy** is used which decides at each iteration using the current information.

A commonly used a priori rule is **depth-first search plus backtracking** (also known as Last In, First Out (LIFO)). This involves keeping track of which node was considered at the last iteration. If that node was not fathomed (i.e. cut off), then one of its two descendants is considered at this iteration. When a node is fathomed, the algorithm backtracks along the solution tree towards the root until it finds the first node (if any) that has a descendant that can be considered.

Another example of an a priori rule is the **breadth-first search**, which considers all the nodes at a particular level of the tree before considering any from lower levels.

The Bounding Process

The Branch and Bound algorithm continues the branching process until an integer feasible solution is found. At this stage, the point criteria of the Branch and Bound algorithm may be used to implicitly enumerate some of the possible solutions, i.e. the **bounding** part of the algorithm can take place.

The bounding process uses the objective function value as a point criterion to discard some of the present list of candidate LP relaxations. Placing upper or lower bounds on the allowable values of a decision variable of an LP reduces the number of values the variable can possibly take. The optimal solution value of the LP which has extra bounds can only **at most** be as good as the solution value of the same LP without the extra bounds. In fact, the new objective function value is likely to be worse than that of the LP without the extra bounds on the variable.

Thus, any LP relaxation that at present gives an objective function value of worse than the best integer feasible solution so far is not worth considering further (since the objective function value will only worsen when extra bounds are placed on its variable values).

2.4.2. Direct Search Enumeration

The direct search enumeration method (also known as the additive algorithm) was first proposed in 1963 as a scheme for solving zero-one PIP problems ([Balas, 1963, 1965]). The method was subsequently elaborated upon, allowing the solution of zero-one MIP problems ([Glover, 1965, Lemke and Spielberg, 1967]) and finally extended to the solution of general MIP problems ([Driebeek, 1966]).

The basic algorithm proposed by Balas, for use on zero-one PIP problems can be shown by a tree composed of nodes and branches. Two nodes joined by a branch differ only in the state of one variable (the three possible states being set to one, set to zero or free). A new node on the tree is created by fixing a chosen free variable to one (known as taking a **forward step**). A node is revisited (i.e. a **backwards step** is taken) by fixing the previously free variable to zero (as seen in Fig. 2.6 below).

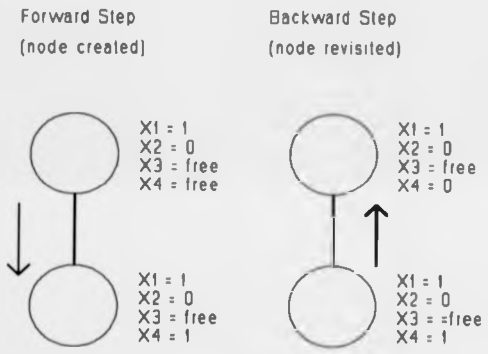


Fig. 2.6: Forward and Backwards Steps.

A depth-first search of the tree is commonly followed until there are no free variables in its last node. One or more backwards steps are then taken until a forward step may be taken to create a new node or until all free variables have been fixed to zero.

Defining the level of a node by the number of variables fixed to one, and the point x^l as the node with l variables fixed at one, the basic approach is as follows (as adapted from (Salkin, pp 211-218)):

1) Fix a free variable x_k from x^l (initially $x^l = x^0$) at value one.
2) Solve the subproblems in the remaining free variables.
3) Fix x_k at value zero (also known as cancelling x_k at level l).
4) Solve the subproblem in the remaining free variables with x_k fixed at zero.

At any stage in the search for the solution to a problem in n variables, a node will contain l variables fixed at one, c variables cancelled at zero, and there will be a subproblem in f=n-(l+c) free variables to solve. Corresponding to the f currently free variables, there are f permissable branches from x^l, as shown in Fig. 2.7.

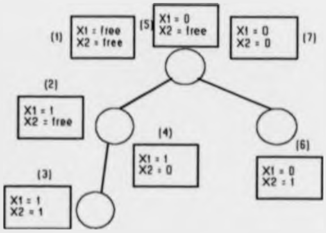


Fig. 2.7: An example of Direct Search Enumeration.

The process will eventually enumerate all (2^n) possible zero-one vectors, x, but for efficiency, it is obviously desirable to implicitly enumerate as much as possible.

Appendix 1E describes some of the implicit enumeration criteria appearing in the literature.

A good point algorithm which indicates which node to branch upon is also very important to any enumeration algorithm.

One good point algorithm for the direct search enumeration algorithm is the Balas Test, as follows:

The subproblem at point x^1 is problem P

Minimise $\quad z = cx^1$

Subject to $\quad Ax^1 \leq b$

and $\quad 0 \leq x^1 \leq \pi$

where $x_j = 0$ or 1 for $j=1,\ldots,n$ and π is a column of ones.

If the c variables that are fixed at zero are dropped and the l columns whose variables are fixed at one are subtracted from the right hand side vector b, then P can be rewritten as

Minimise $\quad z - z^1 = c^f x^f$

Subject to $\quad A^f x^f \leq b^1$

and $\quad 0 \leq x^f \leq e$

where $x_j = 0$ or 1 for $j \in F$.

$x^f = (x_j)$ is the vector corresponding to free variables, F is the corresponding set of indices in x^f, c^f and $A^f = (A_j)$ are costs and columns of A, b^1 is the updated right hand side, and z^1 is the sum of the costs of the l variables fixed to one.

If a free variable x_j is set to one, the constant term b_i^1 in each constraint i becomes $b_i^1 - a_{ij}$.

When $b_i^1 - a_{ij} \geq 0$, constraint i is satisfied.

Thus, a measure of the total "constraint infeasibility" is given by

$$v_j = \sum_{i \in M_j} (b_i^1 - a_{ij})$$

where $M_j = \{i \mid (b_i^1 - a_{ij}) < 0,\ i=1,\ldots,m\}$ and by $v_j = 0$ if M_j is empty.

To reach or return to a zero-one solution, it is reasonable to branch on the variable which maximises v_j.

The extension of the direct search enumeration algorithm to zero-one MIP problems, as proposed by Lemke and Spielberg, is as follows:

A search enumeration is carried out over the integer variables. Each time a node is explicitly examined, an LP in only the continuous variables is solved (i.e. prior to the Simplex iterations, the free variables are fixed to zero). Thus, a feasible LP always produces a mixed integer feasible solution.

This suggests that 2^n LPs must be solved, and that the algorithm could not possibly be efficient unless only a small number of integer entities are present.

There is a way around this problem however. It is possible to derive constraints, using only the zero-one variables, that will be valid at any node. An IP point algorithm could be applied to these constraints, possibly resulting in many cancellations or free variables being set to one (i.e. lots of implicit enumeration). This method is discussed in Appendix 1F.

The extensions of the Lemke and Spielberg method to general MIP problems is reasonably obvious, involving only a more complicated method of fixing and keeping track of variable values.

Having said this, both the Lemke and Spielberg algorithm for MIPs and the Driebeek algorithm for general MIP problems, which is based on it, are only useful if the integer decision variables can only take a narrow range of values. In the case of the Lemke and Spielberg algorithm, this is because the usefulness of many of the implicit enumeration criteria is reduced by the addition of continuous variables (see (Salkin p. 229) for example). In the case of the Driebeek algorithm, which builds on the previous work based around zero-one variables, problems occur because general integers are represented by a set of zero-one variables.

As an example of this, a general integer variable x_i with a lower bound of zero and an upper bound of n would be represented as follows:

$$x_i = \sum_{j=1}^{n} \delta_{ij} \qquad \delta_{ij} = 0 \text{ or } 1;\ j = 1,2,\ldots,n$$

Thus, the Driebeek algorithm becomes impractical for use on large MIP problems with more than a few integer variables, especially if the integer variables have large upper bounds.

2.5. Current Implementation of MIP Solution Algorithms

With the exception of the OSL package mentioned below, all commercial optimisation packages (e.g. XPRESS-MP, SCICONIC, etc.) use the Branch and Bound Enumeration Algorithm to solve IP problems.

The Optimisation Subroutine Library (OSL) available from IBM ([IBM, 1990]) optionally allows the use of Cutting Plane routines to create a Branch and Cut algorithm, although only for zero-one MIP problems. The Branch and Bound search is carried out as normal, except that the user can specify occasions on which cuts are generated and added to the LP relaxations attacked, hopefully providing better solutions.

3. Introduction to Parallel Computing

Now that various MIP-solving algorithms have been introduced, it is necessary to consider how they might be parallelised and on what hardware a parallel algorithm might best be implemented.

3.1. Background

The notion of exploiting some form of parallelism in computer design is nearly as old as the idea of the computer itself. One early reference to parallelism in computer design for instance, dates from October 1842, contained in "Sketch of the Analytical Engine Invented by Charles Babbage" by L.F. Menabrea ([Kuck, 1977]).

Different breakthroughs in technology over the years have actually led to both the implementation and removal of various parallelised processes within computer hardware. For instance, before the advent of electronic components in computers in the 1940s, the bit by bit addition of 32-bit numbers was carried out in parallel. Babbage actually rejected serial addition for his difference engine because of the long execution time involved ([Morrison and Morrison, 1964]). The introduction of electronic components allowed such a great improvement in performance that serial addition became a possibility. The increase in performance was, in fact, so dramatic (somewhere between 1000 to 10000 times), that serial addition became the norm in order to exploit the benefits of using less equipment by only processing one digit at a time.

Despite some such occasional "setbacks", more and more parallelism has been gradually introduced into computer design since the 1970s.

The introduction of the microprocessor at the end of the 1970s and of very large scale integration (VLSI) technology (i.e. technology allowing the production of a chip containing from ten thousand to a million microprocessors) in the 1980s have allowed the implementation of many (though not all) parallel architectures that had previously only been theoretically possible.

3.2. Classifications of Parallel Computer Architecture

An initial classification of computer architecture may be achieved by using Flynn's Taxonomy ([Flynn, 1966]), which is based on the concepts of instruction stream and data stream.

An **instruction stream** is a series of instructions performed by a computer. A **data stream** is a sequence of data used to execute an instruction stream.

Flynn categorizes a computer architecture by "the maximum possible number of simultaneous instructions or data being in the same phase of execution at the most constrained component of the organization".

There are four classifications of computer architecture arising from Flynn's categorizations, each of which has yielded actual hardware. Flynn's classifications are:

Single Instruction stream, Single Data stream (SISD) architecture;
Single Instruction stream, Multiple Data stream (SIMD) architecture;
Multiple Instruction stream, Single Data stream (MISD) architecture; and
Multiple Instruction stream, Multiple Data stream (MIMD) architecture.

Serial computers may easily be classified using Flynn's categorisations. Most serial computers fall into the SISD category. Indeed, true parallelism is not possible using SISD architecture, although fast SISD machines may give the appearance of parallelism by supporting multitasking (i.e. by switching back and forth between two jobs very quickly).

The small number of serial computers that do not fall into the SISD category have been specially designed to implement image processing systems. They operate in such a way that they fall into the MISD category since different elements of a single data stream are in effect simultaneously being acted upon by multiple operations or instructions. The Genesis 2000 system ([Sternberg, 1985]) is such a system currently on the market.

Unfortunately, Flynn's taxonomy is not sufficient on its own to properly classify the various types of parallel hardware available. Although different types of parallel hardware have been developed that do fall into the SIMD and MIMD categories, other categories are necessary to fully reflect the variety of parallel computer designs in use.

The many different types of parallel architecture have arisen in response to the additional problems which occur when implementing parallel algorithms. Implementing a parallel algorithm typically involves at least as much work as implementing a serial algorithm plus further work in controlling and coordinating the tasks to be performed in parallel.

The major aim when developing any algorithm, be it parallel or serial, is to obtain the correct result. If an algorithm does not give the correct result, then there is obviously no point worrying about lesser issues such as algorithm efficiency or speed.

In the case of parallel algorithms, special care must be taken, especially when calculations are being carried out in parallel using the same data.

Consider a bank system performing daily updates to current account balances. Deductions made as a result of standing orders could be performed by one processor while another processor was dealing with deductions made as a result of withdrawals made via automatic cash machines. If both processors were allowed access to a particular current account record simultaneously, an error in the final calculation would result, with the result of only one of the calculations being reflected, and not the combined result of both calculations.

The coordination of such parallel tasks within a parallel algorithm is usually achieved by parallel hardware making use of synchronous or asynchronous organisation techniques.

Synchronous coordination involves forcing all operations to be performed simultaneously and in a manner that removes the dependency of one task on another. For instance, a lock could be placed on the current account record by any task using its data for calculations. Thus, any other task requiring the data would have to wait until the record was released from the lock.

Asynchronous coordination is a looser form of control where processors operate freely on tasks without regard for global synchronisation. Asynchronous algorithms contain explicit flow control in order to coordinate parallel processes. For instance, all standing order calculations could be carried out at night at a time when the automatic cash machines are not operational.

Parallel algorithm design categories are primarily classified by their use of synchronous or asynchronous coordination techniques. The major categories of parallel algorithm design at present are as follows:

<u>Synchronous</u>	<u>Asynchronous</u>
Vector/array	MIMD
SIMD	Reduction
Systolic	

<u>3.2.1. The Vector/Array Paradigm</u>

This synchronous approach maintains algorithm integrity whilst improving algorithm performance by breaking down the overall task into subtasks which must be performed in a given order. In effect, a pipeline is created for these subtasks in that the output data from one subtask in the pipeline becomes the input data for the next subtask (see Fig 3.1).

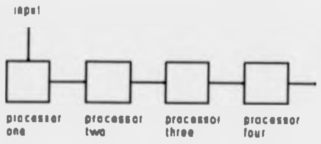


Fig. 3.1: Pipeline example.

The key to improving performance with this method is to keep the pipeline full (i.e. keep all the processors busy). Although the individual subtasks take as long to perform, several pieces of input data may be fully processed through the pipeline in the time it would have taken for a single processor to perform all subtasks on one piece of input data. A certain amount of communication overhead will occur when using this method, which will reduce the overall performance somewhat, but modern communication technology ensures that this overhead will be minimised.

The first vector machines were invented in the 1970s (i.e. the CDC STAR-100 [Hintz and Tate, 1972] and the TI ASC [Watson, 1972]). The pipelining technique has been used since then to attack numerical problems such as matrix and vector calculations. Thus, the pipeline approach is often known as vector/array processing. The more successful implementations of pipelined computers have been the CDC CYBER 205 ([CDC, 1983]), the ETA-10 ([Fazio, 1987]), the CRAY series of supercomputers ([Cray 1976, 1985 Chen, 1984, Russell, 1978]), the Floating Point Systems 164/MAX ([Charlesworth and Gustafson, 1986]) and the IBM 3090 Vector Facility ([Moore et al., 1987]). The FUJITSU VP 100/200 ([Motegi et al., 1984]), HITACHI S-810/10 ([Nagashima et al., 1984]) and the NEC SX1/SX2/SX3 ([Watanabe, 1984, 1987]) vector computers have further developed the ideas behind the CYBER 205 and CRAY architectures. A new market for vector computers was discovered in the late 1980s when graphics supercomputers were marketed by Stellar ([Sporer et al., 1988]) and Ardent ([Miranker et al., 1988]). More general references concerning vector/array processing and supercomputing are ([te Riele et al., 1987]), ([Hwang and Briggs, 1984]) and ([Dongarra, 1987]).

3.2.2. The SIMD paradigm

The next synchronous approach is that of the SIMD paradigm, where all processors perform the same task simultaneously on different data, or else remain idle. The efficiency of the SIMD paradigm depends on how well the data for the overall problem can be partitioned across the available parallel processors. Having to process drastically different amounts of data at different stages of an algorithm would make the SIMD approach very inefficient. Since all the available processors must act in a synchronised manner, processing a large number of data items at one stage of the algorithm implies that a large number of processors will be needed to process them. If the algorithm involves the processing of a small number of data items during other phases, many processors will be left idle.

The SIMD paradigm is at its most effective when the number of data items to be processed is a good match for the number of available parallel processors.

Processor Arrays

The **processor array** is a class of hardware which was specifically developed to implement the SIMD paradigm. It is thus a collection of synchronised processing elements capable of simultaneously performing a single operation on different data (see Fig 3.2). Each processor in the array has a small amount of local memory where the distributed data reside while being processed in parallel.

Some of the first pieces of parallel processing hardware to be implemented were processor arrays which arose from the design of the SOLOMON computer ([Slotnick et al., 1962]). Although the SOLOMON computer was probably never actually built, the elements of its design led to many successfully implemented processor arrays, including the ILLIAC IV ([Barnes et al., 1968]) and PEPE ([Berg et al., 1972]) floating-point arrays, as well as the Goodyear Aerospace STARAN ([Batcher, 1974]) and ICL DAP ([Reddaway, 1973, Hunt, 1981]) arrays of one-bit processors.

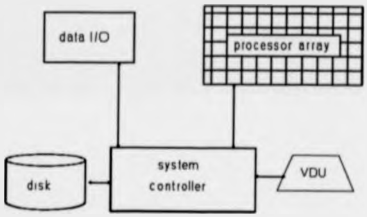


Fig. 3.2: Typical processor array architecture.

As implementations of the SIMD paradigm, processor arrays are most effective in practice when the data upon which they operate matches the dimensional structure of the processor array itself. The two major application areas where this matching occurs are image processing and mathematical modelling (usually of physical processes).

A relatively large number of processors (i.e. a fine grain system) are needed to accurately process images or model physical processes. Two of the most commercially successful fine grain processor array systems are the Goodyear Massively Parallel Processor (MPP) ([Batcher, 1980]) and the Connection Machine (CM) ([Hillis, 1985]). The Goodyear MPP is a descendant of the Goodyear STARAN and the ICL DAP and contains an array of 16384 one-bit processors, arranged as a 128x128 square. The Connection Machine has an array of 65536 processors, arranged as a 256x256 square.

Examples of medium grain processor array systems are the previously mentioned 4096 processor ICL DAP, the 4096 processor Mosaic ([Dornheim, 1985]) and the 512 processor GF-11 ([Beetem et al., 1985]). Most modern processor array systems are fine or medium grain systems, although some large grain systems do exist for use in less prevalent application areas such as economic modelling e.g. the 16 processor Columbia University Parallel Computer ([Christ and Terrano, 1986]).

3.2.3. The Systolic Paradigm

The systolic synchronous parallel paradigm incorporates features of both the vector/array and the SIMD paradigms.

A systolic parallel computer is a pipelined multiprocessor in which data are distributed and pulsed from memory to an array of processors before returning to memory. The name systolic comes from the analogy with the systolic, or pumping, action of the heart ([Kung, 1979]). A simple example of such a computer may be thought to consist of a two dimensional array of processors with memory at the boundary of the array (see Fig 3.3).

P = processor

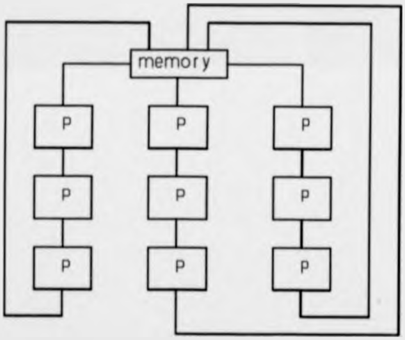


Fig. 3.3: Systolic architecture.

The systolic paradigm is efficient in theory because it avoids input/output bottlenecks by circulating data among the processors as much as possible before returning it to memory. The systolic system is at its best when operating on large amounts of data and performing fine-grain calculations since processing small batches of data inevitably leave processors idle and the maximum speed up is to be achieved by pipelining data through many detailed operations.

Systolic computers are presently only built for specific applications where the systolic paradigm can operate efficiently, although work is being done to develop programmable systolic-array elements to allow more flexibility. Application areas have included signal processing ([McCanny and McWhirter, 1982]) and mathematical transformation calculations ([Kung, 1984, Chakrabarti and JÁJÁ, 1990]). More general references are ([Moore et al., 1987]) and ([Quinton, 1987]).

3.2.4. The MIMD Paradigm

The first asynchronous coordination method to be considered is the MIMD paradigm, which allows many processors to simultaneously execute different instructions on different data. The MIMD paradigm coordinates tasks and processors by using some form of synchronisation mechanism. One obvious synchronisation mechanism is to only allow one processor at a time to access a piece of data at one moment in time. Any other processors must wait until the data item is released. Another mechanism is to only allow one processor at a time to change a piece of data, whilst letting other processors read it (i.e. to have, in effect, a shared memory system). It is the responsibility of the parallel algorithm designer to make use of such synchronisation mechanisms to ensure the integrity of the algorithm.

The MIMD approach is most efficient when dealing with fine detailed problems where individual processors have much work to do on the data they receive. This will keep the individual processors busy for relatively long periods of time, thus helping to reduce the overhead involved in passing around data and control statements from processor to processor.

Numerous different types of hardware have been developed that may be categorised as MIMD machines, i.e. which consist of a number of processing units, each capable of executing different operations on different data. These different types of hardware may be further classified by the methods they use to enable MIMD processing (which include making use of previously mentioned SIMD and pipelining hardware).

The different categories are: **Pipelined MIMD**, **Switched MIMD** and **Network MIMD**.

3.2.4.1. Pipelined MIMD

Pipelined MIMD machines process multiple instruction streams by time-sharing a single sophisticated pipelined instruction processing unit. An example of using pipelined MIMD within a machine was the Denelcor Heterogeneous Element Processor (HEP), which was discontinued in 1985 ([Smith, 1978, Allan and Oldehoeft, 1985]).

3.2.4.2. Switched MIMD

Another method of processing multiple instruction streams is to provide separate instruction processing hardware for each stream. One way of implementing this is to provide a switch via which all connections between the processors must be made.

Switched MIMD machines may be further classified by how they utilise memory.

Shared-memory MIMD machines are organised so that memory is a shared resource of all the processors that is accessed via the switch (see Fig 3.4). Examples of such machines are MIDAS ([Maples et al., 1981]), early versions of the New York University Ultracomputer ([Schwartz, 1980, Elder et al., 1985]), the University of Illinois Cedar ([Gajski et al., 1983]) and the IBM RP3 ([Pfister et al., 1985]).

P = processor

M = memory

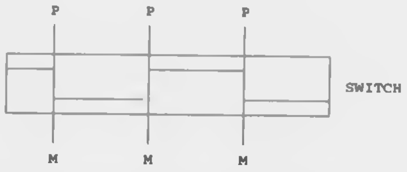


Fig. 3.4: Shared-memory MIMD architecture.

__**Multiprocessors**__

Some of the later categories of **multiprocessor** hardware (e.g. the Encore Multimax ([Wilson, 1987]) and the Sequent Symmetry ([Lovett and Thakkar, 1988])) were designed as shared-memory MIMD machines ([Desrochers, 1987]). Although the first large-scale multiprocessor design occurred in 1959 ([Holland, 1959]), up until 1980, such designs were mainly methods for connecting together several independent serial computers (e.g. the Carnegie-Mellon C.mmp computer linking together 16 DEC PDP-11 minicomputers ([Wulf and Bell, 1972, Wulf et al., 1981])).

The arrival of the cheap microprocessor (circa 1975) allowed the subsequent development of systems using linked microprocessors which cooperated to solve a single problem, i.e. true multiprocessors ([Pease, 1977, Bustos et al., 1979]).

Many early multiprocessor systems failed to live up to their full potential when more than a few microprocessors were joined together. The major problem incurred when scaling up such systems was caused by the slowness of the switch which gave access to the memory. The switch could not cope properly with the demands of more than a few processors and was thus the cause of delays and queuing. This problem gave a major push to the further development of distributed memory and network MIMD hardware.

Distributed-memory MIMD machines (often described as **multicomputers**) distribute memory amongst the processors as local memory and the processors communicate via the switch (see Fig 3.5). Examples of such machines are CHoPP ([Sullivan et al. , 1977]), and the BBN Butterfly ([Crowther et al., 1985]). A more general reference is ([Seitz, 1988]).

P = processor

M = memory

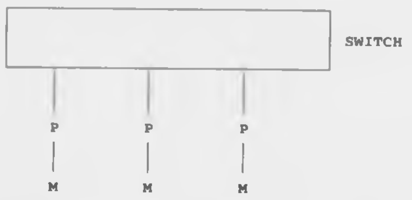


Fig. 3.5: Distributed-memory MIMD architecture.

Again, the speed of the switch in practice limits the amount of parallelism possible, since it limits the amount of communication possible between processors. The balance between communication and calculation is thus a very important consideration when making use of distributed-memory MIMD machines. Too much dependency on message-passing for instruction or to pass data can incur large communications overheads. On the other hand, having too many instructions embedded within a processor's algorithm limits the flexibility of the overall algorithm as well as limiting the amount of data that can be stored locally.

3.2.4.3. Network MIMD

An alternative way of providing separate instruction processing hardware for different instruction streams is to devise a network of processors. Individual processors may only communicate directly with their neighbours in the network, and long-range communication across the network requires a routing algorithm. The removal of the switch from this design removes the limitations on memory access or communications that accompanied the shared-memory and distributed-memory MIMD designs previously mentioned. Each network MIMD processor has access to some local memory and has a number of links for connection to neighbouring processors.

Examples of network MIMD machines (which again are often classified as multicomputers due to their lack of global shared memory) are the CDC Cyber Plus, the NASA Finite Element Machine ([Jordan, 1978]), the Carnegie-Mellon University Cm* ([Swan et al., 1977, Gehringer et al., 1987]) and many of the various implementations of hypercube architecture (e.g. the Cosmic Cube ([Seitz, 1985]), the Intel iPSC ([Pase and Larrabee, 1988]), the Floating Point Systems T series ([Miller et al., 1988]), the NCUBE/10 ([Hayes et al., 1986]), the generalised hypercube (Bhuyan and Agrawal, 1984]), the twisted cube ([Efe, 1989]) and the cube connected cycles architectures ([Preparata and Vuillemin, 1981])).

The Inmos Transputer

Especially useful for network MIMD systems is the transputer, a single-chip microprocessor developed by INMOS Ltd and first marketed in 1985 ([INMOS, 1985, Whitby-Strevens, 1985]). It is intended to be the equivalent of a TRANSistor for multicomPUTER architectures, i.e. to be the lowest level component that needs to be considered when designing a multiprocessor computer.

The T800 transputer (see Fig. 3.6), which appeared in 1987, consists of a powerful processor with 4Kbytes of on-chip RAM, four bidirectional serial lines intended for connections between transputers in an array, a 32-bit port which can be used to program the device or expand the local memory, and very importantly, a 64-bit floating point unit.

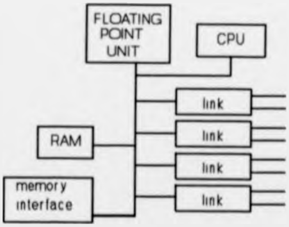


Fig. 3.6: The T800 transputer.

As with distributed-memory MIMD architectures, it is important to consider the balance between communication and calculation when building transputer-based networks, but at least the problem of delays caused by the switch has been removed from the equation.

Transputer-based systems were specifically designed to be used with the Occam parallel processing language, but they also support the use of programs written in more common programming languages (e.g. FORTRAN, PASCAL, C). Transputers are available singly or on boards (usually housing four, nine or sixteen at a time) and as such, make ideal building blocks for parallel systems. A major drawback to the use of transputers when they first became available however, was the lack of a debugging facility. Only one transputer per network could be connected directly to a host computer, and thus only one transputer could be used to output debugging messages. Such messages would first have to be passed to the host machine along a predetermined route of transputers. More recently however, a debugging facility has been introduced to ease the process of program creation.

Transputer-based systems have found many application areas, including optical character recognition ([Patry et al. 1987]), image processing ([Harp et al., 1987]) and robot control ([Pham et al., 1990])

3.2.5 The Reduction Paradigm

The reduction asynchronous coordination paradigm is so called because it is based on a mathematical graph reduction model. Most reduction problems involve flows of data, which are shown as graphs. For instance, consider the problem of finding the average of two numbers, a and b. The graph for this problem would be

```
a
     +    /    2
b
```

The computation algorithm simplifies the graph, stage by stage, until the graph is reduced to a single node. The graph is simplified (and the remaining computation that is necessary "reduced"), by performing the calculations for which there is hard data. For instance, if, for the problem stated above, a=3 and b=5, the first reduction is to reduce the generic problem to the particular instance, as follows:

```
3
     +    /    2
5
```

The next reduction is carried out by performing the one calculation for which there is hard data (i.e. 3+5), giving the resulting graph

The last reduction is carried out by again performing the calculation for which there is hard data (i.e. 8/2), thus giving the final graph (reduced to one node), which is the answer 4.

The reduction paradigm achieves parallelism by using a method called **demand-driven data flow**. This method states that a task may only begin execution when its results are required for use by another already executing task, i.e. when its results are demanded. A reduction program consists of reducible expressions which are replaced by their computed values as the computation progresses through time. Most of the time, the reductions may be done in parallel. Nothing prevents parallel reductions except the availability of data from previous reductions.

As an example of this, consider the quadratic equation,

$ax^2 + bx + c = 0$.

A reduction program to compute the largest possible value for x that satisfies the equation, given values for a, b and c would be as follows (this example adapted from (Lewis and El-Rewini, p16):

graph 1: graph of x=(-b + SQRT(b*b-4ac))/(2a)

```
c                   b

     *    *    -    *    SQRT      +    -    b

a         4         b              /

                                   *

                              2         a
```

graph 2: (for input a=1,b=2,c=-3) reduced by a*c,-b,b*b,2*a
(i.e. four possible parallel operations)

```
-3   *    -    4    SQRT      +    -2

     4                        /

                              2
```

graph 3: reduced by -3*4

```
-12  -    4    SQRT      +    -2

                         /

                         2
```

graph 4: reduced by -(-12), then 4+12

```
16   SQRT      +    -2

               /

               2
```

graph 5: reduced by SQRT(16)

```
4    +    -2

     /

     2
```

graph 6: reduced by -2 + 4

2

/

2

graph 7: reduced by 2/2

1

Note that there are nine operations to be carried out in the formula

x = -b + SQRT(b*b - 4ac))/(2a)

but that the reduction algorithm only took seven reductions to compute the final graph.

Only a few applications have arisen from this classification of parallel algorithm design so far, due to the very high overheads involved in processing graphs. These applications have been centred around the implementation of functional languages ([Darlington and Reeve, 1981, Peyton-Jones, 1987]).

Of all the major classes of parallel design paradigm mentioned above, only the vector/array, SIMD and MIMD classes have yielded hardware that is of use in general. This will remain the case until technological advances are made or new application areas discovered.

3.3. Parallel Problem Decomposition Algorithms

There are three major approaches to the decomposition of a problem so that it may be attacked using a parallel algorithm implemented on one of the different pieces of hardware mentioned above. These are the **algorithmic**, **geometric** and **process farming** approaches to parallel computation. Hybrid methods also occur in practice, which combine two or three of the approaches.

3.3.1. The Algorithmic Approach

The **algorithmic** approach, also known as **data flow decomposition**, involves breaking up the problem solution algorithm into separate, independent subtasks. Each subtask can then be executed in parallel, with data flowing between the subtasks if necessary. Each subtask will perform some computation with the data it receives and then pass data onto any further necessary subtasks.

In cases where algorithmic subtasks must be carried out in a particular order, the algorithmic approach is often used with vector/array (i.e. pipelined) computers, with each processor in the pipeline contributing by executing a section of the overall algorithm.

As an example of the pipeline implementation of the algorithmic approach, consider the following, adapted from (de Carlini and Villano):

The Sieve of Eratosthenes is a method for finding the prime numbers below a given integer N. The algorithm removes from the set of odd numbers less than N, all the multiples p_k^2, $p_k(p_k+2)$, $p_k(p_k+4)$, etc. of the kth prime p_k.

An algorithmic decomposition of the sieve algorithm has been implemented (see [Hoare, 1978]). One processor is used as a "source" that generates and sends out a stream of odd numbers less than N. The remaining processors act as a series of sieves. A stream of odd numbers is input into each of the sieve processors. The first odd number is saved (as it is a prime), and any multiples of it are removed from the stream. The remaining numbers are then passed on to the next sieve.

As an example, in Fig. 3.7, where N=19, the source processor outputs the odd numbers 3,5,7,...,17 to the first sieve. The first sieve takes 3 to be a prime and thus filters out any multiples of 3 (i.e. 9 and 15). The remaining numbers are sent to the next sieve and the process continues until the output stream from the last sieve are primes, and each of the sieve processors holds one prime number.

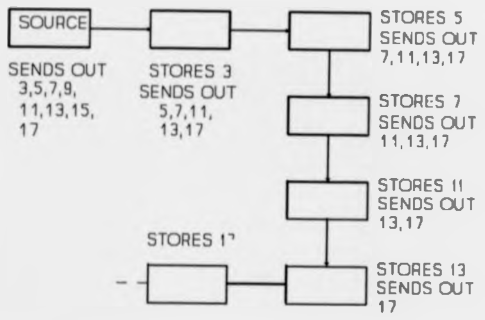

SOURCE
SENDS OUT 3,5,7,9, 11,13,15, 17
STORES 3 SENDS OUT 5,7,11, 13,17
STORES 5 SENDS OUT 7,11,13,17
STORES 7 SENDS OUT 11,13,17
STORES 11 SENDS OUT 13,17
STORES 13 SENDS OUT 17
STORES 17

Fig. 3.7: Sieve of Eratosthenes.

Another example of the pipeline implementation of the algorithmic approach is in Fast Fourier Transformation computation ([Villano, 1990]).

The pipeline is of course not the only way to implement the algorithmic approach to problem decomposition. If a pipeline implementation is chosen however, the benefits of parallelism will only be realised if the pipeline is kept full. It is thus important when decomposing the overall algorithm that the subtasks are chosen prudently. Obviously, the throughput of data through the pipeline is limited by the speed of the slowest subtask. If one subtask in the pipe takes considerably longer to perform than the others, a bottleneck will soon occur and the benefits of parallelising the algorithm will be lost.

3.3.2. The Geometric Approach

The geometric approach to problem decomposition involves designing processes that match the spatial geometry or structure of the problem (and is thus also known as data structure decomposition). The spatial geometry of some problems may be divided so that the separate divisions of the problem space interact and communicate with each other.

Processors may be allocated to each of these areas. Each processor is then considered to be a semi-independent entity, responsible for the data in its own spatial region. In cases where geometric decomposition is used, the computations performed by each processor are usually identical, with the results usually being summed to give some overall effect. The subdivision of the problem is usually carried out so that short range interactions between neighbouring units take place to give a more realistic end solution.

An example of the geometric approach (from (Galletly, page 198-210)) is its use in the simulation of thermal conduction in a two-dimensional rectangular metal plate which is being heated by a heat source at a certain point (see Fig 3.8). Simulation of thermal conduction over the whole plate is difficult, so to simplify the problem, the geometry of the situation is utilised and the plate is subdivided into a number of rectangular areas.

The heat conduction (i.e. temperature) of each of these areas may be estimated by a processor and summed to give an approximate effect for the heat conduction over the whole plate. The temperature of each area will depend on that of its surroundings i.e. the neighbouring areas. It is assumed that one of the areas contains the heat source.

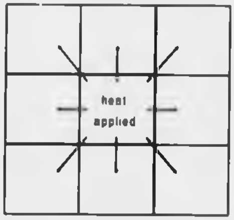


Fig. 3.8: Geometric decomposition applied to the simulation of thermal conduction.

Not all geometrically decomposable problems can be properly attacked on the hardware currently available. Some systems do not allow the reconfiguration of their processor network, whilst others (e.g. the transputer) only have a certain number of links available with which to create connection topologies.

Once a topology has been satisfactorily set up, it is important that the amount of communication between neighbouring processors is carefully considered. The communications overhead between neighbouring processes can become quite appreciable and thus slow down the overall algorithm if communication is not properly balanced against calculation.

<u>3.3.3. The Process Farming Approach</u>

Finally, there is the process farming approach to problem decomposition, which involves making one processor a "master" and the rest anonymous "slaves" (see Fig 3.9). Each of the slave processors is used to perform exactly the same task, albeit on different data. The master processor is in charge of sending out data to the different slaves (in whatever order it sees fit to do so) and of making use of the results as they are returned from the slaves.

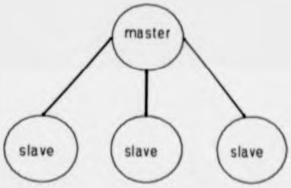


Fig. 3.9: Process farming approach to problem decomposition.

The process farm approach is applicable to many problems whose solution involves many independant but identical sub-calculations being carried out on different sets of data. As the sub-calculations are independent of each other, each may be executed in parallel and the effect summed to give a solution to the whole problem.

A famous example of the process farming approach to problem decomposition is the graphical representation of the Mandelbrot Set of complex numbers. The screen is used to represent a particular portion of the complex plane, with the constituent pixels of the screen representing the individual complex numbers in this portion.

A simple iterative calculation can be carried out to determine whether or not each particular complex number is a member of the Mandelbrot Set. Each slave processor is given a copy of the iterative algorithm and set to wait for input. On receiving input data about a particular complex number, the processor determines whether the number is a member of the Mandelbrot Set or not, and also notes how many iterations were necessary to determine this. These two pieces of information are then sent back to the master processor and the slave waits for more input.

The master processor sends out the information on particular complex numbers to slaves and coordinates the returning information. If the complex number is not a member of the Mandelbrot Set, the pixel representing it is given the colour black. If the number is a member, the pixel representing it is given a colour depending on how many iterations it took to determine. The end result is a colour pattern on the screen representing the Mandelbrot

Set members for the particular portion of the complex plane chosen.

Process farms may be set up with many different processor connection topologies, limited only by the capabilities of the available hardware, but as with geometric decomposition, a careful balance must be maintained between calculation and communication. The process farm approach is usually most successful when the slave processors are kept busy for a relatively long period of time once they have received some input data, thus minimising the proportion of communication.

3.3.4. Hybrid Methods

It is not uncommon for parallel algorithms to be developed which make use of more than one of the three approaches to problem decomposition. One example of this is a combination of farming and algorithmic decomposition used to produce a system for printed character optical recognition ([d'Acierno, 1990]).

In the vast majority of cases however, the needs of specific applications seem to point to the use of one of the methods more than the others. Although in such cases hybrid methods may sometimes be used to increase the efficiency of part of the overall algorithm, the benefit achieved is often not worth the effort involved.

3.4. Review of Literature on Parallel Algorithms used to Solve IP Problems

Background

As with most innovations in computer hardware, the development of parallel processing facilities was motivated by a wish to solve a range of previously intractable (i.e. NP-complete or NP-hard) problems in a reasonable (i.e. polynomial) time as well as by the need to solve already tractable problems more quickly.

Unfortunately, the architecture of early parallel hardware (i.e. vector/array machines such as the CRAY-1 and CDC Cyber 205), and the high cost of construction, strictly limited the number and range of early implementations of parallel algorithms. Schnabel states that much of the early work done using parallel hardware was thus concerned with the solution of partial differential equations and associated areas such as numerical linear algebra ([Schnabel, 1984]). Researchers in other application areas had to resort to constructing their own parallel architectures or to the use of simulation techniques for modelling parallelism on a single-processor machine.

Researchers within the field of Mathematical Programming began to exploit the potential benefits of parallelism by producing parallel algorithms to attack many different categories of combinatorial problem (see [Kindervater and Lenstra, 1986] for a survey). These parallel algorithms were based on the generalised Branch and Bound algorithm, special cases of which have been used to produce several popular search strategies for combinatorial problems ([Nau, Kumar and Kanal, 1984]).

Existing Parallel Algorithms

Roucairol reports that different types of work have been carried out in the field of combinatorial optimisation using parallel hardware ([Roucairol, 1989]):
"firstly, those proposing parallel Branch and Bound without any effective implementation or using a simulated parallelism on a sequential machine; secondly, the implementation on experimental machines (i.e. machines built in research laboratories with exotic architectures); and lastly, later experiments conducted on commercial supercomputers".

The first category includes simulations of various parallel Branch and Bound algorithms for many different combinatorial problems ([Imai, Fukumura and Yoshida, 1979, Li and Wah, 1984,1986, Wah and Ma, 1984, Lai and Sahni, 1984, Mohan, 1983, deBruin, Rinnooy Kan and Trienekens, 1988, Boehning, Butler and Gillett, 1988]).

The second category includes algorithms implemented on specially designed machines such as the Manchester Dataflow Machine ([Kindervater and Trienekens, 1985]), the Carnegie-Mellon Cm* ([Mohan, 1982]), and the Boulder DPU ([Trienekens, 1986]).

The last category includes work done on the ICL DAP and CDC Cyber 205 ([Kindervater and Trienekens, 1985]), the CRAY-2 ([Roucairol, 1986]), the CRAY-XMP ([Lavalée and Roucairol, 1985, Pardalos and Rodgers, 1990]), the Denelcor HEP, Sequent Balance and Encore Multimax ([Boehning, Butler and Gillett, 1988]), the Intel iPSC hypercube ([Mraz and Seward, 1987]) and the Inmos transputer ([Vornberger, 1988, McKeown et al, 1990, Gendron and Crainic, 1992]).

The important features of the parallel Branch and Bound algorithms mentioned above are:

the coordination and utilisation of the parallel processors;
the hardware used for implementation; and
the search strategy used.

The coordination of the parallel processors obviously has a direct effect on the utilisation of the processors (as well as on the complexity of any necessary communications). This can be shown by comparing algorithms that are synchronously and asynchronously coordinated (see [Mohan, 1982,1983, Trienekens, 1986]). It can be seen that an asynchronously coordinated algorithm allows much better utilisation of the parallel processors. This is because it is quite possible that different nodes will take different amounts of time to attack. Thus, a synchronously coordinated algorithm would often have several processors waiting idly for the hardest-working processor to finish. It is thus not surprising that all the algorithms implemented use an asynchronous coordination of processors (except for one of several algorithms discussed in [McKeown et al, 1990]). The different algorithms do still of course exhibit different processor utilizations because of the influences of the hardware and search strategies used.

The search strategies used were of three general types: depth-first search; breadth-first search; and best-first search.

Depth-first search, where a descendant of the previous node is always explored next (if possible) was very popular in the earlier algorithms (e.g. [Imai, Fukumura and Yoshida, 1979, El-Dessouki and Huen, 1980, DeWitt, Finkel and Solomon, 1984, Finkel and Manber, 1985]) because the necessary data structures were small, as was the memory on the available hardware. Each of these algorithms showed that a good speedup of the solution times could be achieved when several processors were used. Pruul showed that if a depth-first approach is used, attacking nodes in parallel yields better solutions earlier, thus allowing better pruning of the search tree and a good reduction in the number of nodes examined ([Pruul, Nemhauser and Rushmeier, 1988]). This is reflected by the use of depth-first searches in more recent algorithms ([McKeown et al, 1990, Gendron and Crainic, 1992]).

Breadth-first search, where all nodes at a level of the search tree must be attacked before any of their descendants, was also used by some early algorithm designers. Li and Wah showed that good speedups of solution times were possible if a breadth-first search was used ([Li and Wah, 1984,1986]). This is probably only because of the raw processing power available when two or more processors are used however, and not due to any cleverness of the algorithm.

A best-first search, where (for minimisation problems) the node with the smallest lower bound is chosen, was used by several of the algorithms (e.g. [Wah and Ma, 1982,1984, Wah, Li and Yu, 1985, Quinn, 1986, Felton, 1988]). Lai and Sahni claimed that a near linear speedup of the solution times could only be achieved by a parallel Branch and Bound algorithm using a best-first search strategy for a small number of processors (i.e. less than sixteen). This was challenged by Li and Wah however, who showed theoretically that a near linear speedup was possible for a large number of processors (i.e. one to two thousand)([Li and Wah, 1984]). Quinn and Deo state an upper bound on the speedup available if the best-first search is used as just below linear until the Amdahl Effect occurs (i.e. until there is not enough work for the available processors)([Quinn and Deo, 1986]). They also state however, that superlinear speedup could be achieved if a different search is carried out by another run of the algorithm. This is highly likely for most of the algorithms since they are asynchronously coordinated and thus non-deterministic in nature.

Finally, let us consider the issues relating to the hardware used to implement the parallel Branch and Bound algorithms. The hardware used can be categorised as either having or not having global shared memory.

Only a small proportion of the algorithms used global shared memory (e.g. [Roucairol, 1987, Imai, Fukumura and Yoshida, 1979, Boehning, Butler and Gillett, 1988]). The remaining algorithms were all implemented on machines that had no shared memory facilities (i.e. distributed memory machines).

Abdelrahman and Mudge characterised the parallel Branch and Bound algorithms implemented on distributed memory systems as either Central List (CL) or Distributed List (DL) algorithms ([Abdelrahman and Mudge, 1988]). Central list algorithms maintain a list of active nodes on one processor and perform the calculations on chosen nodes on the other processors (i.e. the farming approach to parallel Branch and Bound). Such algorithms have the advantage that when a node is chosen, the decision is made with full knowledge of the search so far. The major disadvantages of such algorithms are that a lot of memory is needed to hold the centralised list (some of which may have to be written to disk) and that there is the potential for a message-passing bottleneck at the processor containing the list since all the other processors will interrogate it for work.

Distributed list algorithms place a separate pool of active nodes and an incumbent solution on each processor. Such algorithms thus do not suffer from the bottlenecking problems common to central list algorithms, but have the disadvantage that node selection decisions made by processors are only based on local knowledge of the search and may thus lead to unproductive work. Li and Wah state that the performances of distributed list algorithms are usually worse than those of central list algorithms ([Li and Wah, 1984]). Abdelrahman and Mudge reported results for both types of algorithm as implemented on a NCUBE/six multiprocessor ([Abdelrahman and Mudge, 1988]). The central list algorithm exhibited a good speedup of solution times for the problems attacked, but only for a small number of processors. The distributed list algorithm did not at first produce good results at all (seemingly reflecting the conclusions of Li and Wah), but this was changed by the introduction of a load-balancing scheme.

Load-balancing is a very important issue for all parallel algorithms, since idle or underutilised processors will not help an algorithm to achieve the benefits of parallelism. Ma et al developed a load-balancing mechanism for the hypercube ([Ma et al, 1988]). This mechanism involved idle processors interrogating their neighbours for work and was similar to the load-balancing scheme used by Abdelrahman and Mudge. Felton suggested another load-balancing scheme for use with distributed list algorithms on the NCUBE ([Felton, 1988]). In his scheme, new nodes generated by a processor are added to the local pool of a different processor at random, in order to achieve a good spread of useful nodes to attack over the whole system. Vornberger suggested a method whereby problems are 'sent without request' to other neighbouring processors on the off chance that they might provide useful information ([Vornberger, 1988]). He achieved a superlinear speedup of some problem solution times using this method.

Conclusions

Based on a study of the already implemented parallel algorithms used to solve combinatorial problems, an asynchronously-controlled Branch and Bound algorithm making use of either a depth-first or best-first search would seem appropriate for initial testing of a parallel algorithm on MIP problems. If a small number of processors is to be used then a central list algorithm would be appropriate. If larger numbers of processors are to be used, a distributed list algorithm using some form of load-balancing system would be appropriate.

4. Initial Experiments with Parallelisation

4.1. Choice of the Parallel Algorithm and Hardware

Each of the MIP-solving algorithms introduced in Chapter Two will now be considered for parallelisation. The major criteria for a good choice of algorithm to be parallelised are that:

the algorithm chosen must exhibit a useful amount of exploitable parallelism (i.e. it must be decomposable by the algorithmic, geometric or process farming approaches to parallelism) ;

the parallelised algorithm must be implementable on currently available hardware; and

the implemented algorithm must be flexible enough to deal with different types of MIP problem.

Cutting Plane Algorithms

The algorithmic and geometric decomposition approaches cannot be usefully applied to the FCPA algorithm, but a process farm provides a natural way of exploiting the parallelism contained in the algorithm. Each slave processor can be used independently to generate a cut and then send it back to the master processor for application to the LP relaxation.

The main problem with using the farming approach is that it is only useful to solve the new LP when the cuts have been returned and applied to the problem. The use of a farming algorithm will thus result in some (or all) of the slave processors being idle at certain points in the algorithm.

If we assume that the number of new cuts that the algorithm indicates can be generated is less than the number of slave processors, then obviously some of the slaves will not be sent work and will remain idle. Even if there are more potentially useful cuts than there are slave processors to generate them, both the master and slave processors will always be idle some of the time.

The actual amount of time that the master and slaves are idle depends on whether the algorithm is designed to send data back and forth in a synchronous or asynchronous manner.

If a synchronous control mechanism is used, the master will remain idle until all the busy slaves indicate that they have finished generating cuts, at which point all the new cuts are returned and applied, and the new LP is solved. Obviously, while the new LP is being solved, all the slave processors are idle.

If an asynchronous control mechanism is used, a prioritisation of the potential cuts to be generated would be useful. The cut data most likely to be useful could be sent out to the slaves first, with any lower priority cuts being generated only if time permits and there are enough free slaves. Once enough of the high priority cuts have been generated and returned, the LP can be resolved while the less important cuts are still being generated. Thus, not all of the slave processors will be idle whilst the master processor is solving the new LP. It is not obvious, however, how many of the high priority cuts should be applied before the new LP is solved

Both the synchronous and asynchronous approaches share the minor problem of ensuring that the cuts generated in parallel are distinct, so that the potential benefits of parallelism are not wasted.

The major feature of this algorithm to note as far as its implementation is concerned is that the LP problem grows with every distinct cut added. Thus, a greater amount of memory is needed on the master processor to contain the details of each successive LP to be solved. Since the FCPA algorithm cannot be proven to be convergent unless the MIP problem's objective function always takes integer values, there is no guide to how many iterations the problem will take to solve, and thus how big the final LP problem for solution will be. Since the FCPA algorithm does not yield any feasible solution until the algorithm terminates, it is thus possible that difficult MIP problems could not be solved unless the master processor has access to large amounts of memory.

If such hardware is available however, the parallel FCPA algorithm would be flexible enough to attack many different types of MIP problem, using the different types of cuts mentioned in section 2.1.2 for use with different problems.

Partitioning Algorithms

Benders' Decomposition algorithm involves repeating the process of solving a dual LP problem and a reformulation of the initial MIP problem. This process provides better and better lower and upper bounds on the optimal solution of the initial MIP problem until, when the lower bound on the optimal solution is the same as the upper bound, the optimal solution has been found.

There is no parallelism to be exploited from this algorithm since the dual LP to be solved at each iteration is dependent on the Benders' reformulation solved at the last iteration and the Benders' reformulation solved at each iteration is dependent on the solution of the dual LP at the last iteration. Hence, there is a strict order in which the calculations must be performed.

The Lagrangian Relaxation partitioning method, however, does exhibit some exploitable parallelism. The initial MIP problem is partitioned to create a new problem which only contains constraints that are easy to satisfy. The objective function of the new problem includes a construct made up of the complicated constraints of the original MIP multiplied by a vector λ. If the solution vector of the new (easy to solve) problem is close enough to a given set of values, then the original MIP has been solved. If not, another vector λ must be chosen and another instance of the partitioned problem solved. The best way to parallelise the algorithm would be to implement a process farm. Different sets of multipliers (λs) could be chosen by the master processor and sent to slaves which would solve an instance of the partitioned problem and compare the solution vector with the appropriate values. A process farm implementation of this algorithm would keep the slave processors busy as long as the process of farming out the λ vectors can be performed quickly enough to avoid a bottleneck occurring at the master processor. The busy slaves will only be performing useful work, however, if distinct and effective sets of λ values can be generated by the master processor. To make an effective choice of λs, the master processor algorithm must note the results of previously solved instances of the partitioned problem.

Since the partitioned problem is designed to be easy to solve, the parallel algorithm should be implementable on any type of parallel hardware currently available, as long as enough memory is available for the master processor to keep track of the calculations performed.

A restriction to the usefulness of the parallel algorithm is imposed by its limited flexibility. There are many different categories of MIP problems in existence and real MIP problems can often be formulated in several different ways. It is therefore difficult to write a set of all-encompassing rules to indicate which constraints of a problem are simple and which are complicated. It has also been shown that there are problems for which no λs exist to supply an optimal solution ([Everett, 1963]). A reasonably close solution would thus have to be acceptable in certain instances.

Group Theoretic Algorithms

As mentioned in section 2.3, MIP problems that have been reformulated as PIP Group Minimisation Problems can be solved by using a dynamic programming algorithm, an enumeration algorithm, or a shortest route algorithm. The basic structure of all three algorithms is the same, in that slightly different GMPs are solved until a solution is found where the basic variables all take positive integer values.

The dynamic programming approach is to define a set of recursion relations to find the rth best solution to the GMP and then to perform recursions until a suitable solution is found. Since each step of the recursion process depends on the results of the previous step, there is no exploitable parallelism in this approach.

The enumeration approach is to add a lower bound to one of the variables and resolve the GMP. The value of the lower bound is increased unit by unit until it reaches its highest allowable level (and the GMP is resolved with each unit increase). If performed on each of the variables in turn, this process will obviously eventually explicitly enumerate all the possible solutions to the GMP (which is a PIP). The optimal MIP solution will be given by the member of the set of GMP solutions where all the basic variables take positive values which has the minimum objective function value.

The shortest route approach to solving the GMP is very similar to the enumeration approach previously mentioned in that the different GMPs are constructed by adding lower bounds to the variables and the same enumerative search is carried out. The difference lies in the method of actually solving the different GMPs. The shortest route approach makes use of a specialised shortest route algorithm to solve the GMPs.

A process farm would be the best way to implement either of the enumeration or shortest path algorithms. GMPs with different lower bounds imposed on variables could be solved independently by slaves, with the master checking results to see if any solutions to the MIP had been found. The slave processors would be kept busy as long as the process of farming out work can be performed quickly enough to avoid a bottleneck occurring at the master processor.

Either of the algorithms would be implementable on currently available parallel hardware, as long as enough memory is available for the master processor to implement a book-keeping scheme to record the search.

The Group Theoretic algorithms can be used to attack many different types of MIP once the necessary transformation to GMP problems have been carried out.

Enumeration Algorithms

Both the Driebeek Direct Search algorithm for MIP problems and the Lemke and Spielberg algorithm on which it is based could be parallelised by using a process farm, as the search processes are quite similar to the methods used by the Group Theoretic algorithms mentioned above.

The master processor could choose where to take forward and backwards steps (i.e. set the value of a variable) and farm out the appropriate subproblems to be solved by slaves.

Both the direct search algorithms, however, cause problems when implemented for use on problems containing integer variables which can take a wide range of values.

In the case of the Driebeek algorithm, which represents integer variables by a set of binary variables, it is difficult to assess the amount of memory that should be set aside for the use of the processors. A large amount of memory will need to be set aside on both the master and slave processors to allow the possible representation of integers which can take a large range of values.

It should also be noted that the process of transforming a problem in integer variables into a problem in only binary variables is not unlike transforming the MIP into a combinatorial problem, in that the transformed problem will contain a much larger proportion of discrete variables if the bounds on the original integer variables were not close together. The subproblems to be solved during the search will thus have the characteristics of combinatorial problems.

In the case of both the Direct Search algorithms, which are based on an algorithm for solving PIP problems, the extensions to the PIP algorithm that enable general MIP problems to be solved subtract from the power of the algorithm in that there is so much more enumeration to be done if variables are allowed to take several different values. The implicit enumeration criteria used as part of the MIP algorithm are also inefficient, some of the criteria being weakened by the presence of continuous variables. The result is a poor relation to the Branch and Bound algorithm, discussed below.

The Branch and Bound enumeration algorithm is a much better candidate for parallelisation (as is reflected by the number of parallel Branch and Bound algorithms appearing in the literature). The algorithm basically involves solving many different LPs so as to either explicitly, or hopefully implicitly, enumerate all the possible solutions. Since none of these LPs are dependent on each other, they can be solved independently, again by implementing a process farm. A master processor may make choices of active candidate LPs and the slaves merely solve LPs when the appropriate data is received. The Branch and Bound search is also much better directed than that of the Direct Search algorithm, since the Branch and Bound search criteria work equally as well on MIP problems as on PIP problems.

The enumerative processes carried out by the Group Theoretic algorithms and by Direct Search Enumeration are very similar to those of the Branch and Bound algorithm, although the search carried out by the Branch and Bound algorithm is not so blind. The Group Theoretic algorithms and the Direct Search Enumeration algorithm will therefore not be considered for parallelisation.

This leaves the Fractional Cutting Plane, Lagrangian Relaxation and Branch and Bound algorithms as the only real contenders for parallelisation. The Branch and Bound has the following advantages over the other two algorithms:

the LP problems to be solved as part of the Branch and Bound algorithm are of a constant size so that the required memory of each processor can be properly estimated;

the Branch and Bound algorithm can be proved to be convergent for MIP problems;

intermediate feasible solutions may be found while the algorithm works, thus a "nearly-optimal" solution may be found if required; and, probably most importantly

the Branch and Bound algorithm is the sole base for nearly all the commercially successful IP-solving codes on the market. There are thus many ad hoc methods involving tolerances and parameters that have been developed for use with the Branch and Bound algorithm which can be used to aid the solution of real problems.

When considering the choice of algorithm it must also be remembered that MIPs have previously been defined for our purposes as having a large LP component and relatively few integer entities (e.g. binary variables, general integers or special ordered sets). For this class of problem, serial implementations of the Branch and Bound algorithm expend much more effort in solving the LP relaxations than in choosing the LP relaxations and performing the input/output work of passing problem data and solutions back and forth. Solving the LP relaxation at each node often involves many hundreds of (dual) Simplex iterations. This indicates that a relatively small processor farm arrangement should be able to keep many of its slave processors occupied for much of the time if running a parallel Branch and Bound algorithm, thus achieving much of the possible benefits of parallelism. So, it was decided to parallelise the Branch and Bound algorithm using a process farm arrangement.

Having decided to use the farming approach to parallelism, the Inmos T800 transputer provided a natural platform for experiments. When the research was begun, the T800 had one of the fastest floating point units of the microprocessors on the market, enabling quick LP solutions by the slave processors of the farm.

A well developed FORTRAN compiler was also available for use with a number of transputers installed as part of a PC environment. As well as dealing with standard FORTRAN code, this compiler had many extensions for the message-passing and synchronisation routines necessary to implement a farming application. Additional software was available to enable the electronic reconfiguration of the transputers into many different physical topologies.

There were also, however, disadvantages to using a transputer board within the PC environment. Although the PC environment is relatively cheap, it does limit the amount of communication possible between transputers and the host since only one transputer (the root) can communicate directly with the PC. If another transputer wishes to communicate, either to the screen or to a disk, it has to do this via the root transputer. A transputer only has four links (high speed communication channels) which are not easily reconfigurable when a program is running, thus placing a limit on the number of connection topologies that are possible.

The raw performance of the T800 transputer on floating point work was quite impressive when the research began. To illustrate this we give in Table 4.1 below the results of solving some LP problems on a 16MHz transputer with a floating point processor and 1Mbyte of private memory (from [Ashford, Connard and Daniel, 1992]). These results are compared with times obtained on a 20 MHz IBM PS/2 Model 70 with 20 MHz 80387 co-processor, which at the time the tests were conducted was quite a powerful machine, although it is slow by more recent standards.

Problems SC205, GFRD-PNC, BORE3D, ISRAEL, ETAMACRO and BRANDY are from the NETLIB test set ([Gay, 1985]), WILLETT is from Golden et al. ([Golden et al., 1988]).

Problem	NROWS	NCOLS	NONZ	NDP	TPTR	XPR1.51
BORE3D	234	549	1759	21.20	8.41	13.57
BRANDY	221	470	2371	53.88	24.60	33.95
GFRD-PNC	617	1092	3467	107.93	62.43	95.57
ETAMACRO	401	1089	2890	52.56	27.35	41.97
ISRAEL	175	317	2533	35.04	19.23	16.53
SC205	206	409	758	16.70	9.29	11.26
WILLETT	185	679	2532	130.00	71.35	96.39

Table 4.1: Solving example LP problems.

NROWS is the number of constraints, NCOLS the number of structural columns, and NONZ the number of non-zero elements in the matrix. Times are in elapsed seconds. Identical FORTRAN code was compiled for the NDP and TPTR columns. NDP refers to the Microway NDP FORTRAN compiler V1.4VM ([Microway, 1988]). 3L Ltd's Parallel FORTRAN ([3L Ltd., 1988]) was used for TPTR. XPR1.51 refer to version 1.51 of Dash Associates' XPRESS-MP ([Dash Associates, 1989]) optimiser which uses FORTRAN and assembler.

4.2. Parallel Algorithm Design Considerations

The major objective of the research was to implement a fast, sophisticated parallel MIP code within the PC environment. Dash Associates' XPRESS-MP optimiser [Dash Associates, 1989]) was chosen as the basis for the parallel code since the source was available. A board containing nine 16MHz T800 transputers (each of which had 1Mbyte of personal memory) was chosen as a platform for experiments.

The serial (i.e. non-parallelised) version of XPRESS-MP uses a Branch and Bound strategy to solve IP problems, as do the vast majority of commercial codes. XPRESS-MP also makes use of a few ad hoc methods necessary for the effective solution of real IP problems. All commercial MIP-solving codes make use of a system of switches, tolerances and strategies that have been developed over time and found to work on a wide variety of real problems. Without such a system, many solvable problems remain intractable for the serial Branch and Bound algorithm. It was decided that the parallel algorithm to be developed would also make use of such switches and tolerances in order to continue to attack real, hard problems.

The first major design feature of the serial code is that it makes good use of the available memory to hold the sizable data structures necessary to hold large IP problems.

The data can be classified as 'short node' data that is useful in making decisions for node selection as part of the Branch and Bound process and 'long node' data that is necessary to carry out calculations once a decision has been made.

For each node in the search tree, XPRESS-MP holds the 'short node' information in memory. This consists of the optimum objective function value of the LP relaxation at the node, an estimate of the best integer solution that can be obtained from branching at that node, and pointers to enable the tree structure to be traversed. The 'long node' information consisting of the basis and the current lower and upper bounds for each integer entity is held on disk and only retrieved when necessary for calculations.

Since the process farming algorithm was to be implemented on transputers which only had 1MByte of private memory, this long and short node data structure was retained in order to ensure that the (often large) data structures necessary to make decisions as part of the Branch and Bound enumeration of real MIP problems would fit onto the master processor.

The next design feature of the serial code is the amount of effort the algorithm spends at each node to get accurate estimates of the effects of branching on each non-satisfied variable. The estimates are made when the LP relaxation at the node has just been solved and optimal values (both primal and dual) are available.

It has been shown that it is usually beneficial when solving large MIP problems to devote significant effort at each node to obtaining good estimates ([Beale, 1977]) and so the estimation process was preserved for the parallel code.

4.1. Description of the Initial Algorithm Developed

The initial algorithm developed was loosely based on an algorithm previously implemented by Daniel on a board containing only four 16MHz T800 transputers ([Ashford, Connard and Daniel, 1992]). The Daniel algorithm is a good, if limited, example of a simple process farm, making use of three anonymous slaves. Unfortunately, the Daniel algorithm cannot be extended to use more than three slave processors, as it is only designed to send data to the slaves that are directly connected to the master. The Daniel algorithm was thus only used as a starting point for our initial algorithm.

Like the Daniel algorithm, the slaves of our redesigned algorithm use the same solution, branching variable selection and estimation code as the serial version of XPRESS-MP. In order to allow data to be passed to and from the master from many different slaves, a formal message passing technique had to be used rather than relying on the FORTRAN COMMON variables of the serial XPRESS-MP code or the direct connections of the Daniel algorithm.

Our initial algorithm featured node selection routines that were based on a parallel version of depth-first search, as indicated by the conclusions of section 3.4. Both immediate descendants of a node were tackled simultaneously if two slaves were free, otherwise the more attractive was started and no special attempt made to use a slave becoming free on the less favourable branch. If both descendants of a node were fathomed, a search of all active nodes was carried out, using best-first criteria for comparison if no integer feasible solution had yet been found. Once an integer feasible solution had been found, any comparisons made were based on the criteria suggested by Forrest, Hirst and Tomlin ([Forrest et al., 1974]), also described as 'quick improvement' ([Nemhauser and Wolsey, 1988]).

The FORTRAN-coded implementation of the initial algorithm was designed to be more efficient than the Daniel algorithm, and to allow a proper analysis of the results. The size of the messages passed to slaves was reduced in order to reduce the message-passing overheads incurred. To allow a better analysis of the results, routines were added to count the number of different types of nodes considered (i.e. number of solution nodes found, number of infeasible nodes, number of cut off nodes etc.). This included a count of nodes that were cut off when returned because the cutoff value had been tightened since their LP relaxations were sent out. The maximum and average number of transputers used was also noted for each test run.

For initial tests, the topology featuring three anonymous slaves was retained. The **root** transputer, which was connected directly to the host PC, was assigned to be the master and was thus also directly connected to three neighbouring transputers (the slaves). Each slave was connected only to the master and could not communicate with the other slaves. Thus the topology adopted, as seen in Fig. 4.1 was a simple star, with the root at the centre.

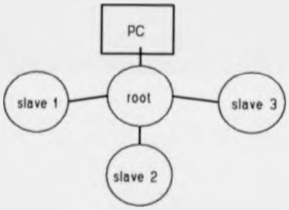


Fig. 4.1: Topology used for initial tests.

The algorithm placed on the slaves was as follows:

Step 1: Wait for a message from the master. If a message is received, go to Step 2.

Step 2: Receive the initial data structure for the LP relaxation of the original MIP problem. Go to Step 3.

Step 3: Wait for a message from the master. If a message is received, go to Step 4.

Step 4: Receive and apply bounds and a starting basis for solving an LP relaxation. Receive the value of the best integer solution found so far, to act as a cut-off. Also receive a tolerance which indicates how much better than the cut-off a new solution must be to be useful. Go to Step 5.

Step 5: Perform iterations of the Simplex algorithm to solve the LP. If the LP relaxation is infeasible, unbounded, or worse than the cut-off, go to Step 7. Otherwise, if the problem is solved, go to Step 6.

Step 6: If the solution is integer feasible, go to Step 7. Otherwise, estimate the effect on the objective function value of branching on each unsatisfied variable. Decide which of these variables is the best to branch upon and create data for a new node on the search tree that would be created by branching. Go to Step 7.

Step 7: Return information for both the long and short node data structures to the master. Go to Step 3.

The algorithm on the master processor acted as a taskmaster and coordinator as follows:

Step 1: Read in the problem data and initialise the long and short node data structures. Initialise data structures to keep track of which transputers are in use. Go to Step 2.

Step 2: Supply data structures for the initial LP relaxation of the MIP problem to each of the three slaves. Go to Step 3.

Step 3: Send details of the initial LP relaxation to slave number one for solution. Indicate that slave one is now busy. Go to Step 4.

Step 4: Wait for a message from any of the slaves. If a message is received, go to Step 5.

Step 5: Receive the solution of the LP relaxation. Note which slave the message came from and indicate that it is now free for more work. If this was the first iteration and the result is that of the initial LP relaxation of the MIP, go to Step 6. Otherwise, go to Step 7.

Step 6: If the initial LP relaxation was unbounded, infeasible or worse than the initial cut-off supplied, then stop. If an integer solution was returned, stop, as this is the optimal solution. Otherwise, go to Step 10.

Step 7: If the LP problem was infeasible or worse than the cut-off supplied, go to Step 11. Otherwise, go to Step 8.

Step 8: If the solution was integer feasible, go to Step 9. Otherwise, go to Step 10.

Step 9: Update the cut-off and incumbent solution for the MIP problem. Remove candidates from the list of nodes to be branched on that would yield worse solutions than the present best. Go to Step 11.

Step 10: Create long and short node data structures for the new node indicated by the incoming message from the slave. Add the node created to the list of possible nodes to branch on. Go to Step 11.

Step 11: Check the size of the list of candidate nodes for branching. If the list is empty, go to Step 13, otherwise go to Step 12.

Step 12: See if there are any idle slaves. If not, go to Step 4. Otherwise, choose an idle slave, choose a node to be branched upon from the candidate list and recover the long node information from the disk. Combine the long and short node information and farm this to the chosen slave. Set this slave as busy and go to Step 11.

Step 13: If there are no transputers busy, stop. If no integer feasible solution has been found, then one does not exist. If one or more integer feasible solutions have been found, the incumbent solution is the optimal. If one or more transputers are busy, go to Step 4.

4.4. Implementation of a Parallel Algorithm on Transputers

In order to implement any parallel algorithm on one or more transputers, there are several stages that must be followed.

The first step after having designed the overall parallel algorithm is to design the necessary communicating parallel processes that will be used to implement it, e.g. the master and slave algorithms mentioned above.

The second step is to decide exactly how each parallel process communicates with other parallel processes and then construct a network of communications channels to do so. As an example, using the simple star topology shown above, the master algorithm sends messages to and receives messages from each of the slaves. It is thus necessary for the master to have four channels for sending messages and four channels for receiving messages (remembering that transputers send messages via unidirectional links). Each of the slaves needs only one channel to send messages and one to receive messages since it is only connected to the master, and does not communicate with other slaves.

The communication network would thus be as in Fig. 4.2:

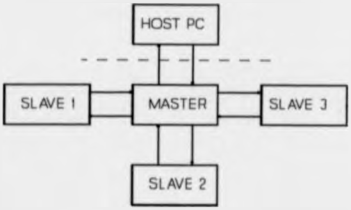


Fig. 4.2: Network of communicating parallel processes.

The third step is to ensure that a physical network of transputers can be constructed that is capable of implementing the communication network already developed. Software available with the Quintek Fast-9 transputer board allows the electronic configuration of the transputers on the board into any physical topology possible using the four links per transputer, although, by default, a pipeline connecting certain links of the transputers is always set up. The software must be used to set up a physical communication network between the transputers that are to be used. This physical network must include explicit details of which of the four links on each transputer are to be connected to which links on other transputers. It should be noted that all transputers present in the system must be present in the physical network, whether they are to be used by the parallel algorithm or not. This is because of the default pipeline always set up by the software. Fig. 4.3 below shows an example of a physical network that could be used to implement the parallel processes shown in Fig. 4.2.

The network of communicating processes originally designed must be correctly placed onto the physical network of transputers or the transputers will jam up and the program will hang. The correct parallel communicating processes must be placed onto the correct transputer so that the designed communications between processes can take place.

Once this has been successfully accomplished, the parallel algorithm can be used to solve problems. It should be noted however that the physical network of transputers must always be set up correctly before the parallel algorithm can be used. When the PC is turned on, the default physical connections of the transputers form a pipeline. If any other connection topology is desired, software must first be used to reconfigure the transputer connections.

Communication between transputers must occur via one of the four links by which they may be connected.

N.B. The numbers in Fig. 4.3 refer to the physical communication links of the transputers. Each transputer has four such links, numbered 0, 1, 2 and 3, although only the numbers of the links actually used by this topology are given in Fig. 4.3.

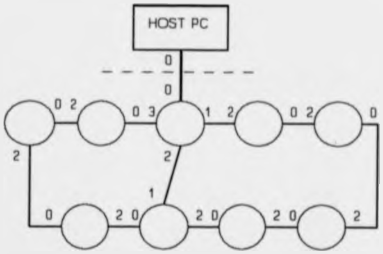


Fig. 4.3: Physical network of transputers.

4.5. Discussion of Initial Results

The initial code, using the star topology shown in Fig. 4.1, was tested on a variety of available MIP and combinatorial problems which are described in Table 4.2 below.

The AZx problems are different instances of a contract allocation problem. BAG882 is a chemical processing problem. HPW15 is the 15th example from Williams ([Williams, 1978]). INGT274 and INGT1345 are ingot casting problems. MO0788 is a power generation model. TAX1 and TAX2 are models of capital investment under different taxation regimes.

Problem	Category	NROWS	NCOLS	NGLOB
AZA	1	115	88	44
AZB	1	105	88	44
AZC	1	105	88	44
BAG882	5	304	304	23
HPW15	1	56	45	30
INGT274	2	13	274	274
INGT1345	3	19	1345	1345
MO0788	6	1123	926	24
TAX1	5	301	314	74
TAX2	5	181	194	34

Table 4.2: Test Problem Statistics.

The NROWS column gives the number of constraints, NCOLS the number of structural columns and NGLOB the number of discrete entities.

The possible categories of problem are: (1) Small Combinatorial Problems; (2) Medium Combinatorial Problems; (3) Large Combinatorial Problems; (4) Small MIP Problems; (5) Medium MIP problems; and (6) Large MIP problems.

The parallel code was parameterised so that it could be run with one, two or three slave processors. When the parallel code is run using only one slave processor, the same problem solution is obtained as when the serial code is run (and the same number of nodes is considered before optimality is proven). The times taken to solve the problem are different however. The parallel algorithm is such that the master processor is idle when it has farmed out an LP relaxation to a slave and the slave is idle once it has solved the LP (until it receives a new LP to solve). Thus, the only real difference between the time taken to solve a problem using the serial code and that taken using the one-slave parallel code should be caused by the message-passing overheads incurred. These overheads are small enough that the solution times obtained when using the one-slave parallel code are of the same order as those obtained when using the serial code, although the serial code is faster.

Problem	T0	T1	T2	T3
AZA	3.79 (7)	3.08 (11)	3.18 (13)	
AZB	455.67 (1901)	268.48 (1927)	233.71 (1919)	
AZC	55.36 (145)	28.51 (141)	21.04 (143)	
BAG882	389.48 (237)	127.10 (171)	121.55 (249)	
HPW15	2.03 (15)	1.76 (23)	1.65 (25)	
INGT1345	61.03 (59)	80.03 (115)	82.00 (123)	
INGT274	28.23 (99)	23.46 (157)	24.94 (165)	
MO0788	55103.12 (3917)	27567.43 (3961)	18147.60 (3959)	
TAX1	3215.56 (1649)	1626.83 (1649)	1101.04 (1655)	
TAX2	119.47 (165)	23.84 (41)	22.46 (51)	

Table 4.3: Results of Using the Initial Parallel Algorithm.

The entries in Table 4.3 give the time and the number of nodes (in parentheses) taken to solve the problem with 1, 2 or 3 transputers acting as slaves. Times are in elapsed seconds.

To demonstrate more clearly the success of using the implemented parallel algorithm on the test problems, we shall measure the **speedup** achieved by adding additional transputers.

The speedup is defined as T_1/T_x where T_x is the time taken to solve a problem using x slave transputers. The speedups achieved for the different test problems are shown in Figs. 4.4 and 4.5 below.

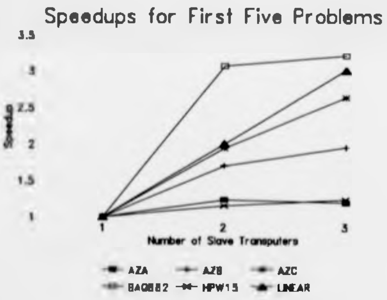


Fig. 4.4: The First Five Test Problems.

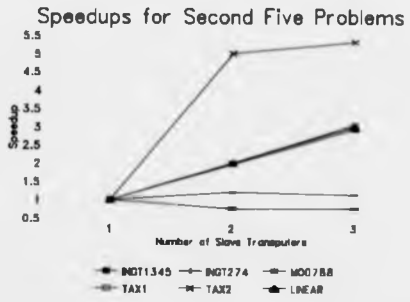


Fig. 4.5: The Second Five Test Problems.

In all the tests, all the available slave transputers were utilised at least once and the average slave transputer usage figures were very high indeed, never falling to eighty per cent, and settling in the high ninety percent range in most cases.

The test problems that benefit the most from the use of additional slave transputers are those in the categories of medium and large MIPs (i.e. the problems which the parallel algorithm was developed to attack). The medium sized MIP problem BAG882 gives a notably superlinear speedup with two and three slaves, as does the large MIP TAX2. The medium MIP TAX1 and the large MIP MO0788 both achieve almost exactly linear speedup with two and three slaves.

In comparison, the combinatorial problems tested do not fare well when additional slaves are added. The small combinatorial problems AZA and HPW15, the medium combinatorial problem INGT274 and the large combinatorial problem INGT1345 never really achieve a speedup much over one.

This indicates that a large message-passing overhead is being incurred in these cases so that the benefits of the extra processing power are being wasted. This is due to the LP relaxations being solved very quickly and the results being returned to the master processor before it has had a chance to farm out much more work to other slaves.

The small combinatorial test problems AZB and AZC fare better than the other combinatorial problems, with AZB achieving a speedup of just under two and AZC a speedup of two and a half. In these cases, the proportion of message-passing is not so high, indicating that the LP relaxations are not so easy to solve, thus allowing the master more time to properly farm out work.

Solving ten test problems cannot give enough empirical evidence on which to base conclusions about the performance of the parallel algorithm. It does however appear, at least in the case of the larger MIP problems for which the algorithm was developed, that the benefits of parallelising the algorithm are being exploited.

1. Development of the n-transputer Parallel Algorithm

As mentioned in the previous chapter, at most three slave transputers may be **directly** connected to the root (or master) transputer whilst implementing the farming algorithm. One of the root transputer's links must be connected to the PC to enable input/output operations such as reading from and writing to the screen or disk.

To progress beyond the Daniel four-transputer algorithm and produce a parallel algorithm for use on n transputers, some form of message-passing system would have to be used, to enable slaves not directly connected to the master to communicate with it indirectly.

The Parallel Fortran compiler purchased from 3L Ltd ([3L Ltd., 1988]) for use in the previous experiments, also provides a **flood-fill configurer** whose function is to ease the construction and administration of larger farming applications. It creates an arbitrary network of anonymous transputers for the farm, keeps constant track of the availability of slave processors, and handles all message-passing to and from the slaves.

When trying to implement an eight-slave farm using the flood-fill configurer however, problems arose when the broadcast of the initial LP relaxation of the MIP to all the slaves was attempted. The flood-fill configurer system provides the user with no information as to which slave transputer a message is destined for or being returned from, i.e. the slaves used are anonymous. It was thus impossible to guarantee that every slave transputer had received the required information packet. Indeed, the use of the basic debugging facilities possible indicated that only a subset of the eight slave transputers were receiving the required broadcast. The flood-fill configurer therefore had to be abandoned, and a complete message-passing harness and administrative system designed to replace it.

5.1. The Message-Passing Harness

The decision to develop the message-passing harness was also influenced by a desire to have more control over the destination of individual LP problems than the flood-fill configurer would have provided. When considering the implementation of the parallel algorithm on a large number of transputers for instance, it might be useful to consider sending descendent LP relaxations to the same slave that had previously solved the parent LP relaxation. Most of the data held on the slave would still be valid, with the exception of the bounds on the variable being branched on and certain book-keeping data structures. This information would be all that needs to be passed to the slave (possibly along with a new cutoff value if this has changed since the last LP was sent).

5.1.1. The Topology Mapping System

In order to implement a message-passing system, the anonymity of the slaves was removed, and the master and slaves were given explicit knowledge of the experimental topology in use. This provided for the possible implementation at a later stage of an adaptive transputer selection strategy. It might, for instance, be desirable to keep a transputer free if the next LP to be solved on it is a descendent of the one last solved by it, so that almost all of the data on the slave is still valid.

The message-passing system developed involves placing a **map** of the current topology onto each transputer. Each transputer is allocated a number, and messages to be passed around the system contain a header indicating to which transputer they should be passed. On receiving a message, a transputer either makes use of the data or passes it on. If the transputer is to pass on data, it reads its map to find the next destination of the data.

The map consists of two arrays, a **link** array, (indicating which transputer is connected to each of this transputer's four communication links), and a **chart** array (indicating which transputer the message should next be passed to in order to finally arrive at its destination). This chart array is fixed (for simplicity), but a further step perhaps necessary on a system with hundreds of slaves would be to provide an adaptive chart that would choose alternative routes depending on the present traffic flow.

As an example of the use of the map, Fig. 5.2 shows the link and chart arrays used to traverse the simple connection topology displayed in Fig. 5.1.

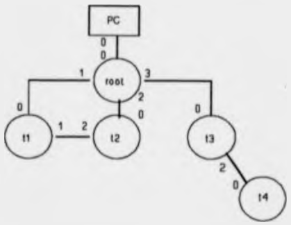


Fig. 5.1: A simple connection topology.

Transputer 0

actual link	0	1	2	3	
link array	H	1	2	3	
chart array	0	1	2	3	3
destination of message	0	1	2	3	4

Transputer 1

actual link	0	1	2	3	
link array	R	2	0	0	
chart array	R	0	2	R	R
destination of message	0	1	2	3	4

Transputer 2

actual link	0	1	2	3	
link array	R	0	1	0	
chart array	R	1	0	R	R
destination of message	0	1	2	3	4

Transputer 3

actual link	0	1	2	3	
link array	R	0	4	0	
chart array	R	R	R	0	4
destination of message	0	1	2	3	4

Transputer 4

actual link	0	1	2	3	
link array	3	0	0	0	
chart array	3	3	3	3	3
destination of message	0	1	2	3	4

Fig. 5.2: The LINK and CHART arrays used to traverse the topology shown in Fig. 5.1.

The four elements of each of the link arrays shown in Fig. 5.2 give information as to what each of the links of the transputer is connected to. A letter H indicates a direct connection to the PC host (obviously from the root transputer only), an R indicates a direct connection to the root transputer. The number 0 indicates that no direct connection is made via that link, and any other number indicates that the link is directly connected to the transputer with that number.

The chart arrays shown in Fig. 5.2 are designed for use with an n-transputer system where the root transputer is numbered as zero and there are (n-1) slaves, numbered 1 to (n-1). Each of the n members of a chart array give information as to where to next pass a message in order to reach the transputer numbered n. The letter R indicates that the message should next be passed to the root transputer. The number 0 indicates that no message will ever be passed to that particular destination next (i.e. it is to be used when referring to the transputer on which the map resides, since no message is ever passed from a transputer to itself). Any other number indicates that the message is next to be passed to the transputer with that number in order to get to its final destination.

Consider, as an example, the process involved in passing a message from transputer 4 to the root.

The four elements of each of the link arrays shown in Fig. 5.2 give information as to what each of the links of the transputer is connected to. A letter H indicates a direct connection to the PC host (obviously from the root transputer only), an R indicates a direct connection to the root transputer. The number 0 indicates that no direct connection is made via that link, and any other number indicates that the link is directly connected to the transputer with that number.

The chart arrays shown in Fig. 5.2 are designed for use with an n-transputer system where the root transputer is numbered as zero and there are (n-1) slaves, numbered 1 to (n-1). Each of the n members of a chart array give information as to where to next pass a message in order to reach the transputer numbered n. The letter R indicates that the message should next be passed to the root transputer. The number 0 indicates that no message will ever be passed to that particular destination next (i.e. it is to be used when referring to the transputer on which the map resides, since no message is ever passed from a transputer to itself). Any other number indicates that the message is next to be passed to the transputer with that number in order to get to its final destination.

Consider, as an example, the process involved in passing a message from transputer 4 to the root.

The chart array on transputer 4 indicates that in order to get to the root, the next destination is transputer 3 (since CHART(0)=3). Searching the link array indicates that link 0 of transputer 4 is directly connected to transputer 3 (since LINK(0)=3). Thus, the message is passed through link 0 of transputer 4 and next arrives at transputer 3.

The chart array on transputer 3 indicates that in order to get to the root, the next destination is the root itself (since CHART(0)=R). Searching the link array indicates that link 0 of transputer 3 is directly connected to the root (since LINK(0)=R). Thus, the message is passed through link 0 of transputer 3 and next arrives at its final destination, the root.

5.1.2. The Clear Path Method for Message-Passing

The transputers that were available to us had access to only 1 Mbyte of personal memory. Since the application was designed to attack large scale MIP problems (whose data structures can take up much space), the effective use of transputer memory both on master and slave is essential. The master transputer has access to the PC as well as its own personal memory and so can save data on disk if necessary, but the slave transputers have no such backup.

The message-passing routines to be placed on the slaves were thus designed to input only one (or in certain infrequent circumstances two) four-byte words of information at a time onto a slave before passing them on to their next destination. The harness thus allows the slave transputer to solve large LP problems and provide message-passing facilities whilst still only accessing its 1 Mbyte of personal memory.

This method is described as the clear path method since it is necessary for the whole message to reach its destination before it can be acted upon. Thus, since the message is only partly stored by each transputer that it passes through, a clear path (possibly across a number of transputers) is necessary to get the message to its destination.

The use of the clear path method by the harness removes the need for a transputer to hold all of the data structures representing an LP relaxation at once during message-passing. Without the Clear Path Method, in the worst possible case, as seen in Fig. 5.3, enough space would be needed to hold three different sets of details on a slave.

An LP relaxation is held in an outward bound buffer of the look process (i.e. being passed away from the master), an LP solution is being held in an inward bound buffer of the look process (i.e. being passed towards the master) and the details of an LP relaxation (and possibly of its solution) are being held as part of the calculation process.

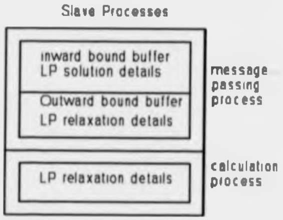


Fig. 5.3: Worst possible case for message passing.

The two main problems faced when implementing the clear path method are that a clear path must be established for each communication and that steps must be taken to avoid deadlocking the message-passing algorithms.

<u>5.1.2.1. Establishing a Clear Path</u>

Since all of a message must reach its destination before it can be acted upon, it is essential that a clear path through the network of transputers be available to the destination in question each time a message is sent.

A system allocating priorities to different types of messages passing through the slaves has been designed to help achieve a clear path for each message, although this does slow down the overall message-passing process somewhat due to the overheads involved. Full details of the priority system are given in section 5.2.2.

Unfortunately, the clear path method of message-passing will not be very valuable for use with large numbers of transputers. It was designed to work with the small number of transputers available, but the difficulty of obtaining a clear path will obviously increase when more slaves are added to the topology. A message-passing harness has thus been designed to replace the clear path harness when more processors become available for experimentation (or if more personal memory can be acquired for each transputer). Basic details of this new design are contained in Chapter Eight.

For the present experiments using the nine-transputer board however, a choice of connection topology minimising the average distance of slave transputers from the master (and thus minimising the average necessary length of a clear path) is used to increase the likelihood of getting a clear path and to combat the delays caused by communication overheads.

<u>5.1.2.2. Avoiding Deadlock Problems</u>

Deadlock problems occur when two separate parallel processes come to a point in their algorithms where they try to communicate with each other, but for some reason they cannot.

Consider the case where task A is at the point in its algorithm where it must send a message to task B and then act upon the details of a return message. If task B is at a similar point in its algorithm in that it is attempting to send a message to task A and then receive a return message, then neither task A nor task B can proceed since there is no way of breaking the communications deadlock.

When all the slaves were directly connected to the master there was no risk of the master attempting to send a message to a slave but finding that the slave is already occupied trying to send a message to the master (as shown in Fig. 5.4 below).

This situation can now occur if a solution is being passed back to the master via the slave to which the master is trying to send a new LP relaxation. Unless precautions are taken to avoid such deadlocking, the algorithm will seize up.

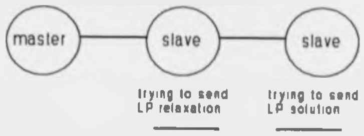


Fig. 5.4: A possible deadlock situation.

Timed processes have been placed on the master and slave transputers in order to avoid deadlocking. When the master processor is sending out an LP relaxation, each 4 byte word of data sent is timed. Since the only thing to stop an outgoing message would be a block to the path, if a word takes too long to send, the sending process is abandoned and the node for the LP relaxation is returned to the list of candidates as though it had not yet been selected. The time allowed for sending a word of data may be specified by the user, but a minimum time must be allowed, based on a communication speed of 1.8 Mbytes per second in one direction for the T800 transputer links ([Inmos Ltd., 1988]).

The message-passing routines on the slave processors time the arrival and departure of LP relaxation messages (using the communication speed set by the user). If part of an LP relaxation cannot be sent during the allotted time, the sending process is abandoned, and the message-passing process on the slave is reset, after a short delay, to look for other messages. This resetting process will filter back all the way to the slaves directly connected to the master.

Full details of the timed processes on the master and slave transputers are given in sections 5.2.1. and 5.2.2. respectively.

5.2. The n-transputer Parallel Algorithm Developed

Both the master and slave algorithms were adapted from the previous 4-transputer algorithms in order to implement the clear path method of message-passing without incurring problems with deadlocking.

5.2.1. The Master Algorithm

The algorithm on the master processor has been expanded since it is to send messages to slaves not directly connected to the root transputer.

In order to avoid the potential deadlock problems mentioned previously, the algorithm on the master processor has been adapted to include a timed process for sending out LP relaxations to slaves. A time interval is set, during which one word (i.e. 4 bytes) of the LP relaxation data must be sent. If the sending routine cannot transmit the word of data during its allotted time, the master algorithm gives up on sending the whole LP relaxation and waits for an incoming message from a slave. The only time when this should occur is when the LP relaxation data has not been able to follow a clear path all the way to its destination.

The process of timing each individual word of data sent is sufficient to ensure that the message has time to reach its destination since the messages are many words in length and the path from master to the furthest slave is short.

The time needed to transmit a 4 byte word of data is either set by the user or calculated using the assumption of a message-passing speed of 1.8 Mbytes per second for the T800 transputer links.

The master algorithm also includes processes which ensure that data structures are not corrupted if the last LP relaxation chosen from the list of candidates cannot be sent after all. These consist of saving the original values of certain data structures and thus being able to reset any values that were changed by the node selection or disk reading processes if the node chosen cannot subsequently be sent to a slave.

After reading in the IP problem information and initialising its data structures, the new master algorithm goes through the following steps:

Step 1 Send out initial broadcast to each of the slaves.

Step 2 Send out a first LP relaxation of the MIP problem arrived at by relaxing all integrality constraints.

Step 3 Get results of first LP relaxation. Stop if the problem is unbounded or infeasible or if the solution is worse than the cut-off value initially set up. Otherwise, create a node and add it to the candidate list.

Step 4 Look for a free transputer. If there are none, wait until a result is returned and then go to Step 11.

Step 5 Attempt to choose a node to branch on from the candidate list. If the candidate list is empty, look to see if any transputers are busy. If so, wait for a result to be returned and go to Step 11. If not, STOP. If an integer feasible solution has been found, take it as the final result. If no integer feasible solution has been found so far, none exists.

Step 6 Check for incoming results. If there are none, save the values of variables that will be updated when the LP is sent out, and go to Step 7; If results are being returned, go to Step 11.

Step 7 Read the long node information about the node from disk. Save the values of any further variables that will be updated when the LP is sent out. (The cycle is such that there are several stages at which it can be interrupted by an incoming result, thus necessitating the retrieval of some previous variable values. Variables are thus updated, and their previous values saved, only if there are no incoming results. This minimises the overall amount of retrieval of variable values). Apply the new bounds to the variable to branch on.

Step 8 Check for incoming results. If there are some, restore any changes made to the data structures and go to Step 11. If not, go to Step 9.

Step 9 Attempt to send out the modification to the base case LP to the slave chosen and update the status indicator for that transputer and the data structures. If the LP has not been sent out after a given amount of time (as set by the user) then restore any changes made to the data structures and go to Step 11. (This will only happen if a directly connected slave tries to send back a result after the master is already committed to sending out an LP problem).

Step 10 Look for incoming results. If there are some, go to Step 11. If not, go to Step 4.

Step 11 Get in an LP solution. If the LP solution is worse than the present cut-off or is infeasible, go to Step 4. If the LP solution is feasible but contains a number of integer infeasibilities, then create a new node to hold the details. Enter this node into the candidate list and go to Step 4. If the LP solution is feasible and contains no integer infeasibilities (i.e. if it is a solution to the IP), then write this solution to disk, update the cut off, remove any nodes from the candidate list that will not produce better integer feasible solutions and go to Step 4.

5.2.2. The Slave Algorithm

In order to implement the n-transputer parallel algorithm, each slave processor runs two concurrently executing processes. The first is the message-passing (or **look**) process, and the second is the LP-solving (or **calculation**) process. When such concurrent processes are run on a transputer, the programmer is allowed to give priorities to the processes. A high priority process will run until it has finished, whereas a low priority process will only run for a given amount of time, or until it finishes or is interrupted by the need to start up a high priority process. In order to properly implement the clear path method of message-passing, it was necessary to give a high priority to the look process, so that messages would be passed on properly without the look process being interrupted. The calculation process must not attempt to interrupt the message-passing and thus was given a low priority.

5.2.2.1. The Look Process

The look process polls the four external communication links of the slave transputer and an internal communication channel with the calculation process until an incoming message is detected.

If the message is to come from the calculation process (and hence can only be an LP solution), then the look process repolls the external channels to ensure that no messages are incoming before giving the go ahead to send out the LP result. This gives a higher priority to messages from other transputers than to newly generated LP solution messages. The external channels on each slave are polled in a specific order (provided in addition to the map information on each slave). The inward facing channels (i.e. those connected to the master or to a slave closer to the master) are polled before the outward facing channels. This is in order to ensure that priority is given to LP solution messages returning to the master.

If the message to be input originates from another transputer, then the look process must first read the header to determine what type of message is to follow and for which transputer the message is intended.

Messages from other transputers can take the form of:

initial broadcast messages of the initial LP relaxation of the MIP problem which are bound for other transputers;

the initial broadcast message intended for this transputer;

LP relaxations to be passed on to other slaves;

an LP relaxation to be solved on this transputer; or

LP solutions to be passed back to the master.

Once the look process has ascertained the destination and type of an incoming message it simply inputs the message one word at a time and then attempts to pass the word input through the appropriate link or internal channel (after referring to the map).

The broadcast message is of particular importance to the look process, as it contains the basic data for the MIP problem (e.g. the number of rows and columns etc.). Many of these general details are used to define the size of data structures that are passed around the system as later parts of the general broadcast or subsequently as part of the LP problem and/or solution information. Thus, once a slave transputer has received its broadcast of general information, it should know the sizes of all the arrays to be passed around and used later.

LP relaxation messages are also of special importance to the look process, since they must be dealt with in a particular way to avoid deadlocking the message-passing harness.

If the message to be input is an LP relaxation message, the look process allows each 4-byte word of the message a certain amount of time to arrive. If a word arrives successfully, the look process attempts to pass it on to its next destination. If the look process of the next transputer on the path cannot receive the word because it is already occupied trying to send a message inward, a delay is caused. This delay will filter back down the path to the master, so that the message being passed is halted. Since the master process sending out the LP relaxation is also timed, it will be abandoned once the blockage has been detected. The master algorithm will then be reset to look for incoming messages, thus allowing the inward bound message that caused the blockage to return to the master successfully.

5.2.2.2. The Calculation Process

The calculation process is very similar to the initial four-transputer slave algorithm in that it waits for an LP relaxation to be sent to it (albeit this time via the look process) and then reacts accordingly. It either reads in the MIP data if a broadcast has just been received, and then waits for another message, or it reads in the modifications to its base case LP if it is to perform a calculation. In the latter case, it proceeds to perform iterations of the Simplex algorithm until it has solved the LP or has ascertained that the LP is infeasible or unbounded. If a solution is found to the LP, the calculation process determines whether its value is better or worse than the present best solution. If the solution found is better than the present best, the calculation process checks to see if the solution is integer feasible. If the solution is integer feasible (and has already proved to be better than the present best solution), it is considered a contender for the new best solution. If the solution is not integer feasible, the calculation process performs the work to estimate the solutions to be arrived at by branching again on the unsatisfied variables.
The results of attacking the LP relaxation are returned to the look process as soon as it will accept them and then forwarded to the master. The calculation process then waits for another LP problem to arrive from the look process.

5.3. Computational Results

The node selection strategy used for the initial testing of the nine-transputer algorithm was identical to that used for the four-transputer algorithm.

Problems of various sizes and complexities were used to test the n-transputer parallel algorithm. In addition to those problems used to test the four-transputer algorithm, the problems shown in Table 5.1 were attacked. CHAL is a local heating load and distribution planning model. CRAC is an oil refinery planning model. DAAC and OK are farm planning problems. DOM1 is a ship scheduling model. GY is a medium term energy planning model. G31 and G32 are petrochemical plant models. MCA, MRX and MR1 are political districting problems. SETX is a project evaluation model. Note that Table 5.1 also gives the time (in seconds) and the number of nodes taken to solve the problems using a serial version of the optimiser code.

Problem	Category	NROWS	NCOLS	NGLOB	NSETS	NSOSM
CHAL	3	985	1320	552		
CRAC	6	294	785	6		
DAAC	4	80	149	31		
DOM1	6	796	585	11	11	41
GY	3	913	888	528		
G31	4	159	146	9		
G32	4	162	148	9		
MCA	6	412	648	22	3	15
MRX	2	166	192	143		
MR1	2	166	192	143		
OK	4	80	149	31		
SETX	4	13	21	3	3	18

Table 5.1: Extra Test Problem Statistics.

The NROWS column gives the number of constraints, NCOLS the number of structural columns, NGLOB the number of discrete entities, NSETS the number of Special Ordered Sets, and NSOSM the number of Special Ordered Set Members.

The problem categories are the same as those for the previous test problems: (1) Small Combinatorial Problems; (2) Medium Combinatorial Problems; (3) Large Combinatorial Problems; (4) Small MIP Problems; (5) Medium MIP Problems; and (6) Large MIP problems.

The code was again parameterised to allow testing with from one to eight slave transputers in order to see how useful the number of extra nodes generated by using more processors were in each case.

The topology shown in Fig. 5.5 below was used to produce the results. The topology was chosen to provide a good likelihood of a clear path for most messages, in that the paths from the master to those slaves not directly connected to it are as short as possible.

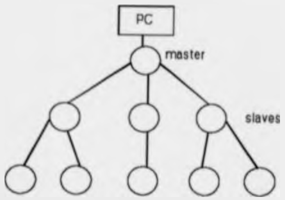


Fig. 5.5: Topology used to test the n-transputer algorithm.

Reproducibility of Results

Before discussing the computational results obtained, it must be pointed out that the implementation of our parallel algorithm is non-deterministic. Hence, the algorithm will not necessarily enumerate exactly the same solutions each time a problem is attacked. If the same solutions are enumerated, they will not necessarily be considered in the same order. Thus, although the same final solution will always be obtained, it may take a different amount of time and/or a different number of nodes to arrive at.

The non-determinism of the algorithm is caused by the asynchronous coordination of the slaves. The specific effects of non-determinism exhibited when the algorithm is implemented are due to the hardware used.

The major causes of the non-determinism of the implementation of our algorithm are the disk-reading and disk-writing operations carried out by the master. Each time a record of long node data is to be written to the temporary file held on the hard disk, the operating system of the PC must decide where on the disk the new record is to be stored. Different runs of the program will result in different parts of the disk being decided upon by the operating system. Obviously, deciding upon and writing to different places on the disk will take different amounts of time. The time taken to write records to the disk has an effect on the number of new LP relaxations that can be farmed out by the master before it is interrupted by incoming LP solutions. Obviously, if more than one slave is busy, the longer it takes the master to write the results of an incoming solution to disk, the less time is left to farm out new LP relaxations to idle slaves before another solution is returned.

Similarly, once the master has chosen a new node to be branched upon, the amount of time taken to find and read the appropriate long node record from the disk partially determines whether there will be enough time to send out the LP relaxation before an LP solution is returned, thus interrupting the process.

So, the disk-reading and disk-writing operations of the implementation of our algorithm can lead to different patterns of transputer usage and different numbers of messages being passed around the system. This latter effect itself leads to a variation in the amount of time taken for some slaves to solve their LP relaxations.

Since the calculation process on each slave is a low priority process, it operates in a certain way. Once it has received an LP relaxation, it will attempt to begin its calculations. It will only be able to do so, however, if the high priority look process does not have any more messages to pass. Once work has begun on the LP relaxation, the performance of the calculation process will be partly determined by the number of messages being passed around by the slave. Each time a message must be passed by the high priority look process on the slave, the low priority calculation process is interrupted. The calculation process is stopped as soon as it is safe to do so, i.e. so that no calculation results are corrupted. Where the calculation process stops depends entirely on the exact moment it is interrupted by the look process. Thus, different numbers of messages being passed around the system also leads to LP relaxations taking different amounts of time to solve.

The overall effect of the non-determinism is that the computational results discussed below cannot definitely be reproduced time after time. Any conclusions reached from the results are thus reached with this point in mind.

In the discussion of these results we shall again use the concept of **speedup**. (Speedup is defined as T_1/T_x where T_x ($x = 1,\ldots,n$) is the time taken for the parallel algorithm to solve the MIP problem when using x slave transputers).

The computational results of the initial tests are listed in Appendix 2A, and the speedups achieved are shown in Figs 5.6 to 5.11 below.

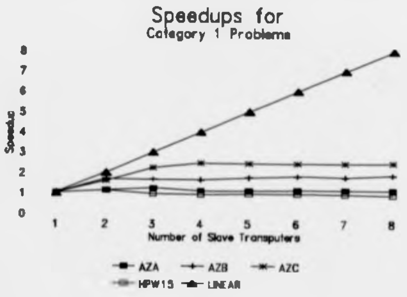


Fig. 5.6: Speedups for Small Combinatorial Problems.

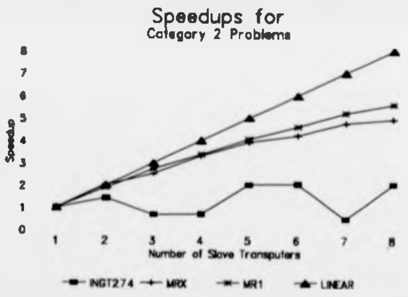


Fig. 5.7: Speedups for Medium Combinatorial Problems.

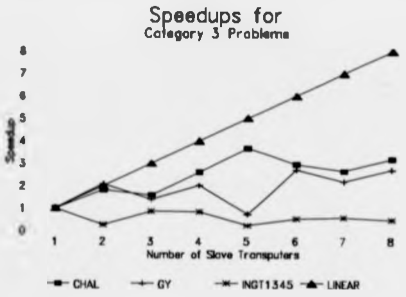


Fig. 5.8: Speedups for Large Combinatorial Problems.

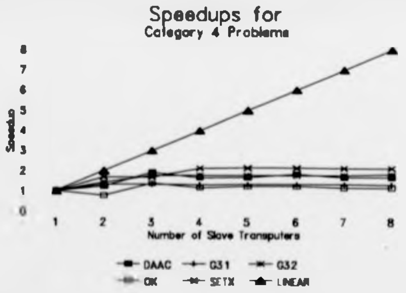


Fig. 5.9: Speedups for Small MIP Problems.

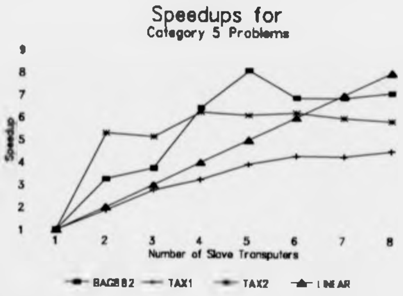


Fig. 5.10: Speedups for Medium MIP Problems.

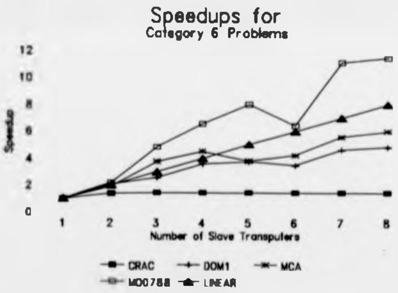


Fig. 5.11: Speedups for Large MIP Problems.

Since the n-transputer parallel Branch and Bound algorithm was designed to attack medium to large scale MIP problems, we would anticipate that attacking such problems would produce good results, with less impressive results being obtained when attacking problems from the other categories.

We shall now discuss the results obtained from attacking the different categories of problem.

Problem Category 1: Small Combinatorial Problems

As can be seen from Fig. 5.6, the problems from this category never achieved a speedup of greater than about two, no matter how many slave transputers were used. The reason for this, in all the cases, is that only a small number of the available slave transputers were actually used, as can be seen from Table 5.2. The point after which no more transputers are used, no matter how many are available, is referred to as Saturation Point.

Problem	Maximum No. of Slave Transputers Used	Average Transputer Usage after Saturation Point
AZA	4	2.5
AZB	4	1.8
AZC	5	2.6
HPW15	3	1.7

Table 5.2: Transputer Usage for Small Combinatorial Problems.

Problem Category 2: Medium Combinatorial Problems

Fig. 5.7 shows that problems MRX and MR1 produce a reasonable, although sublinear speedup curve, whilst problem INGT274 performs similarly to the smaller combinatorial problems previously mentioned.

The reason that problems MRX and MR1 give moderately good results is probably that the available slave transputers are highly utilised. All the available slave transputers are used in each case, with the overall usage falling from one hundred percent (when one slave transputer is used) by only two to four percent with the addition of each further slave. For both problems, when all eight slave transputers are available, roughly eighty percent of their processing power is utilised, with the remainder being lost due to message-passing and other overheads. There is, at most, a twenty three percent increase over the single-slave case in the amount of the search tree considered for problem MRX and only a six percent increase for problem MR1. These increases in the amount of calculation are adequately absorbed by the extra processing power available.

The poor speedup curve for problem INGT274 is due to the fact that at most three slave transputers are used, and on average only one and a half are used when three or more are available.

Problem Category 3: Large Combinatorial Problems

The speedup curves shown in Fig. 5.8 for problems CHAL, GY and INGT1345 are again poor, especially in the case of INGT1345 which never achieves a speedup of one.

The particularly poor showing of problem INGT1345 is probably due (similarly to many of the smaller combinatorial problems) to the fact that it only makes use of a fraction of the available slave transputers. At most four slaves are used, with an average of 1.8 being used after three or more are available. The poor speedup figures are also due to the fact that the single-slave run finds the optimal solution in relatively few nodes and is thus hard to better.

Problems CHAL and GY only make use of three to four slave transputers on average, and thus cannot achieve speedups of more than three or four respectively.

Problem Category 4: Small MIP Problems

The small MIP problems shown in Fig. 5.9 suffer from the same problems as the small combinatorial problems previously discussed.

As can be seen from Table 5.3, only a fraction of the available slave transputers are used on average. Again, the point after which no more transputers are used, no matter how many are available, is referred to as Saturation Point.

Problem	Maximum No. of Slave Transputers Used	Average Transputer Usage after Saturation Point
DAAC	3	1.7
G31	2	1.3
G32	3	1.8
OK	4	1.8
SETX	4	1.9

Table 5.3: Transputer Usage for Small MIP Problems.

Problems G31, G32 and SETX are all solved very quickly indeed in all cases (e.g. in three, six and two seconds at most respectively), and thus speedup is hard to achieve. This is probably due to the very small number of integer variables in these problems. Problems DAAC and OK, although making poor use of the processing power available, do manage to drastically decrease the number of intermediate integer solutions found before the optimum.

Problem Category 5: Medium MIP Problems

Fig. 5.10 shows that superlinear speedup is achieved for problems BAG882 and TAX2 when using between two and six slave transputers, although the speedups become more or less constant after this point. Problem TAX1 achieves a reasonable, if not linear speedup.

BAG882 performs particularly well, such that at its best, the problem is being solved in fifteen percent of the single-slave time. The available slave transputers are only moderately well utilised, with the average usage never exceeding five. The number of intermediate integer solutions and the overall number of nodes searched are notably reduced, however, from the single-slave case.

TAX2 shares the characteristics of BAG882 in that it too achieves a notable reduction in the number of intermediate integer feasible solutions found and nodes searched whilst only using at most four transputers on average.

TAX1 searches more or less the same number of nodes in each case, although the process is faster with more slave transputers allowing a speedup of up to four and a half (with eight slave transputers available).

Problem Category 6: Large MIP Problems

As can be seen in Fig. 5.11, problem MO0788 exhibits superlinear speedup when from two to eight slave transputers are used and problem MCA exhibits superlinear speedup when between two and four slaves are used. Problem DOM1 fares worse however, although it manages a reasonable, if sub-linear speedup. Problem CRAC consistently shows a speedup of about one and a half, no matter how many slaves are used.

The results for problem MO0788 seem to be caused by the extremely high average usage of the slave transputers, which never falls to the ninety percent mark. In the best case, (i.e. when using eight slaves) this allows the reduction of the solution time to only eleven percent of the single-slave time. Fewer nodes are consistently searched overall when using two or more slaves than in the single-slave case.

MCA also searches fewer nodes than the single-slave case when two to four slaves are used (i.e. when the superlinear speedup is achieved). In the three and four slave cases, just under half the nodes are searched. Again, the usage of the slave transputers is high (i.e. between eighty six and ninety three percent) in the cases where superlinear speedup is achieved. Unfortunately, an average usage of just over five transputers is the best that can be managed overall, so speedup is limited to about five or six when five or more slaves are used.

DOM1 searches more or less the same number of nodes in each case, although the process is faster with more slave transputers since the average usage of the available slaves never falls below seventy percent. This allows a speedup of up to five (with eight slave transputers available).

CRAC searches twice as many nodes in half the time of the single-slave case when using three to eight slaves. No more than two and a half slaves are used on average however, thus limiting the overall speedup achievable to a stable one and a half.

Discussion of Results

Most of the poor results seem to have occurred because the potential benefits of the parallel Branch and Bound algorithm have been squandered. Only a fraction of the available processing power has been used to attack most of the small problems and some of the larger problems. This is due to several factors, any one of which can cause a bottleneck at the master processor, so that only a small number of LP relaxations can be farmed out to idle slaves.

The first such factor is the ease of solution of the LP relaxations of a problem. Obviously, if a subproblem can be solved very quickly, it is possible to return the resulting information to the master processor before it has had time to farm out many more subproblems. Thus, if many of the subproblems are easy to solve, it is likely that some of the available slave processors will remain idle.

In order to determine how hard the LP relaxations of the test problems are to solve, the single-slave code was used to measure the average number of Simplex iterations performed per LP relaxation and the average time taken to solve an LP relaxation for each test problem.

The single-slave code was used by the timing process, so that the exact path of all messages is known and there will be no interruption of the decision-making process due to returning LP solutions. The single-slave algorithm is also deterministic since no asynchronous control of multiple processors is actually carried out.

These two values are used to calculate the average time per Simplex iteration for each test problem, which is used as an indicator of how hard the LP relaxations are to solve.

Problem	Average Number of LP iterations per relaxation	Average Time per LP relaxation (ms)	Average Time per LP iteration (ms)
AZA	13.57	452.86	33.37
AZB	5.77	138.42	23.99
AZC	3.13	288.37	92.20
HPW15	8.00	70.67	8.83
INGT274	8.49	192.54	22.67
MRX	47.41	2482.39	52.36
MR1	46.80	2495.04	53.32
CHAL	18.28	5518.75	301.83
GY	11.61	2817.02	242.70
INGT1345	9.00	806.13	89.57
DAAC	5.57	130.39	23.40
G31	16.67	986.67	59.20
G32	15.60	1000.00	64.10
OK	4.94	136.92	27.74
SETX	3.41	36.90	10.81
BAG882	11.28	1875.49	166.31
TAX1	18.98	1709.12	90.07
TAX2	12.75	584.29	45.82
CRAC	25.00	1572.50	62.90
DOM1	48.38	6716.21	138.82
MCA	33.55	2988.46	89.08
MO0788	65.36	15687.84	240.03

Table 5.4: Times for LP iterations.

It can be seen from Table 5.4 that the really hard problems to solve are the large combinatorial problems CHAL and GY and the large MIP problem MO0788. These are followed by the medium-sized MIP problem BAG882 and the large MIP problem DOM1, which will be categorised as reasonably hard. Of these five hard problems, the MIP problems achieve much better speedups than the combinatorial problems. The MIP problems BAG882 and MO0788, both of which exhibit superlinear speedup, search fewer nodes when two or more slaves are used than in the single-slave case, indicating that the single-slave search was relatively poor. The combinatorial problems CHAL and GY search at least the same number of nodes if not more when more than one slave is used. The MIP problem DOM1 attacks roughly the same number of nodes no matter how many slaves are used, but this is probably due to the extremely small proportion of integer variables (i.e. eleven of five hundred and eighty five variables) in this case, which does not allow much variation in the search.

A second factor in the creation of bottlenecks at the master processor is the time spent reading the long node information from the disk once a node has been chosen. The disk-accessing process accounts for a large proportion of the total master algorithm time. As can be seen from Table 5.5, the disk-reading time is also large when compared with the average time needed to solve an LP relaxation for many of the test problems.

Problem	Time to Read Long Node Information TLN (ms)	Ratio of Average Time per LP relaxation to TLN
AZA	38.08	12:1
AZB	41.14	3:1
AZC	40.66	7:1
HPW15	34.93	2:1
INGT274	67.12	3:1
MRX	58.00	43:1
MR1	58.00	43:1
CHAL	190.06	29:1
GY	163.70	17:1
INGT1345	220.87	4:1
DAAC	41.14	3:1
G31	42.67	23:1
G32	42.91	23:1
OK	41.14	3:1
SETX	29.15	1:1
BAG882	58.48	32:1
TAX1	63.68	27:1
TAX2	48.40	12:1
CRAC	79.35	20:1
DOM1	94.25	71:1
MCA	79.97	37:1
MO0788	127.40	123:1

Table 5.5: Disk reading times in relation to Simplex iterations.

The times shown above are calculated assuming an average access time of 26ms and a data transfer rate of 83.75 Kbytes per second on the PC.

The bad results for the small test problems are partially explained by the large amount of time taken to read the disk relative to the time taken to solve an LP relaxation. The number of slaves used on average on the small problems is low, indicating that there was not enough time to farm out many LP relaxations before results were returned.

On the other hand, Table 5.5 shows that for each of the test problems which achieved a near-linear or superlinear speedup, the ratio of the average time per LP relaxation to the disk reading time is high. Similarly, the relatively good results for combinatorial problems MRX and MR1 are explained by a combination of the ease with which their LP relaxations are solved and the relatively short time taken to read the long node information from the disk. It is possible to search the tree very quickly in these cases.

A final factor relating to bottlenecks on the master processor is the accumulation of message-passing overheads. The more time that the master processor spends sending messages, the less time it has to choose a new subproblem to be farmed out before an LP solution is returned, demanding its attention. In the case of a slave processor, the longer messages take to reach it and to be read by it, the longer it remains idle. It is thus desirable to reduce the message-passing times as much as possible. It can be seen from Table 5.6 however, that the message-passing overheads are nearly negligible in the experiments, since when using the topology stated it only takes 17.36 ms to pass the longest message back to the master from the furthest slave (assuming that there is a clear path), and this is far above the average time of 4.41 ms.

The times given in Table 5.6 below are calculated assuming a message-passing speed of 1.8 Mbytes per second for the T800 transputer links and a maximum journey of two transputer links.

Problem	Maximum Message Passing Time from master to slave (ms)	Maximum Message Passing Time from slave to master (ms)
AZA	1.30	1.42
AZB	1.26	1.36
AZC	1.26	1.36
HPW15	0.74	0.86
INGT274	3.60	3.72
MRX	2.80	2.90
MR1	2.80	2.90
CHAL	14.52	14.62
GY	12.18	12.28
INGT1345	17.24	17.36
DAAC	1.30	1.42
G31	1.44	1.54
G32	1.46	1.56
OK	1.30	1.42
SETX	0.24	0.34
BAG882	2.84	2.94
TAX1	3.30	3.42
TAX2	1.94	2.06
CRAC	4.68	4.80
DOM1	6.02	6.12
MCA	4.74	4.86
MO0788	8.96	9.06

Table 5.6: Maximum message-passing times for test problems.

Conclusions

The factors relating to the possibility of a bottleneck on the master processor are:

the ease of solution of the LP relaxations;

the time taken to read the long node information from the disk; and

the message-passing time.

Although nothing can be done to alter the ease of solution of LP relaxations without drastically altering the code, the test results indicate that large MIP problems (for which the algorithm was designed) suffer less from bottlenecking caused by this factor than other problems.

The latter two factors, however, were dealt with so as to improve the performance of the algorithm.

The time taken to read the long node information for an LP relaxation disk depends on the size of the problem and the number of integer entities it contains. The only way that the disk-reading process can be speeded up is to alter the data structures. Some of the long node information was thus held on disk in a packed form, to enable a quicker disk-reading operation.

The message-passing times, which also depend upon the size of the problem being solved and the number of integer entities it contains, were improved by passing some of the LP relaxation data to slaves in packed form. The slaves unpacked the data, acted upon it and returned some of the LP solution information in packed form, thus reducing the overall message-passing overheads.

Once the changes to the code had been implemented, the test problems were attacked again to see what improvement in performance had been generated. The computational results of this second series of tests are given in Appendix 2B.

As can be seen from Figs. 5.12 to 5.17, most of the speedups for the test problems are at least as good as before, with small MIP problem DAAC actually now achieving a superlinear speedup using two slaves, and medium MIP problems BAG882 and TAX1 achieving better speedups for longer.

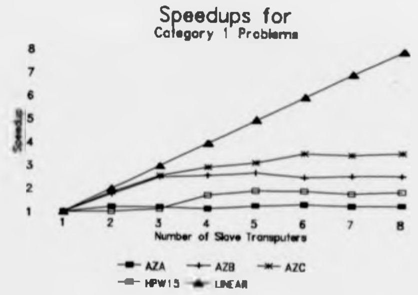


Fig. 5.12: Speedups for Small Combinatorial Problems.

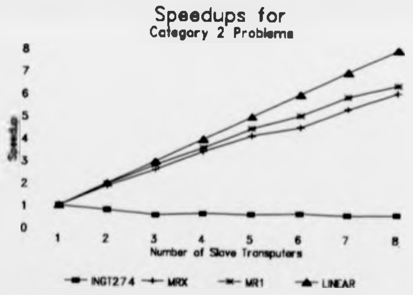


Fig. 5.13: Speedups for Medium Combinatorial Problems.

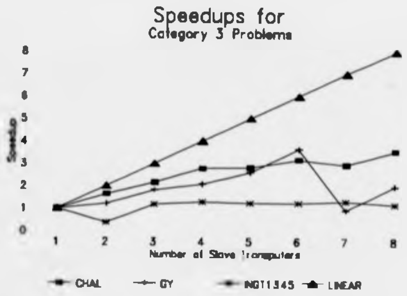


Fig. 5.14: Speedups for Large Combinatorial Problems.

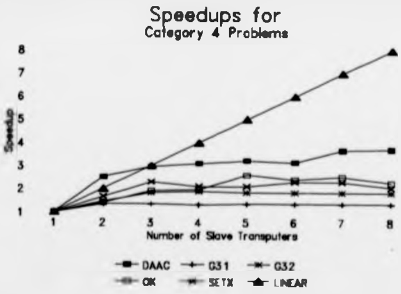


Fig. 5.15: Speedups for Small MIP Problems.

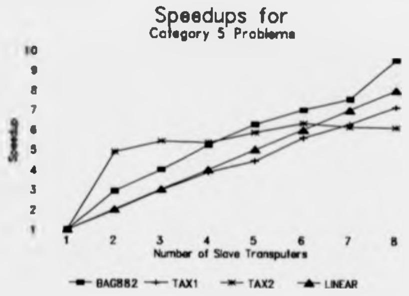


Fig. 5.16: Speedups for Medium MIP Problems.

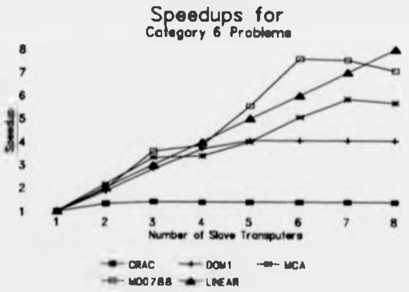


Fig. 5.17: Speedups for Large MIP Problems.

Tests were also carried out to determine how large an overhead was being amassed by sending messages one word at a time. Appendix 2C contains graphs comparing the solution times achieved for each problem on the first and second test runs with those achieved by using the packed data structures mentioned above, but by sending messages of different lengths.

Although the times for the single-slave tests will obviously be improved when the packed data structures are used (since the same search is being carried out in a slightly shorter time because the disk-reading operation is quicker), it is interesting to note that in all three test runs using the code with packed data structures, the algorithm generally performs much better than the original code, no matter how many slaves are used.

It is also worth noticing that the effects of passing messages in different ways so as to accumulate different amounts of overhead are hardly noticeable (once the packed data structures have been implemented), except in the case of the smaller problems where there are some improvements.

Since the solution times of the larger problems (for which the code was designed) show little change when different length messages are passed, it was decided to keep the messages at one word in length, as this requires only a simple FORTRAN code.

It must, of course, be taken into account that due to the non-determinacy of the algorithm (when more than one slave is used and asynchronous coordination is employed), the algorithm may well have carried out different searches for the four runs.

6. Comparison of Different Node Selection Strategies

Introduction

The computational results given in Chapter Five show that it is possible to increase performance when using more than one slave transputer to attack certain categories of problem. These favourable results are partly due to the structure of the problems attacked, with problems performing well if they achieve a good balance of computation to message-passing. It can also be seen that the test problems which achieved the best increases in performance also searched a similar, or lesser number of nodes when more than one slave was used. Thus, the node selection strategy used by the algorithm responded well to the presence of more than one slave transputer when used to attack the test problems.

Unlike problem structure, the node selection strategy used by the parallel algorithm is very much under the control of the algorithm designer. A comparison of different node selection strategies was thus carried out to determine if particular categories of problem react well to certain strategies.

6.1. Traditional Node Selection Criteria

(Nemhauser and Wolsey, page 359) suggest possible criteria for choosing which active node to branch on:

Node Selection Method One

Choose the node that is most likely to lead to an optimal IP solution. Once an optimal IP solution has been found, even if it cannot immediately be proved to be optimal, the best possible value of the cutoff has been found. This can have a marked effect on subsequent fathoming of nodes. The **best estimate** rule provides appropriate node selection criteria for this method. As mentioned in Section 2.4.1, nodes are created by deciding which unsatisfied variable to branch upon. The degradations to the LP relaxation solution caused by branching up or down on each variable x_j is calculated and stored as D_j^+ and D_j^- respectively (using the penalty calculations described in Appendix 1G). These values can be used to estimate the degradation to the IP solution that will result if a node is chosen for branching.

As an example, for maximisation problems, the degradation to the present incumbent IP solution that will occur if a node is chosen such that variable x_j is branched upon can be estimated as follows (for the case where $D_j^+ \geq D_j^-$):

If the variable x_j has a new upper bound imposed upon it, the estimate of the new IP solution is

$$\hat{z} = z_{LP} - D_j^- - \sum_{(k \in N)} MIN(D_k^-, D_k^+)$$

where z_{LP} is the current LP solution, N is the set of unsatisfied global entities and $j \neq k$.

If the branch is made the other way, such that variable x_j has a new lower bound imposed upon it,

$$\hat{z} = z_{LP} - D_j^+ - \sum_{(k \in N)} MIN(D_k^-, D_k^+)$$

The best estimate rule chooses the node which appears to degrade the IP solution the least.

<u>Node Selection Method Two</u>

Try to find a node that will quickly lead to a feasible solution to the IP. The quick improvement method of Forrest, Hirst and Tomlin provides appropriate criteria for this method ([Forrest et al., 1974]).

The node chosen is that which gives the maximum value (for maximisation problems) to

$$\frac{z_{LP} - z_{IP}}{z_{LP} - \hat{z}}$$

where z_{LP} is the current LP objective function value, $\underline{z}_{IP}$ is the present cutoff value, and $\hat{z}$ is the estimate of the new IP solution.

Note that in order to maximise the above fraction, nodes where $\hat{z} > \underline{z}_{IP}$ will be preferred to nodes where $\hat{z} \leq \underline{z}_{IP}$. Preference will also be given to nodes where $z_{LP} - \hat{z}$ is small.

6.2. The Node Selection Strategies Used for Experiments

Four different node selection strategies were devised, based on the two node selection methods mentioned above. These node selection strategies were then used to attack the set of test problems, and the results compared with those obtained by using the node selection strategy discussed in previous chapters (which is recapped below).

Strategy One involves making a choice from all candidate nodes each time a new LP relaxation is required. The criterion used for comparison of the nodes is provided by the best estimate method. The process of comparing all candidate nodes each time, using the best estimate criterion should produce good IP solutions, although they will not be produced very quickly. Once solutions have been found however, they will tend to provide good cutoff values, so that many of the remaining nodes on the search tree can be fathomed quickly.

Strategy Two again involves a choice from all candidate nodes. In this case however, the best estimate criterion is only adhered to until an integer feasible solution has been found. Thereafter, the Forrest-Hirst-Tomlin criterion is used. The efficiency and speed of this method will depend on the quality of the initial IP solution found using the best estimate criterion. If the initial IP solution is good, the remaining search should be quick, since the Forrest-Hirst-Tomlin criterion used thereafter usually chooses nodes close to the incumbent. If the initial IP solution found is far from the optimum however, the remaining search may take a long time, as nodes will be generated near the incumbent instead of in more promising areas.

Strategy Three involves a parallel version of the depth first search. The most promising son (if any exist) of the last node solved will be chosen next unless it seems more profitable to branch again on the parent. If both sons of a node have been generated and there are free slave transputers so that another node can be branched upon, a choice is made from all the remaining eligible candidate nodes. The criterion for choosing from all candidate nodes and for comparing parent and son is the same as in strategy two, in that the best estimate criterion is used until a feasible IP solution has been found, after which the Forrest-Hirst-Tomlin criterion is used. (Nemhauser and Wolsey, page 358) state that one of the principle advantages of a depth first search is that feasible IP solutions are more likely to be found deep in the tree than in nodes near the root. The initial aim of this node selection strategy is to make use of both the depth first criterion and the best estimate criterion during the search for an initial IP solution. It is hoped that the combination of the two criteria should lead to a good IP solution quickly. Once the initial IP solution has been found, the Forrest-Hirst-Tomlin criterion is used to search the surrounding area. If the initial solution was good (as we hoped) then the rest of the nodes on the tree can be fathomed quickly.

Strategy Four involves a similar process to strategy three, except that any comparisons of or choices from the candidate nodes are based only on the best estimate criterion. This strategy is not so dependent on getting a good initial IP solution, since the best estimate criterion do not tend to choose nodes close to the incumbent as often as the Forrest-Hirst-Tomlin criterion.

The **Original Strategy** used in previous chapters for node selection is also similar to strategy three. A parallel version of the depth first search is carried out, whereby both immediate descendants of the last node solved are tackled simultaneously if two slaves are free. If only one slave is free, the more attractive descendant is attacked next and no special effort made to consider the other, less favourable branch. If both descendants of a node are fathomed, a search of all active candidate nodes is carried out, using the best estimate criterion for comparison if no integer feasible solution has been found, and the Forrest-Hirst-Tomlin criterion thereafter.

6.3. Computational Results

The four node selection strategies were tested on the set of IP problems used in Chapter Five. Figs. 6.1 to 6.22 below show the resulting solution times achieved, and a comparison is made with the results obtained using the original node selection strategy.

Category One - Small Combinatorial Problems

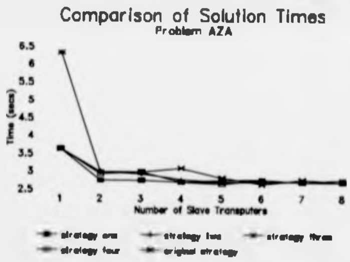


Fig. 6.1: Solution Times for Problem AZA.

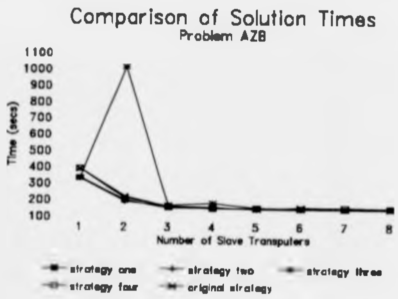


Fig. 6.2: Solution Times for Problem AZB.

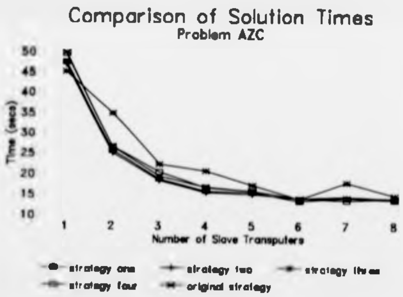


Fig. 6.3: Solution Times for Problem AZC.

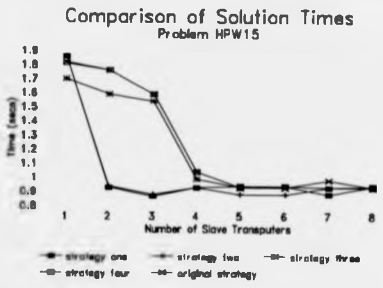


Fig. 6.4: Solution Times for Problem HPW15.

Category Two - Medium Combinatorial Problems

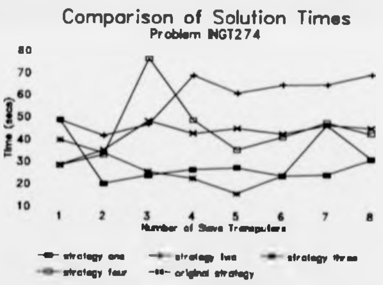


Fig. 6.5: Solution Times for Problem INGT274.

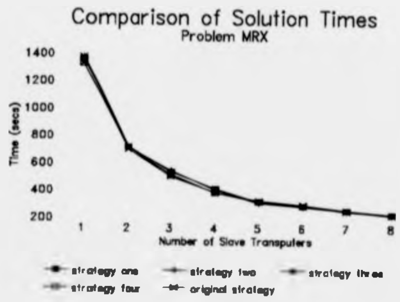


Fig. 6.6: Solution Times for Problem MRX.

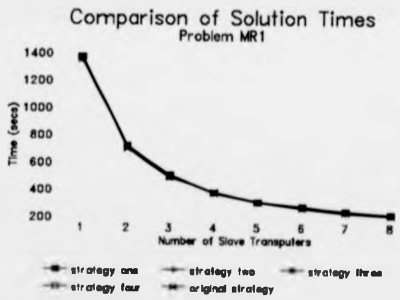


Fig. 6.7: Solution Times for Problem MR1.

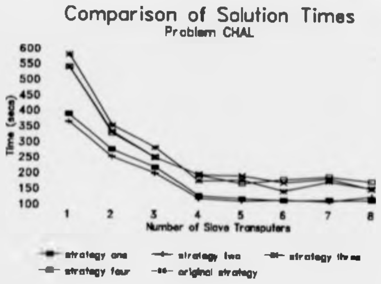


Fig. 6.8: Solution Times for Problem CHAL.

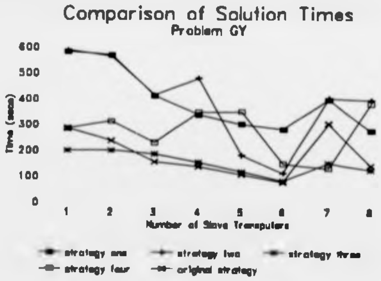


Fig. 6.9: Solution Times for Problem GY.

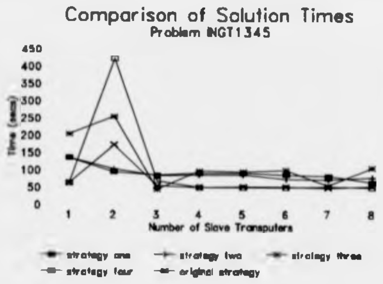


Fig. 6.10: Solution Times for Problem INGT1345.

Category Four - Small MIP Problems

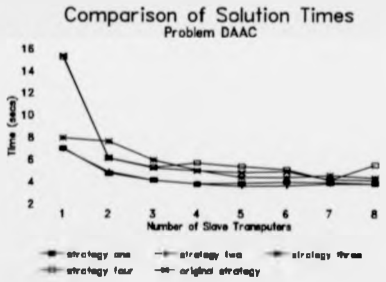


Fig. 6.11: Solution Times for Problem DAAC.

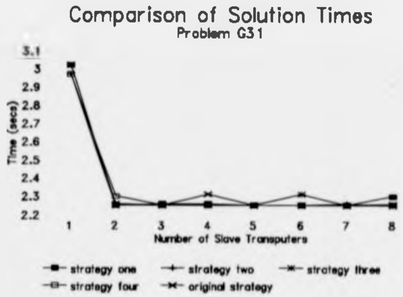


Fig. 6.12: Solution Times for Problem G31.

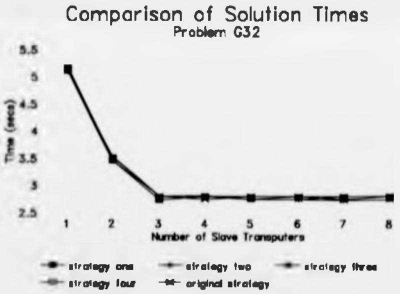


Fig. 6.13: Solution Times for Problem G32.

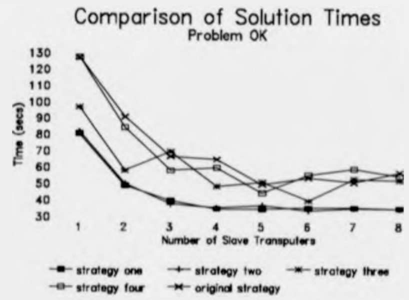


Fig. 6.14: Solution Times for Problem OK.

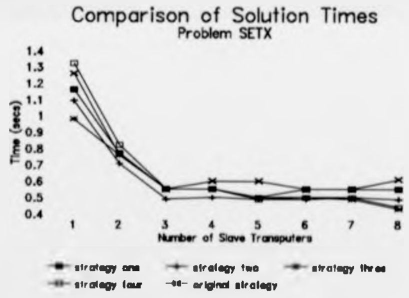


Fig. 6.15: Solution Times for Problem SETX.

Category Five - Medium MIP Problems

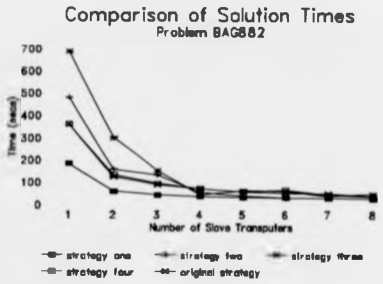


Fig. 6.16: Solution Times for Problem BAG882.

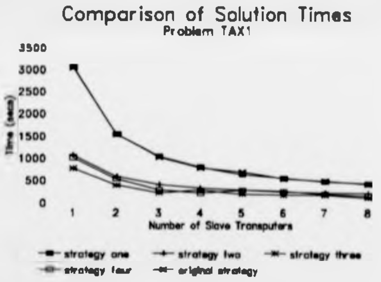


Fig. 6.17: Solution Times for Problem TAX1.

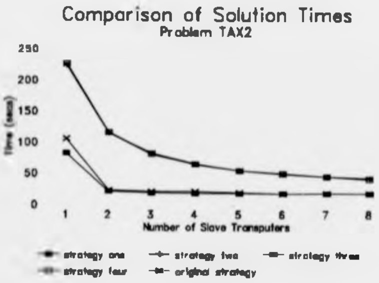


Fig. 6.18: Solution Times for Problem TAX2.

Category Six - Large MIP Problems

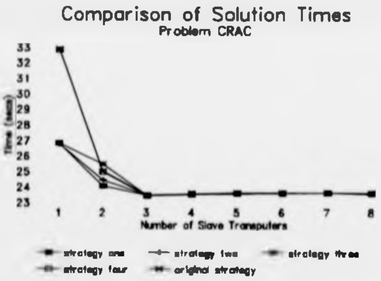


Fig. 6.19: Solution Times for Problem CRAC.

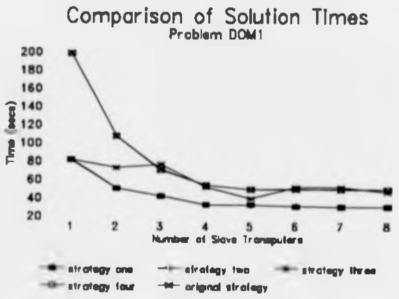


Fig. 6.20: Solution Times for Problem DOM1.

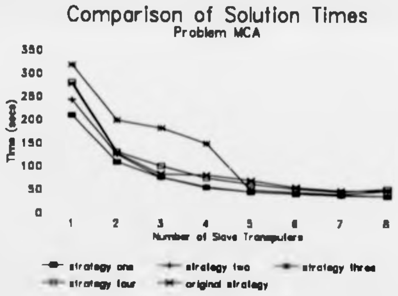


Fig. 6.21: Solution Times for Problem MCA.

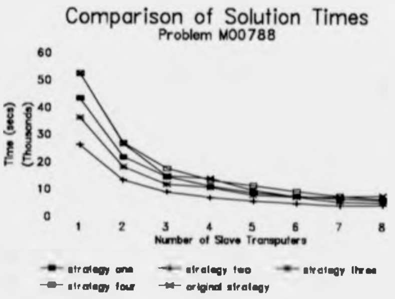


Fig. 6.22: Solution Times for Problem MO0788.

<u>Discussion of Results</u>

We shall now discuss the results obtained from attacking the different categories of problem using the various node selection strategies. Appendix 3 lists the computational results in full.

<u>Problem Category 1: Small Combinatorial Problems</u>

As can be seen from Figs. 6.1 to 6.4, the solution times of the problems from this category are quite similar, no matter which of the node selection strategies is used. This is because the average usage of slave transputers is still fairly low (as can be seen in Table 6.1 below).

Note that the point after which no more transputers are used, no matter how many are available, is referred to as Saturation Point.

Problem	Maximum No. of Slave Transputers Used					Average Transputer Usage after Saturation Point				
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
AZA	6	6	6	6	6	3.0	3.0	3.0	3.0	2.6
AZB	7	7	6	6	6	2.6	2.6	2.7	2.6	2.6
AZC	8	8	8	8	8	3.9	4.0	4.0	4.1	4.0
HPW15	4	4	5	5	5	2.1	2.1	2.4	2.4	2.4

Table 6.1: Transputer Usage for Small Combinatorial Problems.

When attacking problems AZA and AZB (which contain only binary variables as their discrete component) using node selection strategies one or two, only one solution node was found during any of the runs. Problem AZC (which contains only semi-continuous variables as its discrete component), performs in an identical manner. In the case of problem HPW15 (whose discrete component is made up of general integers with upper bounds of five, ten and twelve), the single slave runs using strategies one or two find two solution nodes during the search, whereas the multi-slave runs find only one solution node.

When using strategies three to five to attack any of the test problems in this category, the number of solution nodes vary (from one to twenty six in the case of AZB) depending on the number of slaves used.

This indicates that, for these problems at least, strategies one and two are performing as intended by choosing nodes that lead to the optimal solution.

Problem Category 2: Medium Combinatorial Problems

Figs 6.5 to 6.7 show again a difference between the test problems that contain only binary variables and those which contain general integers.

Problems MRX and MR1 (which contain only binary variables) give almost identical solution times, no matter which node selection strategy is used. The average transputer usage is very high for these problems, and the vast majority of nodes that are generated are attacked twice, indicating that the LP relaxations are very easy to solve.

Problem INGT274, which contains only general integers with upper bounds of nine, performs differently with each node selection strategy, although on average only one and a half slave transputers are used in each case. Strategy one seems to be the best for this problem, closely followed by strategy three. The number of nodes generated using either of these strategies is much lower than the number generated using the other strategies.

It is imagined that strategy two resulted in a bad first solution, since it performs the worst overall. The number of nodes added to the candidate list but never attacked is very large in comparison with the number when strategy one is used. This indicates how dependent strategy two can be on finding a good solution quickly.

Problem Category 3: Large Combinatorial Problems

The solution times shown in Fig. 6.8 to 6.10 for problems CHAL, GY and INGT1345 are again varied.

Problem CHAL (which contains only binary variables as its discrete component) responds well to strategies one and two, which tend to find fewer (and better) solution nodes. The other strategies generate and attack many more candidate nodes and thus take longer to solve the problem. The average transputer usage is quite high in all cases however, so the other strategies still solve the problems fairly quickly.

Problem GY (which contains binary variables and general integers with upper bounds of two, three, four and five) displays the opposite results, with strategies one and two performing worse than the others. This is because, although strategies one and two usually produce fewer solution nodes, they take a long time to do so. On the other hand, the other strategies quickly find feasible IP solutions (as many as eighteen in the case of strategy four), one of which is optimal. The average transputer usage is also a little higher for strategies three, four and five.

The solution times for problem INGT1345 (which contains general integers with upper bounds of nine) are quite similar when three or more slaves are used. This is because the average number of slaves used does not rise much above two. Strategies four and five perform the best overall. They seem to find good solutions fairly late in the search (as indicated by a large number of nodes being removed from the candidate list when a solution has been found). This suggests that the optimal solution is deep in the tree in this problem.

Problem Category 4: Small MIP Problems

The small MIP problems shown in Figs. 6.11 to 6.15 on the whole show similar results regardless of the node selection strategy used. This is because of the relatively low transputer usage on average (see Table 6.2 below).

Problem	Maximum No. of Slave Transputers Used					Average Transputer Usage after Saturation Point				
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
DAAC	6	6	7	7	7	3.0	3.0	2.7	2.8	2.8
G31	2	2	2	2	2	1.3	1.3	1.3	1.3	1.3
G32	3	3	3	3	3	1.8	1.8	1.8	1.8	1.8
OK	6	6	6	6	7	2.7	2.7	2.7	2.8	2.7
SETX	4	4	5	5	5	2.3	2.3	2.3	2.3	2.5

Table 6.2: Transputer Usage for Small MIP Problems.

Problems G31, G32 and SETX produce similar results no matter which strategy is used. This is because they are all solved very quickly indeed in all cases (e.g. in three, five and one and a half seconds at most respectively) due to the very small number of discrete variables in these problems. Problems G31 and G32 contain binary variables and general integers with upper bounds of one and seven. Problem SETX contains special ordered sets of type one.

Problem DAAC, which only contains binary variables as its discrete component, performs quite similarly in all cases, although strategies one and two give the best solution times. This is because they consistently find the optimal solution node fairly quickly . Thus, the number of nodes generated and attacked is fewer than when other strategies are used, and the solution times are reduced accordingly.

Problem OK, which also only contains binary variables as its discrete component, performs quite a lot better when strategies one and two are used. This is for the same reasons as given for problem DAAC. The only difference is the number of poor solutions generated by using strategies three, four and five. In the case of problem OK, more poor solutions are generated, and thus the search takes longer.

Problem Category 5: Medium MIP Problems

Figs. 6.16 to 6.18 show the solution times achieved for problems BAG882, TAX1 and TAX2, all of which contain only binary variables as their discrete components.

Problem BAG882 is solved in similar times when four or more slaves are used. When one to three slaves are used however, strategy one performs much better than the others. Strategy two performs quite poorly when one to three slaves are used, indicating that a relatively poor first solution is found. Strategies three, four and five find very large numbers of solution nodes when one to three slaves are used, and hence tend to waste time considering lots of useless nodes.

Problems TAX1 and TAX2 seem to show opposite results. TAX1 responds well to strategies two, three and four and poorly to strategies one and five, whereas TAX2 responds in exactly the opposite way. The key to these results is the number of infeasible nodes generated. In the case of TAX1, strategies one and five only tend to find one or two solution nodes, but generate many infeasible nodes whilst doing so. In the case of TAX2, strategies one and five again only tend to generate one or two solution nodes, but generate very few infeasible nodes in the process. Strategies two, three and four tend to find more solution nodes and generate a fairly large (but consistent) number of infeasible nodes whilst doing so.

Problem Category 6: Large MIP Problems

As can be seen in Figs. 6.19 to 6.22, different types of large MIP problem react differently to the node selection strategies used.

Problem CRAC, which contains binary variables, produces very similar solution times, no matter which strategy is used. This is probably due to the low average usage of transputers (which is never much above three).

Problem DOM1, which contains special ordered sets of type one and two, performs well when strategies one and two are used. The other strategies do not fare much worse when four or more slaves are used, although strategies four and five perform quite poorly when one and two slaves are used. The key again seems to be that strategies one and two find relatively few solution nodes and thus waste less time exploring useless nodes. The times are not too varied however, since the usage of transputers is slightly higher for strategies three, four and five.

Problem MCA, (which contains special ordered sets of type one and two, as well as binary variables and general integers with upper bounds of two), produces fairly similar results for all strategies except strategy three, which performs relatively poorly. Strategy one performs the best overall, as it tends to find the optimal solution quickly. The transputer usage is high in all cases, although slightly higher in the cases where more nodes are searched, so the time differences are not large.

Problem MO0788, which contains general integers with upper bounds of two, five and ten, performs similarly with four or more transputers, although strategy two does seem to be the most successful overall. This is because it finds the optimal solution after a relatively small number of solution nodes. Strategy one performs similarly, but takes longer to find the optimal solution. Strategies three, four and five find a large number of IP solution nodes and are either lucky or unlucky when it comes to finding the optimal solution. Strategy three tends to find the optimal solution quickly enough to produce a lower solution time than strategy one.

Conclusions

Although it is difficult to draw conclusions from testing the different node selection strategies using such a small group of problems, the following opinions were formed.

Strategy One finds good solutions and thus produces strong cutoff values which are very useful in the subsequent fathoming of candidate nodes.

Strategy Two shares many of the positive attributes of strategy one, but can be very inefficient if a good solution is not found fairly quickly. If a good solution is found however, the Forrest-Hirst-Tomlin criteria can help to finish the search quickly.

Strategy Three tends to perform relatively badly, especially if a fairly good solution is not found quite quickly. If a good solution is not found quickly, more nodes are invariably searched than by other strategies, although if the transputer usage is high enough, this doesn't slow down the search too much.

Strategy Four performs relatively well if enough of the processors can be kept busy. Although a large number of nodes are almost always searched using this strategy, in some circumstances the optimal solution can be found quickly by luck.

Strategy Five is very similar to Strategy Three, and tends to perform in a very similar way.

Strategy one gives the best overall performance on the test problems available, and copes well with the large MIP problems. In the cases where strategy one performs relatively poorly, the results are not much worse than those of better strategies, especially when large numbers of transputers are used. In most circumstances however, using strategy one leads to one of the best solution times achieved for the problem, if not the best.

It is therefore deemed worthwhile to use strategy one, (whereby all candidate nodes are considered for branching on each occasion, and comparisons made using the best estimate criteria), as the default node selection strategy for the parallel algorithm.

7. Parallel Branch and Bound Strategies

The previous chapter demonstrated that variations in the problem solution times may result when different node selection strategies are used by the parallel Branch and Bound algorithm. Node Selection Strategy One (i.e. comparing all nodes each time, using the best estimate criteria), was chosen as the default strategy for future use, as it produced good problem solution times for most categories of problem on which it was tested, including the large MIP problems. Although this default strategy should allow most problems to be solved quickly, it is possible to improve on its performance in certain circumstances. The structures of certain types of problem may be more efficiently exploited by further parallelising part of the Branch and Bound algorithm.

7.1. Further Parallelisation of the Algorithm

The solution (within a reasonable time) of many real large MIP problems has been shown to depend upon successful exploitation of the modeller's knowledge of the problem structure. This knowledge is especially useful when deciding which variables to branch upon, and, when the branching priorities of the variables have been decided, which branches to make.

When deciding on the branching priorities of variables, for instance, those variables whose values indicate whether or not certain actions should be taken should obviously be branched upon before any variables whose values reflect the effects of the actions.

When a variable has been chosen, the branches made upon it and the order in which they are made also have an effect on the overall solution time. Much time can be wasted by exploring unproductive areas of the search tree if a bad choice of branch has been made. The process of choosing the number and order of branches to be made may be adapted to make better use of the parallel processing facilities available.

As mentioned in Section 2.4.1, once an integer-constrained variable has been chosen to be branched upon, a **binary** branch is usually made. For example, if a general integer variable x, say, which has an upper bound of five and a lower bound of zero, takes the value 2.4 in a feasible LP solution, then two more LP problems are generated for later exploration. The first such problem will contain an extra constraint of the form $x \le 2$ and the second will contain an extra constraint of the form $x \ge 3$ (see Fig 7.1 below).

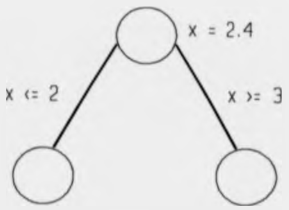


Fig. 7.1: An Example of Binary Branching.

The strength of the parallel algorithm discussed in the previous chapters is that it attacks several such subproblems in parallel, thus gaining much more information in the same period of time.

When attacking certain types of problem, there is the possibility of enhancing this effect so that even better use is made use of the processing power available. In order to perform this enhancement, **multiway branching** is performed on the chosen variable. For example, instead of the two branches made on the variable x in the previous example, several branches (of the form shown in Fig 7.2 below) could have been made.

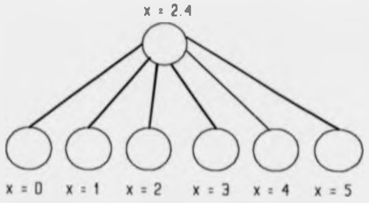


Fig. 7.2: An Example of Multiway Branching.

The binary branching strategy used by the serial Branch and Bound algorithm (and hence implemented as part of the initial parallel algorithm), was first implemented when processing power was not great enough to consider more than two branches in a reasonable time. There is nothing however, in the basic Branch and Bound algorithm to prohibit the making of more than two branches on an appropriate variable. The power of the Branch and Bound algorithm may thus be increased in certain circumstances, by making more than two branches and considering them in parallel.

One way in which the multiway branching concept has the potential to enhance the overall performance of the algorithm is by keeping the slaves busier at the beginning of the search. For example, when the first few nodes of an MIP problem are considered by the algorithm, there is usually not enough work to keep all the slaves busy. It is not until a later stage of the search, when the list of candidate nodes is longer, that all slaves can be kept busy (assuming that nodes can be chosen from the list and farmed out quickly enough). If more branches are made on chosen variables, the candidate list will grow to an appropriate size much more quickly. Since the LP relaxations at the start of the search should be the hardest to solve (if sensible priorities have been chosen), the multiway branching strategy has the potential to lead to quick solutions. There is, of course, the potential drawback that attacking the problems created by the additional branches may not provide any useful information, and might only be creating work for idle hands.

Another way in which the multiway branching strategy may potentially be used to enhance performance is by providing stronger branches on certain types of variable. When branching on an unsatisfied variable that can still take many different integer values, branches can be made based on predictions of the likely final value of the variable (i.e. in the optimal solution). If the variable takes a value which is thought to be close to its final value, branches can first be made that fix the variable to the integer values on either side of its present value. Subsequent branches could then be made, in order to completely fathom the node.

Of course, multiway branching can only be usefully applied to problems formulated using general integer variables or Special Ordered Sets (which will be defined in section 7.2), as binary or semi-continuous variables can only be branched upon at most twice. Tests were carried out on problems formulated using general integer variables and special ordered sets, in order to see if it is worth adding multiway branching to the default branching strategy of the algorithm for such problems.

7.2. Multiway Branching on General Integers

When an unsatisfied general integer variable has been chosen (by using the variable selection techniques discussed in section 2.4.1), the number and order of the branches to be made must be decided.

The branching strategy of the serial Branch and Bound algorithm is as follows. Two branches are made on the chosen general integer variable before the node is fathomed. In order to determine the preferred branching direction, (i.e. to decide which branch to perform first), estimates are made of the degradation to the LP solution that will be incurred if the branch is made. The default strategy is to branch first in the direction which produces the minimum degradation to the LP solution. This is because the node created by implementing this branch might later be compared with other nodes during the node selection process. Under the default node selection strategy decided upon, the node which degrades the IP solution the least will be chosen. As mentioned in the previous chapter, the degradation to the IP solution incurred by choosing a node is estimated as

$$z_{LP} - MIN(D_j^+, D_j^-) - \sum_{(k \in N)} MIN(D_k^-, D_k^+)$$

where D_j^+ and D_j^- are the degradations to the LP solution incurred by making the branches up or down to the nearest integer from the chosen variable x_j, z_{LP} is the LP solution, N is the set of unsatisfied global entities, and $j \neq k$.

As mentioned in section 2.4.1., the degradations D_j^+ and D_j^- caused by branching up and down to the nearest integer are calculated as follows:

$D_j^- = p_j f_j$

$D_j^+ = p_j^+(1 - f_j)$

where p_j^- and p_j^+ are unit penalties incurred by branching down and up respectively, and f_j is the fractional part of the value of the unsatisfied variable x_j (as seen in Fig. 7.3 below).

The per unit penalties for variable x_j are calculated using the method described in Appendix 1G.

Note that the branches imposed are inequalities (as can be seen in Fig. 7.3).

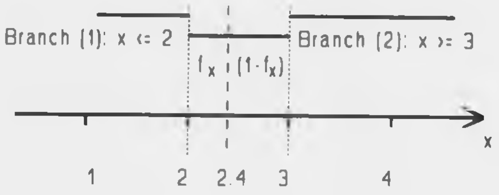


Fig. 7.3: Binary branching on variable x.

The first step when extending the above techniques to multiway branching is to calculate the estimated degradations to the LP solution caused by fixing the value of the unsatisfied variable to the nearest integer values above and below. These degradations are calculated using the formulae for D_j^+ and D_j^- shown above, although the corresponding branches imposed this time are equalities (see Fig. 7.4 below). If circumstances warrant it, as they do in this example, further branches are then made on either side, to ensure the proper fathoming of the node.

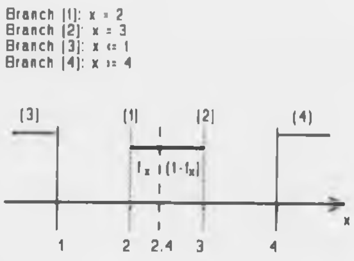


Fig. 7.4: Multiway branching on variable x.

The method previously used to estimate degradations to the LP solution is extended to provide estimates for the outer two branches as follows:

D_j^{--}, the estimated degradation for branch (3) is calculated as

$p_j^- (1 + f_j)$

and D_j^{++}, the estimated degradation for branch (4) is calculated as

$p_j^+ (2 - f_j)$

where p_j^- and p_j^+ are unit penalties incurred by branching down and up respectively (calculated as in Appendix 1G), and f_j is the fractional part of the value of the unsatisfied variable x_j (as seen in Fig. 7.4 above).

If more than two branches are to be made, the number of further branches depends upon the circumstances.

7.2.1. Three-Way Branching on General Integers

If the general integer variable can only take three integer values (as in Fig. 7.5), or if the branches already made are at one end of the spread of possible integer values (as in Figs. 7.6 and 7.7), then only one more branch should be made.

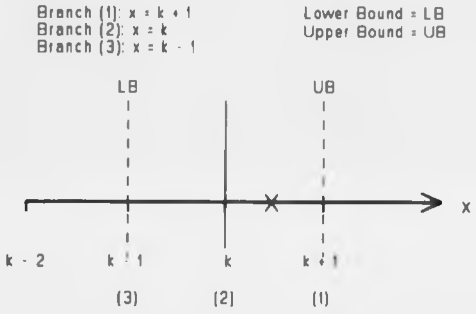


Fig. 7.5: Situation where only three integer values are possible.

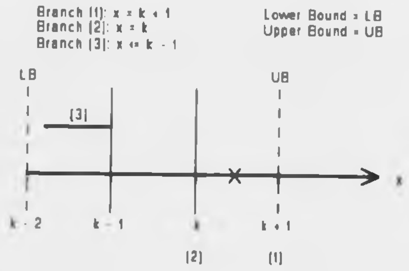


Fig. 7.6: When an initial branch is at the upper bound.

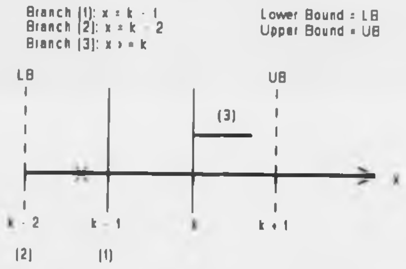


Fig. 7.7: When an initial branch is at the lower bound.

The three branches will be applied in ascending order of estimated degradation to the LP solution.

7.2.2. Four-Way Branching on General Integers

In other circumstances, a total of four branches can be made (as in Fig. 7.8 below). Again, these branches will be applied in ascending order of the estimated degradation to the LP solution.

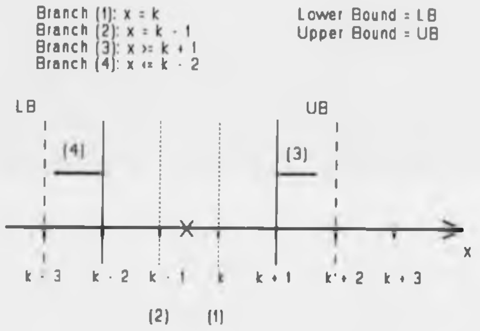


Fig. 7.8: Situation where four branches are possible.

7.2.3. Computational Results for Multiway Branching on General Integers

Making the first two branches is simply a matter of setting the appropriate variable to an integer value. It is hoped that it will thus be possible to quickly determine that the new LP is infeasible or to solve the new LP more quickly than its parent.

In order to test the usefulness of multiway branching techniques on general integers, the test problems detailed in Table 7.1 were attacked. The problems were attacked using multiway branching in conjunction with the four different node selection strategies introduced in the previous chapter. This should determine whether multiway branching on general integer variables responds well to any of these strategies.

Problem	Category	NROW	NCOL	NGLENT	Description
GY	3	913	888	528	240GI2,96GI3,48GI4,48GI5, 96BV
G31	4	159	146	9	5GI1,1GI7,3BV
G32	4	162	148	9	5GI1,1GI7,3BV
HPW15	1	56	45	30	10GI5,10GI10,10GI12
INGT274	2	13	274	274	274GI9
INGT1345	3	19	1345	1345	1345GI9
MO0788	6	1123	926	24	3GI2,18GI5,3GI10

Examples of the notation used above are:

5GI3 =	5 general integer variables with lower bounds of 0 and upper bounds of 3.
3BV =	3 binary variables.

Problem Categories: Small Combinatorial (1); Medium Combinatorial (2); Large Combinatorial (3); Small MIP (4); Large MIP (6).

Table 7.1: Test problems for multiway branching on general integers.

The computational results were as shown in Figs. 7.9 to 7.15 below.

KEY

Binary Branching:	problem attacked using the conventional binary branching approach and the default node selection strategy decided upon in the last chapter (i.e. node selection strategy one as described in section 6.2 on page 188).
Multi-way Branch X:	problem attacked using the multiway branching approach and node selection strategy X (as described in section 6.2 on pages 188-190)

Category 1: Small Combinatorial Problems

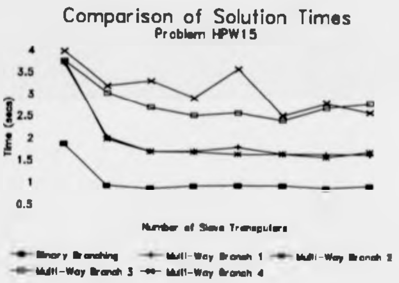


Fig. 7.9: Multiway branching on HPW15.

Category 2: Medium Combinatorial Problems

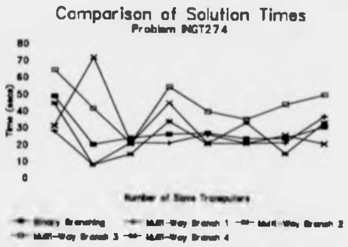


Fig. 7.10: Multiway branching on INGT274.

Category 3: Large Combinatorial Problems

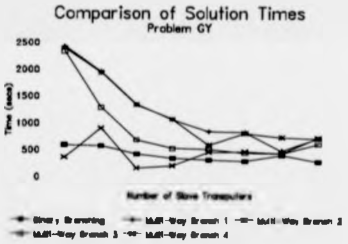


Fig. 7.11: Multiway branching on GY.

Category 3: Large Combinatorial Problems

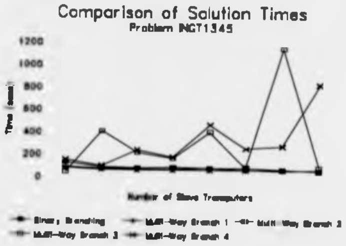


Fig. 7.12: Multiway branching on INGT1345.

Category 4: Small MIP Problems

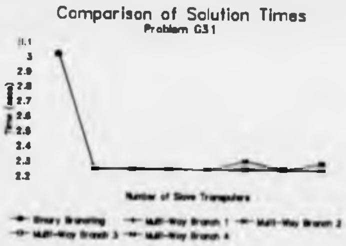


Fig. 7.13: Multiway branching on G31.

Category 4: Small MIP Problems

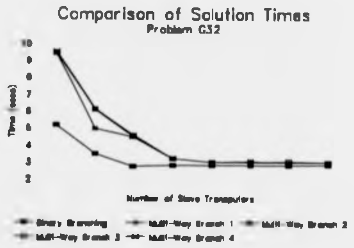


Fig. 7.14: Multiway branching on G32.

Category 6: Large MIP Problems

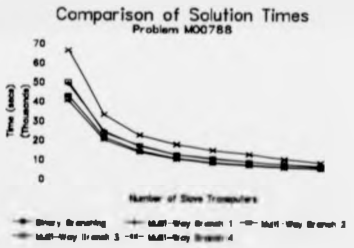


Fig. 7.15: Multiway branching on MO0788.

Discussion of Results

Appendix 4A lists the computational results in full. As can be seen from Figs. 7.9 to 7.15, multiway branching does not increase the speed of solution of the problems in the test set. In each case, the binary branching strategy under node selection strategy one (i.e. the default approach) tends to solve the problems more quickly. Although the slave transputers are more heavily utilised under the multiway branching scheme, the extra work does not provide much more useful information. More nodes are actually generated and attacked during the search, but the vast majority of these extra nodes provide results that are infeasible or worse than the cutoff. That is, the multiway branching scheme seems to be making work for idle hands. Only part of the increase in transputer utilisation is due to the increased number of nodes attacked. When an LP relaxation has been solved, if a feasible, but not integer-feasible solution has been found, data must be produced for up to four branches, as opposed to two branches under the binary branching scheme. Thus, the increase in transputer usage is probably due to more LP relaxations being solved and to certain nodes taking slightly longer to finish the work at the slave. So, under the present circumstances, multiway branching on general integer variables does not appear to be a good way of exploiting the available parallel processing facilities.

7.3. Multiway Branching on Special Ordered Sets

More promising multiway branching possibilities occur for Special Ordered Sets ([Beale and Tomlin, 1970]).

A set of variables is said to form a Special Ordered Set of Type 1 (or an **S1 set**), if at most one of the set members can take a value that is greater than zero. In order for the S1 set to be of use as a modelling structure, the set members must be such that they can be meaningfully ordered within the set. The ordering of the set members is provided by the **Reference Row** of the S1 set, which is of the form

$$x = \sum_{i=1}^{n} X_i \delta_i$$

where $\delta_1, \delta_2, \ldots, \delta_N$ are the members of the S1 set, and X_1, $X_2, \ldots,$ X_N are items of data that correspond to the set members.

The N set members are ordered so that $X_1 < X_2 < \ldots < X_N$. If the X_i data items provide a meaningless or arbitrary order, the S1 set cannot be exploited efficiently by the solution algorithm. If however, the X_i data items provide a meaningful order, better branches can often be made on the set than would otherwise be the case.

S1 sets are often used in the modelling of a set of mutually exclusive actions, one of which must be made. For instance, the variables $\delta_1, \delta_2, \ldots, \delta_N$ can be used to reflect N mutually exclusive decisions. The N δ variables are specified as members of an S1 set, and some output (e.g. a resulting cost or profit) is associated with each of the decisions. These outputs, the X_i data items, can provide the ordering of the set members.

The N different δ variables can be used in the objective function with accompanying costs or profits, so that the output may be optimised.

To properly model the mutual exclusivity of the N decisions, it is also necessary to include a constraint of the form

$$\sum_{j=1}^{N} \delta_j = 1$$

and to state that $\delta_j \geq 0$ for $j=1,2,\ldots,N$.

This latter constraint is known as the **Convexity Row** for the S1 set. Although it is not necessary to include the convexity row in order to use a S1 set, it is essential to include it if the set members are meant to represent mutually exclusive decisions. The following discussion will assume that some form of convexity row is always present as part of an S1 set, although the more general form

$$\sum_{j=1}^{N} \delta_j + s_c = b_c$$

will be used, where s_c is the non-negative slack variable, and b_c the right hand side, of convexity row c.

S1 sets are commonly used within project evaluation models. If a single project must be chosen from a list of N candidates, the set of variables $\delta_1,\ldots,\delta_N$ (which represent choosing or not choosing the various projects), are specified as an S1 set. The definition of the S1 set ensures that at most one of the set members will take a non-zero value, whilst a convexity row of the first form mentioned can be used to ensure that this value will be unity. The expected revenues from the different projects can be used to order the variables within the set (i.e. to provide the X_i values for the reference row).

Another useful type of Special Ordered Set is the Special Ordered Set of Type 2 (or **S2 set**).

A set of variables is said to form an S2 set if at most two of the set members can take values greater than zero. If two of the variables do take values greater than zero, the variables must be adjacent set members. Again, the set members must be usefully ordered by means of a reference row if the S2 set is to be of any use. S2 sets are mainly used in the modelling of nonlinear functions, where again, a convexity row of the first form mentioned must be part of the model. It will be assumed that a convexity row of some form is always present for an S2 set.

The use of special ordered sets has made many intractable problems solvable by non-parallel Branch and Bound algorithms. This is because attacking a problem formulated using special ordered sets involves branching on **groups** of variables, and thus setting the values of several variables at once. A sensible branching strategy is of course necessary to ensure that the most promising subproblems are attacked.

7.3.1. Calculations of Estimated Degradations for Branching on Special Ordered Sets

As mentioned in section 2.4.1, when deciding on an unsatisfied entity to branch upon after an LP relaxation has been solved, the usual strategy is to choose the entity which causes the most degradation to the LP solution. The estimated degradations to the LP solution caused by branching on special ordered sets are calculated as follows (adapted from [Beale and Forrest, 1976]).

Let variables $\delta_1, \delta_2, \ldots, \delta_n$ form an S1 or S2 set, and have corresponding reference row values

$X_1 < X_2 < \ldots < X_n$

and values in the optimal LP relaxation of

$v_1, v_2, \ldots, v_n$

Calculate the average reference row entry

$$X^- = \sum_i X_i v_i / \sum_i v_i$$

and determine which variables have corresponding reference row values that fall on either side of it.

These variables are said to provide **straddle points**.

In Fig. 7.16 below, for instance, variables δ_3 and δ_4 are the members of a Special Ordered Set of Type 1 (S1 set) that provide the straddle points. (Note that the branches indicated are numbered arbitrarily here).

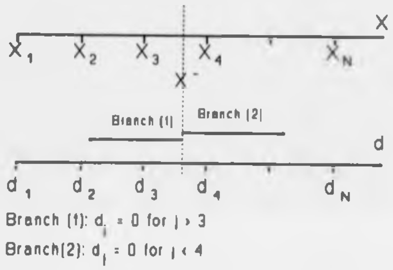


Fig. 7.16: Finding the average reference row value and the straddle points for an S1 set.

The average reference row value is assumed to provide a good guide as to which variable (for S1 sets) or variables (for S2 sets) should be allowed to take non-zero values. In the **best** possible solution to the problem, a single variable δ, with corresponding reference row value $X^{\cdot}$ (i.e. the average reference row value), would have to exist and take a non-zero value. Since the variable δ does not exist however, we attempt to make branches such that the variables that do exist (and are members of the set) take values that produce an identical average reference row value (and satisfy the set). The degradations to the LP solution that result from making such branches are used in the overall algorithm to determine whether the set is chosen as the unsatisfied global entity to branch upon next.

The task of estimating degradations caused by imposing the branches must be approached differently for S1 and S2 sets, as follows.

Estimated Degradation caused by branching on S1 sets

In the case of S1 sets, the effects of making a branch to the left straddle point are first determined. Since this is an S1 set, it is assumed that only the left straddle point variable will take a non-zero value once the branch has been made. Assuming that there is a convexity row for the set, of the form

$$\sum_{j=1}^{N} \delta_j + s_c = b_c$$

then the value of δ_L when the branch is made (i.e. v_L) can thus be calculated as $(b_c - s_c^{opt})$, where s_c^{opt} is the value of the slack variable in the optimal LP solution.

A vector can therefore be constructed containing, for each row i, the value of $v_L C_{iL}$ (where C_{iL} is the coefficient in row i of the left straddle point variable δ_L). This vector is said to contain the "corrected" contributions of the variables to each row of the problem. That is, these are the values that the rows will take after the branch has been made and the values of the set member variables are corrected so that they satisfy the set.

Once the "corrected vector" has been constructed, an "uncorrected vector" containing the present state of the rows of the problem must be constructed for comparison.

The "uncorrected vector" contains, for each row i of the problem,

$$\sum_j C_{ij} v_j$$

where C_{ij} is the coefficient of variable δ_j in row i.

The difference between the elements of the uncorrected and corrected vectors represents the changes that we assume will be made in each row once the branch is imposed. In effect, the difference between the elements represents a move from the present solution, where several variables take non-zero values, to a solution that satisfies the S1 set as an integer entity because only one variable takes a non-zero value.

The degradation to the LP solution incurred by changing each of the rows is then calculated. For each row, the contribution towards the total degradation is calculated by multiplying the distance moved by a per unit degradation figure for the row. The per unit degradation figure used for each row is calculated in a heuristic way (similar to that for general integer variables), as described in Appendix 1G.

The total estimated degradation to the LP solution that will be incurred if a branch is made to the left straddle point is finally calculated by summing the contributing degradations from each row.

The whole process is then repeated to determine the estimated degradation caused by branching to the right straddle point. The estimated degradation caused by branching on the set is assigned to be the minimum of the two. Although no decision is made as to where to branch until the set has been chosen as the entity to branch upon, Fig. 7.16 above shows the binary branches that are assumed as part of the process of calculating the estimated degradation for the set.

Estimated degradation caused by branching on S2 sets

The effects of branching on an S2 set must be estimated in a different way, since the branch to be made may lead to one **or** two variables taking non-zero values. If two variables do take non-zero values however, we expect the variables to be adjacent, a fact that we can use when preparing a corrected vector to show the effects of branching.

As mentioned previously, the **best** solution for the unsatisfied set would be achieved if there existed a single variable δ^{*}, which took a non-zero value and which had a corresponding reference row value X^{-}

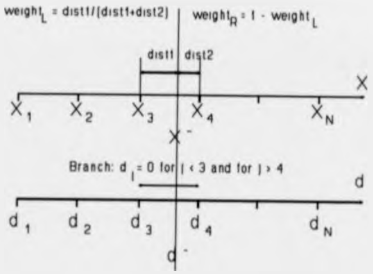
In order to create the corrected vector for branching on an S2 set, the effect of the artificial variable δ^- is imitated. So that the average reference row value for the variable will still be X^- (as shown in Fig. 7.17 below), the value, v^-, of δ^- in the LP solution, and the coefficients, C_{i-}, of δ^- in the different rows of the problem are interpolated from those of the straddle point variables.

The value v^-, of the artificial variable δ_i (which is assumed to be the only variable to take a non-zero value after the branch), is calculated as $(b_c - s_c^{opt})$.

For each row i, the coefficient C_{i-} of variable δ^- is calculated as

$$weight_L * C_{iL} + weight_R * C_{iR}$$

where $weight_L$ and $weight_R$ are measures of how near X^- is to X_L and X_R (see Fig. 7.17 below), and C_{iL} and C_{iR} are the coefficients of the left and right straddle point variables δ_L and δ_R in row i.



weight$_L$ = dist1/(dist1+dist2)
weight$_R$ = 1 - weight$_L$
dist1
dist2
X
X_1
X_2
X_3
X_4
X_N
X^-
Branch: $d_j = 0$ for $j < 3$ and for $j > 4$
d
d_1
d_2
d_3
d_4
d_N
d^-

Fig. 7.17: Example of branching on an S2 set.

The corrected vector is created, with each of the i elements calculated as $C^{i-}v^-$, and thus representing the state of row i if the branch is carried out.

The corrected vector represents the effects of branching in such a way that only the straddle point variables can take non-zero values (i.e. of producing the cheapest legal S2 set with the same average reference row as the LP solution). Although, as mentioned above for S1 sets, no decision is made as to where to branch until the set has been chosen as the entity to branch upon, this is the branch that is used as part of the process of estimating the degradation to the set.

This degradation can now be calculated, as in the case of S1 sets, by totalling the "distance" moved by the per unit degradation for each row (calculated in the same way as for S1 sets, as described in Appendix 1G).

7.3.2. Conventional Branching Strategies for Special Ordered Sets

Once it has been decided to branch upon a special ordered set, the conventional strategy used to make binary branches is as follows (from [Beale and Forrest, 1976]).

Firstly, determine which variables are the first and last set members to take non-zero values in the present LP solution, and name those variables δ_A and δ_B. A vector of interpolated coefficients can be created corresponding to any variable δ_j between δ_A and δ_B by calculating , for each row i,

$$C_{ij} = (1-\theta)C_{iA} + \theta C_{iB}$$

where θ is defined by the equation

$$X_j = (1-\theta)X_A + \theta X_B$$

The differences between the interpolated row coefficients and the actual row coefficients of the problem act as a guide to the extent to which the current LP solution misrepresents the consequences of giving the variable δ_j the reference row value X_j in the optimal IP solution. Thus, a process similar to the comparison of the corrected and uncorrected vectors is carried out for each variable between and including δ_A and δ_B. The difference between the real and interpolated coefficients for each row i is weighted by the appropriate per unit movement penalty for the row (as described in Appendix 1G), to give a measure of the misrepresentation of the row. The sum of these misrepresentations is calculated over all the rows, to give the final total measure of misrepresentation for the variable.

The variable chosen for branching is the one which is being misrepresented the most (i.e. whose total measure of misrepresentation is the highest).

Branching on S1 sets

Once the variable (δ_W) which suffers the worst misrepresentation has been determined, the conventional branching strategy for an S1 set is to form two branches

(1) setting $\delta_j = 0$ if the corresponding $X_j \leq X_w$

(2) setting $\delta_j = 0$ if the corresponding $X_j > X_w$

as seen if Fig. 7.18 below.

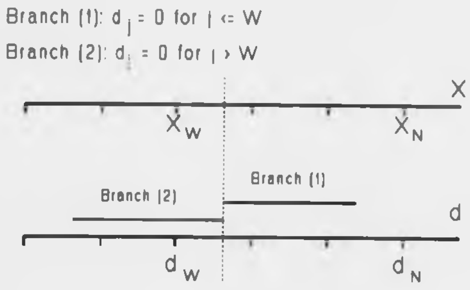

Branch (1): $d_j = 0$ for $j <= W$
Branch (2): $d_j = 0$ for $j > W$
X
X_W
X_N
Branch (1)
Branch (2)
d
d_W
d_N

Fig. 7.18: The binary branching strategy as applied to S1 sets.

The order in which these two branches are applied depends upon the estimated degradation to the LP solution incurred by making the branches. The branch which degrades the LP solution the least will be made first.

For the purposes of calculating the estimated degradations it is assumed that branch (1) (where $\delta_j = 0$ if the corresponding $X_j \leq X_w$) produces the same degradation as if δ_{w+1} was the only variable allowed to take a non-zero value, and branch (2) (where $\delta_j = 0$ if the corresponding $X_j > X_w$) produces the same degradation as if δ_w was the only variable to take a non-zero value.

Corrected and uncorrected vectors are created and the effects of the change summed over the rows of the problem.

<u>Branching on S2 sets</u>

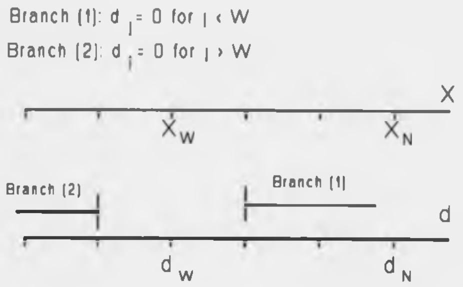
In the case of S2 sets, once δ_w has been determined, the conventional branching strategy is to form two branches

(1) setting δ_j to zero if the corresponding $X_j < X_w$
(2) setting δ_j to zero if the corresponding $X_j > X_w$

as in Fig. 7.19 below. (N.B. The arrow-headed lines in the figure describe the branches where the variables are set to ZERO).

The order in which these two branches are made again depends upon the size of the estimated degradations caused by branching, with the branch that causes the least degradation being made first.

Note that the effect of the branches in either circumstance is that variable δ_w is always allowed to be non-zero and one of the variables adjacent to it is allowed (although not forced) to take a non-zero value.



Branch (1): $d_j = 0$ for $j < W$
Branch (2): $d_j = 0$ for $j > W$
X
X_W
X_N
Branch (2)
Branch (1)
d
d_W
d_N

Fig. 7.19: The binary branching strategy as applied to S2 sets.

The estimated degradations caused by making the branches on the S2 set can again be calculated by making use of corrected and uncorrected vectors.

7.3.3. Separation Schemes for Multiway Branching on Special Ordered Sets

Two different approaches to multiway branching will now be considered. Method One assumes that the technique of Beale and Forrest (described previously) for deciding which set member to branch on first, is to be used. Method Two ignores the Beale and Forrest technique for choosing the set member and branches in a fixed way, determined by the average reference row value.

The two methods may produce the same branches on a set if there are only a few set members, but if there are more than a few set members, the probability of different branches being made increases.

7.3.3.1. Multiway Branching Separation Method One: Based on Set Member Choice of Beale and Forrest

Method One uses the technique described by Beale and Forrest (see section 7.3.2) for determining where to branch upon a set. Once a set member has been chosen, using the Beale and Forrest "worst misrepresentation" method, the multiway branches applied will be further determined by the type of set, the number of set members and the current bounds on the set.

Multiway branching on S1 sets using Separation Method One

The number and position of the multiway branches made on an S1 set depend on the position of X_w, (the reference row entry for the worst represented variable δ_w, chosen using the Beale and Forrest criterion), with respect to the current upper and lower bounds on the set.

In the case where there are only two set members to branch upon, obviously only two branches can be made. If δ_w is the first or last member of a set within the current bounds of the set, only two branches will be made (see Fig. 7.20 below).

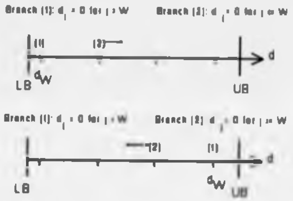

Branch (1): $d_i = 0$ for $i > W$
Branch (2): $d_i = 0$ for $i <= W$
(1)
(2)
d
LB
d_W
UB
Branch (1): $d_i = 0$ for $i < W$
Branch (2): $d_i = 0$ for $i >= W$
(2)
(1)
d
LB
d_W
UB

Fig 7.20: Situations where two branches are made on an S1 set using Separation Method One.

In all other circumstances, three branches can be made, as long as there are at least three set members within the current set bounds that can be branched to (e.g. as in Fig. 7.21).

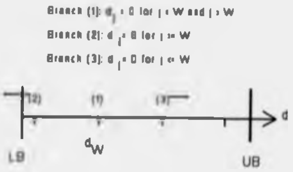

Branch (1): $d_j = 0$ for $j < W$ and $j > W$
Branch (2): $d_j = 0$ for $j >= W$
Branch (3): $d_j = 0$ for $j <= W$
(2)
(1)
(3)
d
d_W
LB
UB

Fig. 7.21: The situation where three branches are made on an S1 set using Separation Method One.

Branch (1) is always made so that only δ_w can take a non-zero value. Branch (2) is always made so that only variables δ_j where $j < W$ (if there are any within the current bounds on the set) can take non-zero values. Branch (3) is made such that only variables δ_j where $j > W$ (if there are any within the current bounds on the set) can take non-zero values.

For the purposes of calculating the estimated degradations it is assumed that branch (2) (if it can be made) produces the same degradation as if δ_{w-1} was the only variable allowed to take a non-zero value, and branch (3) (if it can be made) produces the same degradation as if δ_{w+1} was the only variable to take a non-zero value. Corrected and uncorrected vectors are created and the effects of the change summed over the rows of the problem. The branches will as usual be applied in ascending order of estimated degradation.

Multiway branching on S2 sets using Separation Method One

The number and position of the multiway branches made on an S2 set also depend on the position of X_w with respect to the current upper and lower bounds on the set.

In the case where there are only three set members to branch upon, obviously only two branches can be made. If δ_w is the first or last member within the bounds of a set with four or more members, two branches will be made (see Fig. 7.22 below). If δ_w is the second or second to last member within the bounds of a set with four or more members, three branches will be made (see Fig. 7.23 below). In all other circumstances, four branches can be made, as long as there are at least five set members that can be branched to (e.g. as in Fig. 7.24).

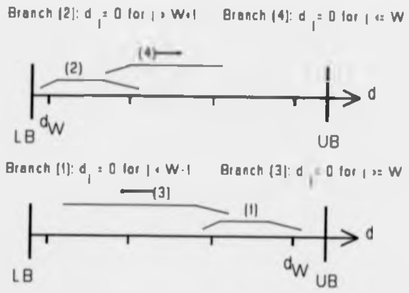

Branch (2): $d_j = 0$ for $j > W+1$
Branch (4): $d_j = 0$ for $j <= W$
(4)
(2)
d
LB
d_W
UB
Branch (1): $d_j = 0$ for $j < W-1$
Branch (3): $d_j = 0$ for $j >= W$
(3)
(1)
d
LB
d_W
UB

Fig. 7.22: The situation where two branches are made on an S2 set using Separation Method One.

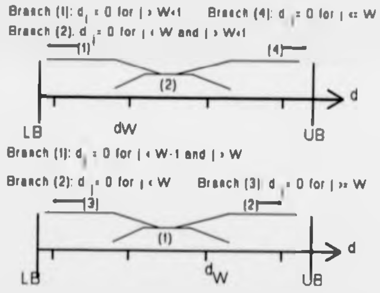

Branch (1): $d_j = 0$ for $j > W+1$
Branch (4): $d_j = 0$ for $j <= W$
Branch (2): $d_j = 0$ for $j < W$ and $j > W+1$
(1)
(4)
(2)
d
LB
dW
UB
Branch (1): $d_j = 0$ for $j < W-1$ and $j > W$
Branch (2): $d_j = 0$ for $j < W$
Branch (3): $d_j = 0$ for $j >= W$
(3)
(2)
(1)
d
LB
d_W
UB

Fig. 7.23: The situation where three branches are made on an S2 set using Separation Method One.

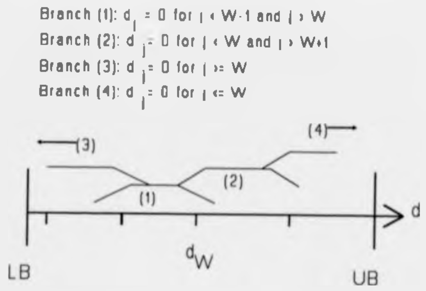

Branch (1): $d_j = 0$ for $j < W-1$ and $j > W$
Branch (2): $d_j = 0$ for $j < W$ and $j > W+1$
Branch (3): $d_j = 0$ for $j >= W$
Branch (4): $d_j = 0$ for $j <= W$
(3)
(4)
(1)
(2)
d
d_W
LB
UB

Fig. 7.24: The situation where four branches are made on an S2 set using Separation Method One.

Branch (1) is made so that only δ_w and δ_{w-1} can take non-zero values (if both are within the current bounds of the set). Branch (2) is made so that only δ_w and δ_{w+1} can take non-zero values (if both are within the current bounds of the set). Branch (3) is made so that only variables δ_j, such that $j < W$ can take non-zero values (if there are any such variables within the bounds on the set). Branch (4) is made so that only variables δ_j, such that $j > W$ can take non-zero values (if there are any such variables within the bounds on the set).

For the purposes of calculating the estimated degradations it is assumed that branch (3) (if it can be made) produces the same degradation as if δ_{W-1} was the only variable allowed to take a non-zero value, and branch (4) (if it can be made) produces the same degradation as if δ_{W+1} was the only variable to take a non-zero value. The estimated degradations for branches (1) and (2) are calculated in a similar way to the way that the estimated degradation is calculated as part of the process of choosing the set as the entity to branch upon. That is, an artificial variable is created between two of the set members and used to represent the effects of branching to them. The artificial variable created to estimate the degradation caused by making branch (1) is shown in Fig. 7.25 below. Note that the artificial variable is created midway between δ_W and δ_{W-1}.

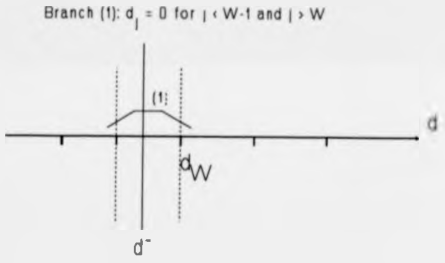


Fig. 7.25: Creation of an artificial variable to estimate the degradation caused by making Branch (1).

Corrected and uncorrected vectors are created for each branch made and the effects summed over the rows of the problem. The branches will as usual be applied in ascending order of estimated degradation.

7.3.3.2. Multiway Branching Separation Method Two: Based on a Fixed Set Member Choice

Separation Method Two creates branches on S1 and S2 sets based on the assumption that the average reference row value provides a good guide as to where to branch.

Multiway branching on S1 sets using Separation Method Two

If multiway branching is considered on an S1 set, an obvious separation under Method Two is to branch as follows. First find the average reference row value X^{*} and the straddle points. Let δ_L and δ_R be the left and right straddle point variables, with corresponding reference row entries X_L and X_R.

Up to four branches can be formed

(1) setting $\delta_j = 0$ for $j=L,...,N$

(2) setting $\delta_j = 0$ for $j=1,...,L-1$ and $j=R,...,N$

(3) setting $\delta_j = 0$ for $j=1,...,L$ and $j=R+1,...,N$

(4) setting $\delta_j = 0$ for $j=1,...,R$

as in Fig 7.26 below

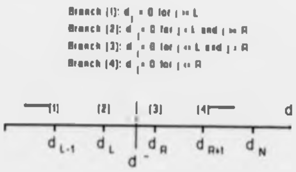

Branch (1): $d_j = 0$ for $j >= L$
Branch (2): $d_j = 0$ for $j < L$ and $j >= R$
Branch (3): $d_j = 0$ for $j <= L$ and $j > R$
Branch (4): $d_j = 0$ for $j <= R$
(1)
(2)
(3)
(4)
d
d_{L-1}
d_L
d^-
d_R
d_{R+1}
d_N

Fig. 7.26: Multiway Branches made on S1 sets under Separation Method Two.

It is hoped that either branch (2) or branch (3) will yield the optimal solution. Note that some of the branches might not always be necessary e.g. if δ_L or δ_R are the first or last members of the set respectively that are within the current set bounds.

The estimated degradations incurred by making branches (2) and (3) (i.e. by branching to the left and right straddle points respectively) have already been calculated as part of the process of choosing the set as the entity to be branched upon (see section 7.3.1). The estimates for branches (1) and (4) are calculated in exactly the same way. The branches are applied in ascending order of the estimated degradation that they will cause.

Multiway branching on S2 sets using Method Two

For S2 sets, up to three branches can be formed

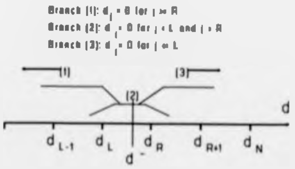
(1) setting $\delta_j = 0$ for $j=R,\ldots,N$

(2) setting $\delta_j = 0$ for $j=1,\ldots,L-1$ and $j=R+1,\ldots,N$

(3) setting $\delta_j = 0$ for $j=1,\ldots,L$

as seen in Fig. 7.27 below.

These branches seem even more natural than the usual binary separation. It is hoped that branch (2), where the two straddle point vectors are allowed to take non-zero values, will yield the optimal solution. Note that again, some of the branches might not always be necessary e.g. if δ_L or δ_R are the first or last members of the set respectively that are within the current set bounds.



Branch (1): $d_j = 0$ for $j \geq R$
Branch (2): $d_j = 0$ for $j < L$ and $j > R$
Branch (3): $d_j = 0$ for $j \leq L$
(1)
(3)
(2)
d
d_{L-1}
d_L
d_R
d_{R+1}
d_N
d^*

Fig. 7.27: Multiway branches made on S2 sets under Separation Strategy Two.

The estimated degradations incurred by making the above branches are again calculated under the assumption that the average reference row value does provide a good guide to the best place in the set to branch. The estimated degradation caused by making branch (2) has already been calculated as part of the process of choosing the set as the entity to be branched upon (see section 7.3.1). The estimates for the other two branches are calculated in the same manner as the estimates for members of S1 sets. That is, for the purposes of calculating the estimated degradations it is assumed that branch (1) (if it can be made) produces the same degradation as if δ_L was the only variable allowed to take a non-zero value, and branch (3) (if it can be made) produces the same degradation as if δ_R was the only variable to take a non-zero value.

The branches are again applied in ascending order of the estimates of the degradation that they will cause.

7.3.4. Computational Results of Trees with Multiway Branching

It is important to test separation schemes such as those suggested above on real problems as there is a trade-off between exploring one entity in its entirety and a more depth first exploration of the tree. The various "depth-first" and "consider all" node selection strategies discussed in Chapter

Six were thus used in conjunction with the multiway branching strategies previously discussed for special ordered sets. Tests were performed on the problems shown in Table 7.2 below.

Problem	Category	NROW	NCOL	NGLENT	Description
DOM1	6	796	585	11	10S1(3),1S2(11)
MCA	6	412	648	22	1S1(5),1S1(6),1S2(4), 18BV,1GI2
MINE1	2	351	320	155	5S1(15),150BV
MINE2	2	359	338	168	8S1(10),160BV
SETX	4	13	21	3	3S1(6)

Key:

2S1(3) =	2 S1 sets, each with 3 members
5S2(6) =	5 S2 sets, each with 6 members.
3BV =	3 binary variables.
7GI4 =	7 general integers with lower bounds of 0 and upper bounds of 4.

Problem Categories: Medium Combinatorial (2); Small MIP (4); Large MIP(6).

Table 7.2: Test problems for multiway branching on special ordered sets.

The two new problems, MINE1 and MINE2 are strategic planning models concerning the closure of coal mines over a period of years.

Branching variable priorities

Special ordered sets and binary variables are often used to represent very important features of a model, such as mutually exclusive decisions or different modes of operation. It is therefore likely that special ordered sets and relevant binary variables should be preferred for branching over other less important variables. For instance, there is no point in deciding what colour a factory will be painted if it is built before deciding whether or not to build it.

In order to establish the merits of using branching variable priorities with multiway branching techniques, the test problems were attacked using the following combinations of node selection and branching strategies.

BB:no priority	Binary branching strategy, choosing where to branch on the set using the Beale and Forrest criteria, no branching variable priorities defined.
MWB:B+F:no priority	Multiway branching strategy, choosing where to branch on the set using the Beale and Forrest criteria (i.e. Separation Method 1 from section 7.3.3), no branching variable priorities defined.

MWB:B+F:priority	Multiway branching strategy, choosing where to branch on the set using the Beale and Forrest criteria (i.e. Separation Method 1 from section 7.3.3), special ordered sets given priority for branching over other variables.
MWB:Fix:no priority	Multiway branching strategy, choosing where to branch on the set using fixed branching criteria (i.e. Separation Method 2 from section 7.3.3), no branching variable priorities defined.
MWB:Fix:priority	Multiway branching strategy, choosing where to branch on the set using fixed branching criteria (i.e. Separation Method 2 from section 7.3.3), special ordered sets given priority for branching over other variables.
Strategy X:Problem Y	Problem Y attacked using node selection strategy X (as defined in section 6.2 on pages 188-190).

The computational results are shown in Figs. 7.28 to 7.47.

Problem SETX (Small MIP)

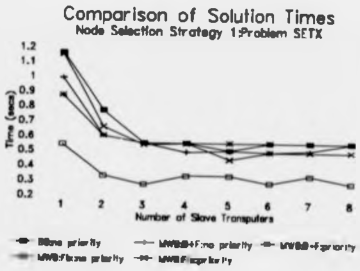


Fig. 7.28: Multiway branching on SETX:node selection strategy 1.

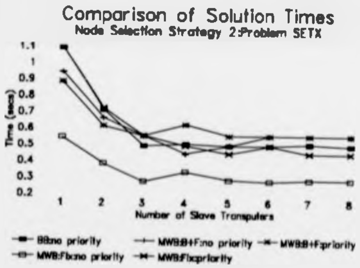


Fig. 7.29: Multiway branching on SETX:node selection strategy 2.

Problem SETX (Small MIP)

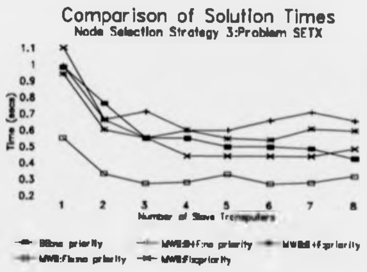


Fig. 7.30: Multiway branching on SETX:node selection strategy 3.

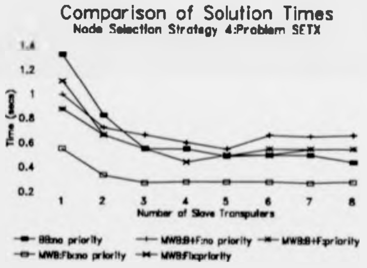


Fig. 7.31: Multiway branching on SETX:node selection strategy 4.

Problem MINE1 (Medium Combinatorial)

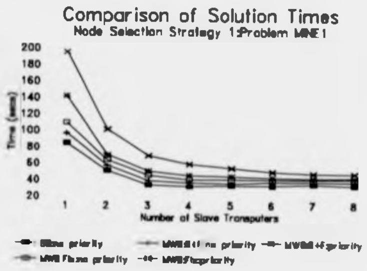


Fig.7.32: Multiway branching on MINE1:node selection strategy 1.

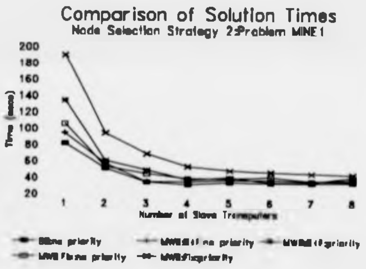


Fig.7.33: Multiway branching on MINE1:node selection strategy 2.

Problem MINE1 (Medium Combinatorial)

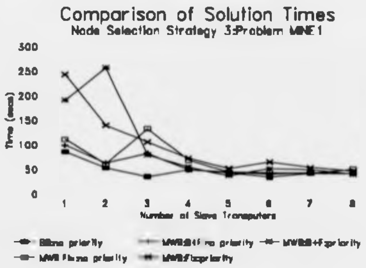


Fig.7.34: Multiway branching on MINE1:node selection strategy 3.

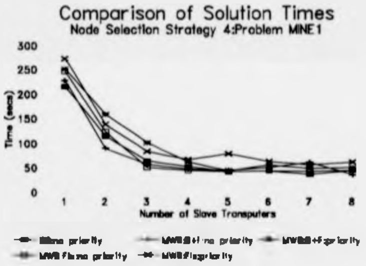


Fig.7.35: Multiway branching on MINE1:node selection strategy 4.

Problem MINE2 (Medium Combinatorial)

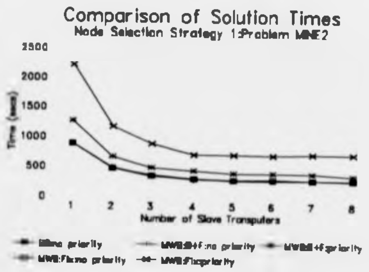


Fig.7.36: Multiway branching on MINE2:node selection strategy 1.

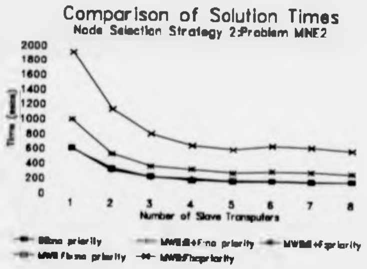


Fig.7.37: Multiway branching on MINE2:node selection strategy 2.

Problem MINE2 (Medium Combinatorial)

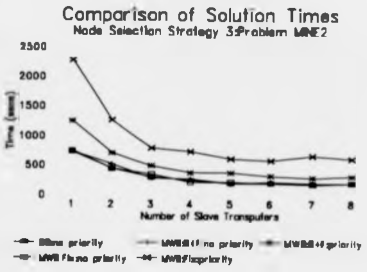


Fig.7.38: Multiway branching on MINE2:node selection strategy 3.

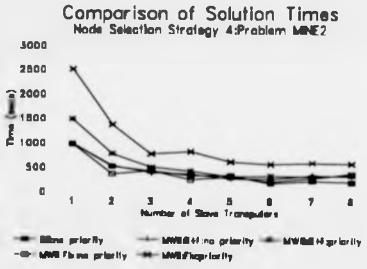


Fig.7.39: Multiway branching on MINE2:node selection strategy 4.

Problem DOM1 (Large MIP)

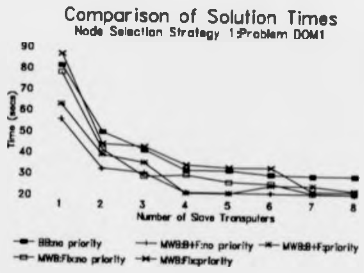


Fig.7.40: Multiway branching on DOM1:node selection strategy 1.

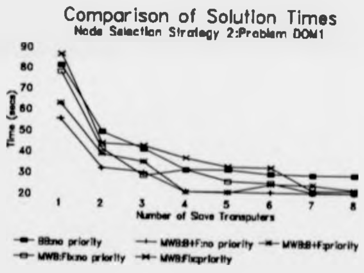


Fig.7.41: Multiway branching on DOM1:node selection strategy 2.

Problem DOM1 (Large MIP)

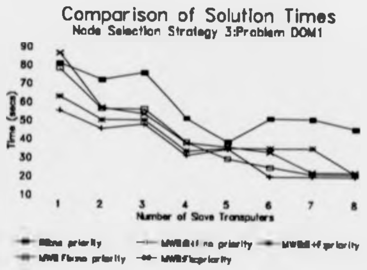


Fig.7.42: Multiway branching on DOM1:node selection strategy 3.

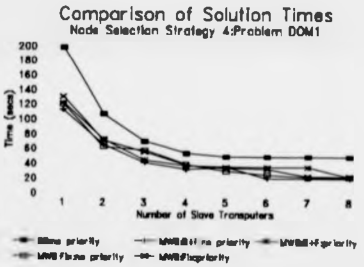


Fig.7.43: Multiway branching on DOM1:node selection strategy 4.

Problem MCA (Large MIP)

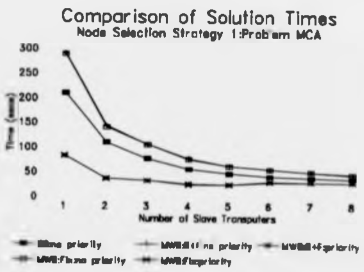


Fig.7.44: Multiway branching on MCA:node selection strategy 1.

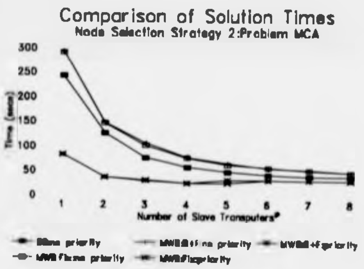


Fig.7.45: Multiway branching on MCA:node selection strategy 2.

Problem MCA (Large MIP)

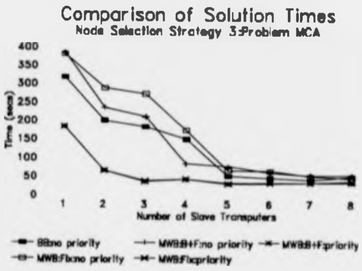


Fig.7.46: Multiway branching on MCA:node selection strategy 3.

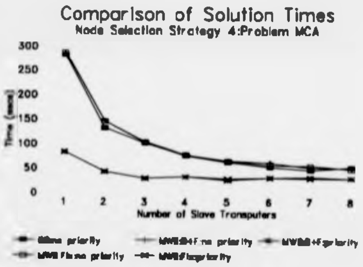


Fig.7.47: Multiway branching on MCA:node selection strategy 4.

Discussion of Results

Appendix 4B lists the computational results in full. The test problems will be considered here in order of their different size categories.

Small Problems

It can be seen from Figs. 7.28 to 7.31 that, no matter which of the node selection strategies is used, the best results for solving the small MIP SETX come from using the fixed separation approach to branching. The fixed separation approach with no branching priorities gives the best overall results in all cases. As this problem only contains sets as integer-constrained global entities, the effects of the priorities can be ignored, as all sets were given equal priority. Any slight difference in results from runs where priorities were or were not used are probably caused by the nondeterminacy of the algorithm. Thus, the good solution times for this problem were due to the use of the fixed separation approach to branching on special ordered sets. It is worth noting that these solution times were usually better than those for the binary separation approach.

Since SETX is always solved very quickly (and the overall average transputer usage never exceeds three), most of the solution times are very similar and it is thus not possible to make much of a comparison between the various node selection strategies. It is noted however that the "compare all nodes" strategies (i.e. strategies 1 and 2) do tend to perform slightly better overall. They produce slightly faster solutions than the "depth first" strategies (i.e. strategies 3 and 4) when used in conjunction with the Beale and Forrest separation approach and solutions that are at least as good when used with the fixed separation approach. Within the "compare all" and "depth first" strategy classes there appears to be no benefit to using the Forrest-Hirst-Tomlin criterion to compare nodes as opposed to the best estimate criterion.

Medium-Sized Problems

The medium sized (combinatorial) problem MINE1 reacts as follows in the various tests (as shown in Figs. 7.32 to 7.35). In this case, the fixed separation branching strategies produce solution times that are generally worse than those of the strategies based on the Beale and Forrest separation criterion, although the gap narrows as more transputers are used (and the average transputer usage settles down between three and four).

The use of branching priorities has a fairly large detrimental effect on the problem solution times, although the damage is smaller when the "depth first" node selection strategies are used, since the depth first searches tended to branch on the sets early in the search and proceed from there.

The binary branching strategy actually performs the best overall, but the multiway branching strategies produce comparable results when four or more slave transputers are used.

Again, the "consider all" node selection strategies produce better results than the "depth first" strategies (especially when only a small number of transputers are used), as less solutions are found and fewer nodes searched overall. The "depth first" strategies still produce reasonable results however, when four or more slave transputers are used. This is due to reasonably good solutions being found by the depth first search, allowing the fathoming of many nodes at once.

Within the "compare all" and "depth first" strategy classes there appears to be some benefit obtained by using the Forrest-Hirst-Tomlin criterion in conjunction with the best estimate criterion to compare nodes as opposed to using only the best estimate criterion.

The medium sized (combinatorial) problem MINE2 performs in a fairly similar way to MINE1 (as can be seen in Figs. 7.36 to 7.39). The same effects are noted for both problems, although on different scales.

Again the use of branching priorities has a detrimental effect on the problem solution times, although the effect is much more visible for this problem. When the problem is attacked without using branching priorities it is solved much more quickly. Indeed, the solution times obtained using multiway branching are at least as good as those obtained using binary branching if branching priorities are not given to the sets.

The fixed branching strategies still produce results slightly worse than those of the strategies based on the Beale and Forrest criterion, although if branching priorities are not used, the results are very similar indeed for this problem as the average transputer usage settles down to between four and five.

The "consider all" node selection strategies still tend to provide slightly better results than the "depth first" strategies. Within the "compare all" and "depth first" strategy classes there again appears to be a benefit obtained by using the Forrest-Hirst-Tomlin criterion in conjunction with the best estimate criterion to compare nodes as opposed to using only the best estimate criterion.

Large Problems

The test results for the large (MIP) problem DOM1 can be seen in Figs. 7.40 to 7.43).

Firstly, it is encouraging to see that all of the multiway branching strategies produce better results than the binary branching strategy during most of the tests on this problem.

The fixed branching strategies again produce worse results than those of the strategies based on the Beale and Forrest criterion (although all strategies are again hindered by the use of branching priorities). Also, the "consider all" node selection strategies again tend to provide slightly faster solutions than the "depth first" strategies, especially when only a small number of transputers are used. Within the "consider all" and "depth first" categories, very little difference is observed between the effects of comparing nodes using the Beale and Forrest criterion and using the best estimate criterion for this problem.

The results for the final large (MIP) problem, MCA, are shown in Figs 7.44 to 7.47. Again, it is gratifying to see that it is possible to improve upon the solution time for the problem by making use of multiway branching as opposed to the conventional binary branching techniques.

The solution times for this problem however are greatly affected by the use of branching priorities. When branching priorities are used, the fixed branching strategies produce solution times that are very similar to those of the strategies based on the Beale and Forrest criterion. Both of these methods produce better solution times than the binary separation method under these conditions.

However, when branching priorities are not used, the familiar pattern of the fixed separation method producing worse solution times than the Beale and Forrest method is observed. In these cases, the solution times depend upon the node selection strategy used. Using the "consider all" strategies produces solution times that are slightly worse than the binary separation method (although the difference is quite small when several transputers are used). Using the "depth first" strategies under these circumstances allows solution times that are very similar to or only slightly worse than those produced by the binary separation method.

Within the "consider all" category of the node selection strategies no advantage is gained by using the Forrest-Hirst-Tomlin criterion in conjunction with the best estimate criterion as a way of comparing nodes. Within the "depth first" category however, better results were usually obtained by using the best estimate as the only criterion when comparing nodes.

Conclusions

When attacking the larger problems in the test set, the performance of our parallel Branch and Bound algorithm was improved by making more than two branches on special ordered sets. When attacking the smaller problems in the test set, multiway branching techniques were used to achieve a performance close to that obtained by binary branching, as long as a fairly large number of slaves were used.

The small number of test results obtained seem to indicate that multiway branching is best used in conjunction with a "consider all" node selection strategy (such as the default strategy decided upon previously), as this seems to produce the best results for medium to large problems. It may be worth considering changing the method of comparing nodes to allow the inclusion of the Forrest-Hirst-Tomlin criterion once an integer feasible solution has been found when dealing with problems containing special ordered sets.

No real preference can be given to either of the methods for multiway branching due to the small number of problems available, although it is noted that the Beale and Forrest separation criterion for branching on special ordered sets have given better or equivalent results for most of the test problems.

Finally, giving branching preference to special ordered sets has been shown to be a two-edged sword when attacking our set of test problems. It has proved to be of great benefit to the solution of one of the test problems, but a hindrance to the solution of the others. It is worth noting that the benefits gained from using priorities were eroded as more slave transputers were used, perhaps an indication that priorities can be ignored if a large number of transputers are to be used.

8. Conclusions and Recommendations

Conclusions as to the value of the algorithms used and the way in which they were implemented can be reached by considering the results of the previous chapters.

One perhaps obvious conclusion can be drawn immediately referring to the implementation of the present parallel farming algorithm, as discussed in Chapter Five. When using network MIMD hardware such as the transputer, where the processors do not have access to a shared global memory, the farming algorithm used will eventually face the problem of a bottleneck at the master processor. The master processor performs a cycle wherein it chooses LPs to be solved, and sends them to idle slaves. The cycle is only interrupted by LP solutions being returned, or by the optimal solution to the problem being found. If the master processor cycle is frequently interrupted by the return of LP solution information via messages from the slave processors, there will not be much time to send out new LP relaxations to idle slaves.

This raises the issue of the required balance between calculation and message-passing on the processors. Chapter Five indicates that, for a small number of processors at least, the bottlenecking problem can be minimised by careful design of the algorithm and its implementation.
The master processor cycle was modified so that the choice

of which LP to send next could be made as quickly as possible. The actual messages sent to and from the slave processors were also altered so that they could be sent more quickly. This allowed the master algorithm to send out more LP problems in a given amount of time, and allowed the slave algorithms to spend more time in solving the LPs and less time in passing messages.

However, when larger numbers of processors are used, the bottlenecking problem will again become an inhibiting factor on the performance of the algorithm. The results from the smaller test problems attacked in Chapter Five probably reflect what will happen with the larger problems when more slave processors are available. As more messages need to be sent to more slave processors, a limit will be reached on the average number of slave transputers that can be used to solve a problem. After a certain point, the master algorithm cannot send out enough LP problems to keep all the idle slaves busy, as it is interrupted too frequently by the return of LP results from busy slaves.

The answer to this problem would be to move from the centralised list farming algorithm presently in use, to a distributed list farming algorithm, if more slave processors are available. It is the fact that all the information concerning the enumeration of the search tree is placed on the master processor for decision making purposes (i.e. that there is a centralised list), that causes the bottleneck at the master processor. If the slaves were able to pass their results to different "masters" for use, the bottlenecking problem, although not removed, would be reduced (or perhaps more accurately, spread out).

An outline for a proposed distributed list (DL) farming algorithm for use with ten or more transputers is as follows.

The three transputers that are directly connected to the root (i.e. master) transputer could be designated sub-masters. The search space of the problem to be attacked could be pre-partitioned based on user-defined priorities, and each of the sub-masters would be given an appropriate subset of the search space to work on. Each of the sub-masters would also be allocated a number of slave processors (i.e. a "sub-slave" group) with which it could easily communicate (allowing for the limited number of processors that could be directly connected to each sub-master). The algorithm could then proceed in a similar way to the centralised list (CL) algorithm, with each sub-master generating its own candidate list of nodes to be branched upon, choosing LP relaxations and sending them to its sub-slaves for solution. Any change in the cutoff information resulting from an integer-feasible solution being found would be broadcast to all the other processors by the appropriate sub-master.

This concept of "sub-masters" and "sub-slave groups" could be extended dynamically when bottlenecking problems again became serious, so that each of the sub-masters could transform one or more of its sub-slaves into "sub-sub-masters" as the need arose.

The pre-partitioning of the search space would also be a good step forward for the algorithm since for the altered algorithm to be truly useful a Dual Simplex algorithm would have to be placed on the slaves/sub-slaves, along with the existing Primal Simplex algorithm already in use. The Dual Simplex algorithm can be used to very quickly solve an LP problem if the information on the parent of the problem is available. Thus, if the sub-master chooses as the next node to be branched upon, the son of a node just solved, the slave processor containing the information on the parent would be chosen to solve the new LP. The only difference between the parent and son nodes would be the bounds on one of the integer-constrained entities. Thus, only these new bounds need be sent to the slave, which could then use the Dual Simplex algorithm to quickly find the new LP solution.

It is worthwhile at this point to note why the potentially useful combination of the Dual Simplex algorithm and the policy of solving an LP relaxation on the same slave as its parent was not used by our centralised list farming algorithm when testing the depth-first node selection strategies in Chapters Five and Six.

Firstly, the version of the XPRESS-MP code on which our algorithm is based does not support a Dual Simplex algorithm. Although more recent versions of XPRESS-MP have added a Dual Simplex algorithm, in the process of doing so, many of the data structures used to hold LP information were changed. Thus, in order to make use of a Dual Simplex algorithm, either a large proportion of the coding of our present algorithm would have to be rewritten so as to enable the use of the new XPRESS-MP data structures, or a Dual Simplex algorithm would have to be written from scratch. Neither of these possibilities were considered, due to the time that would be involved.

Secondly, the transputers used did not have enough memory to make room for both a Dual and a Primal Simplex algorithm on each slave, as well as the necessary message-passing and node creation routines.

If the sub-master, sub-slave algorithm was to be implemented in the future, it would be worthwhile to change the data structures used by the LP-solver so that the Dual and Primal Simplex algorithms of the later versions of XPRESS-MP could be used. Although this would involve replacing the current hardware with a system where the transputers had access to a greater amount of personal memory, this would have to be done anyway in order to replace the current clear path message-passing system, which could not be used with a larger number of processors.

The clear path system was devised so that only the minimum amount of the memory of slave processors had to be allocated to message-passing routines. Although a clear path system could conceivably still be implemented for the message-passing between members of sub-master, sub-slave clusters, it is thought that the total message-passing overheads amassed by a system containing many sub-master, sub-slave clusters would be large. This problem can forseeably be overcome in one of two ways.

Firstly, a fully buffered message-passing system could be developed, wherein the full LP relaxation or LP solution details being passed from transputer to transputer could be stored in data structures if enough memory was made available.

Secondly, a system could be devised to make use of the new Inmos T9000 transputer ([Inmos, 1991]). One of the claims that Inmos make for such a system is that any of the transputers therein would be able to communicate **directly** with any of the other transputers. This would obviously remove the need to save LP information on a slave transputer before passing it on to its final destination.

The introduction of hardware with more personal memory available on each transputer would have several advantages beyond the replacement of the clear path message-passing system. Additional memory available on the master processor (or the sub-master processors) would allow more (or all) of the long node information for the MIP problem to be stored in the personal memory, thus reducing (or removing) the delays caused by the disk-reading operation. Indeed, if the sub-master, sub-slave algorithm was to be introduced, this additional memory would be essential, since the sub-masters, which are not connected directly to the PC, cannot read from or write to the hard disk of the PC.

Even when using the current centralised list farming algorithm, if the disk reading operation could be removed from the master algorithm, this would help greatly to reduce the non-determinism of the algorithm. If the long node information could always be retrieved from memory in the same amount of time, the same number of LPs should always be sent out to the slaves. Although it is not certain that the slaves would always be interrupted at the same point in their cycles, the likelihood is greatly increased.

Chapter Six shows that the node selection strategy used by the parallel algorithm can have a great effect on problem solution times. A strategy whereby all candidate nodes are considered for branching and are compared using the best estimate criteria has been chosen as the default node selection strategy for use with our parallel Branch and Bound algorithm. This strategy has performed well when used to attack the (admittedly small) selection of problems in our test set. Although not many test problems were available, they were of different types and sizes, i.e. combinatorial and MIP problems of different sizes, that have been formulated using assorted types of integer-constrained entity. Using this node selection strategy almost always produced one of the better solution times achieved for each of the test problems in Chapter Six. In the circumstances where using this strategy did not lead to the fastest solution times for a problem, the solution times it did achieve were not much worse, especially if a large number of slave transputers were used. These results are mainly due to the strategy successfully finding good integer-feasible solutions which are of great use in fathoming nodes and reducing the overall search space.

Chapter Seven shows that the use of multiway branching techniques can help to reduce the solution times for certain types of problem in our test set. The test problems formulated using general integers did not respond particularly well to the use of multiway branching techniques, as the extra branches made only served to make work for idle hands. In the case of the test problems formulated using special ordered sets however, some benefit was gained by making use of multiway branching techniques. Experiments were carried out to test the effects of using two different multiway branching strategies in conjunction with several different node selection strategies. The small set of test problems attacked contained small, medium and large IP problems formulated using both S1 and S2 sets. The results show that, for the larger MIP test problems at least (i.e. the category in which we are interested), the use of multiway branching techniques can increase the performance of the parallel Branch and Bound algorithm beyond that achieved by making use of the conventional binary branching strategy. When the smaller problems were attacked, the solution times achieved were still similar to, though not as good as, those achieved under the binary branching scheme.

The best test results were achieved by making use of multiway branching techniques in conjunction with a "consider all nodes" node selection strategy similar to the default strategy decided upon above. The results indicated however, that it may be worthwhile to change the method of comparison of nodes to incorporate the use of the Forrest-Hirst-Tomlin criterion once an integer feasible solution has been found when dealing with problems formulated using special ordered sets.

The use of simple branching variable priorities in conjunction with multiway branching techniques was shown to be of varying value. In most of the experiments carried out, the presence of branching variable priorities actually hindered the search for the optimal solution, although in the case of one large problem, exactly the opposite effect was observed. It is thus thought that the default approach will be to not use such priorities, but to allow the user to define priorities for any problem if desired.

In conclusion, the small number of test results obtained indicate that the parallelism of the Branch and Bound algorithm can be exploited in order to reduce the solution times of certain categories of IP problem, especially the larger MIP problems in which we are interested. The structure of certain types of large problem (i.e. those formulated using special ordered sets) seems to be such that multiway branching techniques can be effective in their solution. Good results may be obtained for some problems even when using a relatively small number of parallel processors, as long as the algorithm and code are carefully designed.

Although the limits imposed by the hardware used have prohibited the solution of some extremely large problems, it is thought that these too would be soluble if these limits could be overcome. A larger number of parallel processors will be necessary to effectively solve very large MIP problems, and thus a more advanced parallel algorithm (such as the sub-master, sub-slave algorithm discussed above) will need to be implemented to overcome the bottlenecking problems foreseen. New technology, such as the Inmos T9000 transputer mentioned above, however, may reduce the effects of bottlenecking by allowing easier communication between all processors.

It will thus be possible in the future to carry out further research into the design and implementation of parallel algorithms for the solution of very large MIP problems using transputer-based hardware. Judging by the results achieved with a small number of parallel processors, this should prove to be a fruitful area for years to come.

References

Abdelrahman, T.S. and Mudge, T.N. (1988), "Parallel branch and bound algorithms on hypercube multiprocessors", in Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications - Vol II, ACM Press, 1988, pp 1492-1499.

Allan, S.J. and Oldehoeft, R.R. (1985), "HEP SISAL:Parallel Function Programming", in Parallel MIMD Computation: HEP Supercomputer and its Applications, Cambridge, MA: MIT Press, 1985, pp 123-150.

Ashford, R.W., Connard, P. and Daniel, R.C. (1992), "Experiments in Solving Mixed Integer Programming Problems on a Small Array of Transputers", JORS 43(5), May 1992, pp 519-531.

Balas, E. (1963), "Linear Programming with Zero-One Variables" (in Rumanian), Proceedings of the Third Scientific Session on Statistics, Bucharest, December 5-7.

Balas, E. (1965), "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Operations Research 13(4), pp 517-548.

Balinski, M. and Wolfe, P. (1963), "On Benders' Decomposition and a Plant Location Problem", Mathematica Working Paper ARO-27, 1963.

Barnes, G.H., Brown, R.M., Kato, M., Kuck, D.J., Slotnick, D.L. and Stokes, R.A. (1968), "The ILLIAC IV computer", IEEE Trans. Comput., 17, pp 746-757.

Batcher, K.E. (1974), "STARAN parallel processor system hardware", AFIPS Conf. Proc. 1974 National Computer Conference, May 1974, pp 405-410.

Batcher, K.E. (1980), "Design of a massively parallel processor", IEEE Trans. Comput., 29, September 1980, pp 836-840.

Beale, E.M.L. (1954), "An alternative method for linear programming", Proceedings of the Cambridge Philosophical Society 50 (1954), pp 513-523.

Beale, E.M.L. (1958), "A method of solving LP problems when some but not all of the variables must take integral values", Statistical Techniques Research Group, Technical Report no. 19, Princeton University, July 1958.

Beale E.M.L. and Tomlin J.A. (1970), "Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables", Proceedings of the Fifth International Conference on Operational Research, ed Lawrence J., Tavistock Publications, pp 447-454.

Beale, E.M.L. and Forrest, J.J.H. (1976), "Global optimisation using special ordered sets", Mathematical Programming 10, pp 52-69.

Beale E.M.L. (1977), "Integer programming" in "The state of the art in numerical analysis", ed Jacobs D.A.H., Academic Press , 1977.

Beale E.M.L. (1979), "Branch and bound methods for mathematical programming systems", Annals of Discrete Mathematics 5 (1979), pp 185-191.

Beetem, J., Dennean, M. and Weingarten, D.H. (1985), "The GF-11 supercomputer", Proc. 12th Annual Int. Symp. on Computer Architecture, Boston, Mass., pp 108-118.

Benders, J.F. (1962), "Partitioning Procedures for Solving Mixed Variables Programming Problems", Numerische Mathematik 4,pp 238-252.

Ben-Israel, A. and Charnes, A. (1962), "On Some Problems of Diophantine Programming", Cahiers du Centre d'Etudes de Recherche Operationelle 4, pp 215-280.

Berg, R.O., Schmitz, H.G. and Nuspl, S.J. (1972), "PEPE - an overview of architecture, operation and implementation", Proc. IEEE Natl. Electron. Conf. 27, pp 312-317.

Bhuyan, L.N. and Agrawal, D.P. (1984), "Generalised hypercube and hypercube structures for a computer network", IEEE Trans. Comput. 33, pp 323-333.

Boehning, R.L., Butler, R.M. and Gillett, B.E. (1988), "A parallel integer linear programming algorithm", European Journal of Operational Research 34(1988), pp 393-398.

Borosh, I. and Treybig, L.B. (1976), "Bounds on Positive Integral Solutions of Linear Diophantine Equations", Proc. Amer. Math. Soc. 55, pp 299-304.

Brooks, R. and Geoffrion, A. (1966), "Finding Everett's Lagrange Multipliers by Linear Programming", Operations Research 14(16), pp 1149-1153.

Bustos, E., Lavers, J.D. and Smith, K.C. (1979), "A parallel array of microprocessors - an alternative solution to diffusion problems", COMPCON'79 (Fall) Digest, pp 380-389.

Cannon, T.L. and Hoffman, K.L. (1989), "Large-Scale 0-1 Linear Programming on Distributed Workstations", Working Paper, Dept. of Operations Research, George Mason University, Fairfax, VA, February 1989.

CDC (1983), CDC CYBER 200 Model 205 computer system hardware reference manual, Publication 60256020 (St. Paul, Minnesota: Control Data Corporation).

Chakrabarti, C. and JáJá, J. (1990), "Systolic architectures for the computation of the Discrete Hartley and the Discrete Cosine Transforms based on Prime Factor Decomposition", IEEE Trans. Comput., 39(11), pp 1359-1368.

Charlesworth, A.E. and Gustafson, J.L. (1986), "Introducing replicated VLSI to supercomputing: the FPS-164/MAX scientific computer", IEEE Trans. Comput., 19(3), pp 10-23.

Chen, S.S. (1984), "Large-scale and high-speed multiprocessor system for scientific applications:CRAY X-MP Series", in "High Speed Computation" (ed. J.S. Kowalik), NATO ASI Series vol. 7, Berlin:Springer, pp 59-67.

Childress, J.P. (1969), "Five Petrochemical Industry Applications of Mixed Integer Programming", Bonner and Moore Associates, Inc., Houston, March 1969.

Christ, N.H. and Terrano, A.E. (1986), "A micro-based supercomputer", Byte Magazine, April, pp 145-160.

Christofides, N. (1970), "The shortest Hamiltonian Chain of a graph", J. SIAM 19, pp 689-697.

Cook, S.A. (1971), "The Complexity of Theorem-Proving Procedures", Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York pp 151-158.

Cray (1976), CRAY-1 computer system reference manual, Publication 2240004 (Mendota Heights, Minnesota:Cray Research Inc.).

Cray (1985), "Introducing the CRAY-2 computer system", Cray Channels, Summer, pp 2-5.

Crowther, W. et al. (1985), "The Butterfly Parallel Processor", IEEE Computer Arch. Tech. Comm. Newsletter, Sept/December 1985, pp 18-45.

d'Acierno, A., De Pietro, G. and Villano, U. (1990), "A Parallel Architecture for Optical Character Recognition", in Proc. Int. Conf. on Parallel Computing:Achievements, Problems and Prospects", Capri, Italy, 3-7 June 1990.

Dakin, R.J. (1965), "A tree search algorithm for mixed integer programming problems", The Computer Journal 8 (1965), pp 250-255.

Dantzig, G. (1951), "Maximization of a linear function of variables subject to linear inequalities", in "Activity Analysis of Production and Allocation", (ed. Tj.C. Koopmans), Wiley, New York, pp 339-347.

Dantzig, G., Fulkerson, D. and Johnson, S. (1954), "Solution of a Large Scale Travelling Salesman Problem", Operations Research 2(4),pp 393-410.

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

Dantzig, G.B. (1963), "Linear Programming and Extensions", Princeton University Press, Princeton, New Jersey.

Dantzig, G.B., Blattner, W.O. and Rao, M.R. (1967), "All Shortest Routes from a Fixed Origin in a Graph", in Theory of Graphs: International Symposium, Gordon and Breach, New York, pp 85-90.

Darlington, J. and Reeve, M. (1981), "ALICE - a multiprocessor reduction machine for the parallel evaluation of functional programming languages", Proc. ACM Conf. Functional Programming Languages and Computer Architectures, New Hampshire, pp 65-75.

Dash Associates (1989), "XPRESS-MP Reference Manual", Dash Associates, Blisworth House, Church Lane, Blisworth, Northants, UK.

deBruin, A., Rinnooy Kan, A.H.G. and Trienekens, H.W.J.M. (1988), "A simulation tool for the performance evaluation of parallel branch and bound algorithms", Mathematical Programming 42(1988), pp 245-271.

Desrochers, G.R. (1987), "Principles of Parallel and Multiprocessing", New York NY: McGraw-Hill.

DeWitt, D., Finkel, R.A. and Solomon, M. (1984), "The Crystal multicomputer: design and implementation experience", Technical Report 553, Dept. of Computer Science, University of Wisconsin, Madison, 1984.

Dijkstra, E. (1959), "A Note on Two Problems in Connection with Graphs", Numerische Mathematik 1, pp 269-271.

Dongarra, J.J. (1987), "Experimental Parallel Computing Architectures", Amsterdam: North-Holland.

Dornheim, M.A. (1985), Caltech, "JPL advancing supercomputer as low-cost tool in aerospace work", in Aviation Week and Space Technology, 22 April, pp 93-101.

Driebeek, N. (1966), "An Algorithm for the Solution of Mixed Integer Programming Problems", Management Science 12(7), pp 576-587.

Efe, K. (1989), "Programming the twisted cube architectures", in Proc. 9th Int. Conf. DCS, June 1989, pp 254-262.

Elder, J., Gottlieb, A., Kruskal, C.K., McAuliffe, K.P., Randolph, L. Snir, M., Teller, P. and Wilson, J. (1985), "Issues related to MIMD shared-memory computers: The NYU Ultracomputer approach", Proc. 12th Int. Symposium on Computer Architecture, June 1985, Boston, Mass., pp 126-135.

El-Dessouki, O. and Huen, W.H. (1980), "Distributed enumeration on between computers", IEEE Trans. Comput. 29(9), pp 818-825.

Everett III, H. (1963), "Generalised Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources", Operations Research 11(3), pp 399-471.

Fazio, D. (1987), "It's really much more fun building a supercomputer than it is simply inventing one", COMPCON, IEEE, February 1987, pp 102-105.

Felton, E.W. (1988), "Best-first branch and bound on a hypercube", in Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications - Vol II, ACM Press, 1988, pp 1500-1504.

Finkel, R.A. and Manber, U. (1985), "DIB - a distributed implementation of backtracking ", Technical Report 588, Dept. of Computer Science, University of Wisconsin, Madison, 1985.

Flynn, M.J. "Very high speed computing systems", Proc. IEEE, 54 (1966), pp 1901-1909.

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 (1974), 736-773.

Fox, B. and Landi, D. (1970), "Searching for the Multiplier in One Constraint Optimization Problems", Operations Research 18(2), pp 252-262.

Gacs, H.P. and Lovasz, L. (1981), "Khachiyan's Algorithm for Linear Programming", Mathematical Programming Study 14, pp 61-68.

Gajski, D., Kuck, D., Lawrie, D. and Sameh, A. (1983) "Cedar - a large scale multiprocessor", IEEE Proc. 1983 Int. Conf. on Parallel Processing, pp 524-529 (London:IEEE).

Garey, M.R. and Johnson, D.S. (1979), "Computers and Intractability, A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, San Francisco, pp 245

Gay, D.M. (1985), "Electronic Mail Distribution of Linear Programming Test Problems", Mathematical Programming Society Committee on Algorithms (COAL) Newsletter 13, pp 10-12.

Gehringer, E.F., Siewiorek, D.P. and Segall, Z. (1987), "Parallel Processing: The Cm* Experience", Digital Press, Bedford, Mass.

Gendron, B. and Crainic, T.G. (1992), "Parallel implementations of a branch and bound algorithm for multicommodity location with balancing requirements", Report 813, Centre for Research on Transportation, University of Montreal.

Geoffrion, A. (1972), "Lagrangian Relaxation and its Uses in Integer Programming", Western Management Science Institute Working Paper No. 195, University of California at Los Angeles, December 1972.

Geoffrion, A.M. and Graves, G.W. (1974), "Multicommodity distribution systems design by Benders' decomposition", Management Science 20(5), pp 822-844.

Glover, F. (1965), "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem", Operations Research 13(6), pp 879-929.

Golden B., Sharda R. and Wasil E. (1988), "Mathematical Programming Software for the microcomputer: recent advances, comparisons and trends", Working paper, School of Business, University of Oklahoma, U.S.A.

Gomory, R.E. (1960), "An Algorithm for the Mixed Integer Program", RM-2597 Rand Corporation, July 1960.

Gomory, R.E. (1963), "An Algorithm for Integer Solutions to Linear Programs" in Recent Advances in Mathematical Programming, (eds Graves and Wolfe),New York,McGraw-Hill,1963.

Gomory, R.E. (1965), "On the relation between integer and non-integer solutions to linear programming problems", Proceedings of the National Academy of Sciences of the United States of America 53 (1965), pp 260-265.

Gomory, R.E. (1967), "Faces of an Integer Polyhedron", Proceedings of the National Academy of Sciences of the United States of America 57 (1967), pp 16-18.

Gomory, R.E. (1969), "Some polyhedra related to combinatorial problems", Linear Algebra and its Applications 2 (1969), pp 451-558.

Harp, J.G., Palmer, K.J. and Webber, H.C. (1987), "Image Processing on the Reconfigurable Transputer Processor", in "Parallel Programming of Transputer Based Machines", (ed. T. Muntean), IOS publishing, Amsterdam, 1988,pp 252-260.

Hayes, J.P. et al. (1986), "A microprocessor-based hypercube supercomputer", IEEE Micro, 6(5), pp 6-17.

Held, M. and Karp, R.M. (1970),"The Travelling Salesman Problem and Minimum Spanning Trees",Operations Research 18, pp 1138-1162.

Held, M. and Karp, R.M. (1971),"The Travelling Salesman Problem and Minimum Spanning Trees:Part II", Mathematical Programming 1, pp 6-25.

Hintz, R.G. and Tate, D.P. (1972), "Control Data STAR-100 processor design", COMPCON, IEEE, September 1972, pp 1-4.

Hoare, C.A.R. (1978), "Communicating Sequential Processes", Comm. of the ACM, vol. 21, no. 8, Aug. 1978, pp 666-677.

Holland, J.H. (1959), "A universal computer capable of executing an arbitrary number of subprograms simultaneously", Proc. East. Joint Comput. Conf. 16, pp 108-113.

Hunt, D.J. (1981), "The ICL DAP and its application to image
processing", in "Languages and Architecture for Image Processing" (eds. M.J.B. Duff and S. Levialdi), Academic Press, London, pp 275-282.

Hwang, K. and Briggs, F.A. (1984), "Computer Architecture and Parallel Processing", New York NY: McGraw-Hill.

IBM (1988), "IBM Mathematical Programming System Extended/370 (MPSX370) Version 2 User Guide", IBM Italia S.p.A., Viale dell'Oceano Pacifico, 173,00144, Roma, Italy, October 1988.

IBM (1990), "Optimisation Subroutine Library: Guide and Reference", First Edition, April 1990, IBM Corporation, Neighbourhood Road, Kingston, New York, U.S.A. 12401.

Imai, M., Fukumura, T. and Yoshida, Y. (1979), "A parallelised branch and bound algorithm - implementation and efficiency", Systems Computers Controls 10(3), pp 62-70.

Inmos Ltd. (1985), "IMSt414 Transputer Reference Manual" (Bristol:INMOS Ltd).

Inmos Ltd. (1988), "Communicating Process Architecture", Prentice-Hall, p 84.

Inmos Ltd. (1991), "The T9000 Transputer Products Overview Manual", First Edition, SGS-Thomson Microelectronics, 1991, p 35.

Jordan, H.F. (1978), "A special purpose architecture for finite element analysis", IEEE Proc. 1978 Int. Conf. on Parallel Processing, pp 263-266 (London:IEEE).

Karp, R.M. (1972), "Reducibility among Combinatorial Problems", in "Complexity of Computer Computations" (eds. R.E. Miller and J.W. Thatcher), Plenum Press, New York, pp 85-103.

Khachiyan, L.G. (1979), "A Polynomial Algorithm in Linear Programming", Soviet Mathematics Doklady 20, pp 191-194.

Kindervater, G.A.P. and Lenstra, J.K. (1986), "Parallel computing in combinatorial optimisation", Report OS-R8614, Centre for Mathematics and Computer Science, Amsterdam.

Kindervater, G.A.P. and Trienekens, H.W.J.M. (1985), "Experiments with parallel algorithms for combinatorial problems", Report OS-R85 12, Centre for Mathematics and Computer Science, Amsterdam, 1985.

Klee, V. and Minty, G.J. (1972), "How Good is the Simplex Algorithm?", in Inequalities III, (ed. O. Shisha), Academic Press, New York, pp 159-175.

Kuck, D.J. "A survey of parallel machine organisation and programming", Comput. Surv. 9 (1977), pp 29-59.

Kung, H.T. and Leiserson, C.E. (1979), "Systolic arrays (for VLSI)", in Sparse Matrix Proceedings 1978 (eds. I.S. Duff and G.W. Stewart), SIAM, pp 256-282.

Kung, H.T. (1984), "Systolic algorithms for the CMU Warp processor", Proc. 7th Int. Conf. on Pattern Recognition, Montreal, pp 570-577.

Lai, T-H. and Sahni, S. (1984), "Anomalies in parallel branch and bound algorithms", Communications of the ACM 27(6), pp 594-602.

Land, A.H. and Doig, A.G. (1960), "An automatic method for solving discrete programming problems", Econometrica 28, pp 497-520.

Lavalée, I. and Roucairol, C. (1985), "Parallel branch and bound algorithms", presented at Euro VIII Congress, Bologna, Italy, Report MASI no.112, Univ. Paris VI, 1985.

Lemke, C.E. (1954), "The dual method of solving the linear programming problem", Naval Research Logistics Quarterly 1 (1954), pp 36-47.

Lemke, C.E. and Spielberg, K. (1967), "Direct Search Algorithm for Zero-One and Mixed Integer Programming", Operations Research 15(5), pp 892-914.

Li, G-J. and Wah, B.W. (1984), "Computational efficiency of parallel approximate branch and bound algorithms", in Proceedings of the 1984 International Conference on Parallel Processing, pp 473-480.

Li, G-J. and Wah, B.W. (1986), "Coping with anomalies in parallel branch and bound algorithms", IEEE Trans. Comput. 35(6), pp 568-573.

Lovett, T. and Thakkar, S. (1988), "The Symmetry multiprocessor system", Proc. 1988 Int. Conf. of Parallel Processing, University Park, Pennsylvania, pp 303-310.

Ma, R.P., Tsung, F-S. and Ma, M-H. (1988), "A dynamic load balancer for a parallel branch and bound algorithm", in Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications - Vol II, ACM Press, 1988, pp 1505-1513.

Manne, A.S. (1971), "A Mixed Integer Algorithm for Project Evaluation", Memorandum 71-3 (February 1971), Development Research Center, International Bank for Reconstruction and Development, Washington,D.C.

Maples, C., Rathbun, W., Weaver, D. and Meng, J. (1981), "The design of MIDAS - a Modular Interactive Data Analysis System", IEEE Trans. Nucl. Sci.,28, pp 3746-3753.

McCanny, J.V. and McWhirter, J.G. (1982), "On the implementation of signal processing functions using one-bit systolic arrays", Electron. Lett. 18, pp 241-243.

McKeown, G.P., Rayward-Smith, V.J., Rush, S.A. and Turpin, H.J. (1990), "Using a transputer network to solve branch and bound problems", presented at O.R. Society Conference 1990, University of Wales, Bangor, (School of Information Systems, University of East Anglia, Norwich).

Microway Ltd (1988), "Microway NDP FORTRAN Compiler Reference, V1.4VM", Mass., U.S.A.

Miller, P.C., John, C.E.S. and Hawkinson, S.W. (1988), "FPS T series parallel computer", in Programming Parallel Processors, Reading, MA: Addison-Wesley, 1988, pp 73-91.

Miranker, G.S., Rubenstein, J. and Sanguinetti, J. (1988), "Squeezing a Cray-class supercomputer into a single-user package", COMPCON, IEEE, March 1988, pp 452-456.

Mohan, J. (1982), "A study in parallel computation: the travelling salesman problem", Technical Report CMU-CS-82-136, Computer Science Department, Carnegie-Mellon University.

Mohan, J. (1983), "Experience with two parallel programs solving the travelling salesman problem", in Proceedings of the 1983 International Conference on Parallel Processing, IEEE, New York, pp 191-193.

Moore, W., McCabe, A. and Urquhart, R. (1987), "Systolic Arrays", Bristol: Adam Hilger.

Morrison, P. and Morrison, E. (1961) "Charles Babbage and his calculating engines",New York:Dover, p 34.

Motegi, M., Uchida, K. and Tsuchimoto, T. (1984), "The architecture of the FACOM vector processor", in "Parallel Computing 83", (eds. M. Feilmeier, J. Joubert and U. Schendel), Amsterdam:Elsevier, North-Holland, pp 541-546.

Mraz, R. and Seward, W. (1987), "Performance evaluation of parallel branch and bound search with the Intel iPSC hypercube computer", Proceedings of Supercomputing 88, Boston, pp 82-91.

Nagashima, S., Inagami, Y., Odaka, T. and Kawabe, S. (1984), "Design considerations for a high speed vector processor: the HITACHI S-810", Proc. Int. Conf. CD (MD:IEEE).

Nau, D.S., Kumar, V. and Karal, L. (1984), "General branch and bound and its relation to A* and AO*", Artificial Intelligence 23, pp 29-58.

Nemhauser, G.L. and Widhelm, W.B. (1971), "A Modified Linear Program for Columnar Methods in Mathematical Programming", Operations Research 19, pp 1051-1060.

Nemhauser G.L. and Wolsey L.A. (1988), "Integer and Combinatorial Optimisation", Wiley-Interscience Series in Discrete Mathematics and Optimisation, New York, 355-367.

Padberg, M.W. and Hong, S. (1980), "On the Symmetric Travelling Salesman Problem: A Computational Study", Mathematical Programming Study 12, pp 78-107.

Pardalos, P.M. and Rodgers, G.P. (1990), "Parallel branch and bound algorithms for quadratic zero-one programs on the hypercube architecture", Annals of Operations Research 22(1990), pp 271-292.

Pase, D.M. and Larrabee, A.R. (1988), "Intel iPSC concurrent computer", in Programming Parallel Processors, Reading, MA: Addison-Wesley, 1988, pp 105-124.

Patry, P., Salome, J. and Kuchler, P. (1987), "Optical character recognition on a network of transputers", in "Parallel Programming of Transputer Based Machines" (ed. T. Muntean), IOS publishing, Amsterdam, 1988,pp 433-448.

Pease, M.C. (1977), "The indirect binary n-cube microprocessor array", IEEE Trans. Comput. C-26, pp 458-473.

Peyton-Jones, S.L. (1987), "The implementation of functional languages", Englewood Cliffs, New Jersey, Prentice-Hall.

Pfister,G.F., Brantley, W.C., George, D.A., Harvey, S.L., Kleinfekder, W.J., McAuliffe, K.P., Melton, E.A., Norton, V.A. and Weiss, J. (1985), "The IBM research parallel processor prototype (RP3): Introduction and Architecture", Proc. 12th Int. Symposium on Computer Architecture, June 1985, Boston, Mass., pp 764-771.

Pham, D.T., Hu, H. and Pote, J. (1990), "A transputer-based system for locating parts and controlling an industrial robot", SERC/DTI Transputer Initiative Mailshot, November 1990, pp 53-59.

Preparata, F.P. and Vuillemin, J. (1981), "The cube connected cycles: a versatile network for parallel computation", Commun. ACM, 24, pp 300-309.

Pruul, E.A., Nemhauser, G.L. and Rushmeier, R.A. (1988), "Branch and bound and parallel computation: a (sic) historical note", Operations Research Letters 7(2), pp 65-69.

Quinn, M.J. (1986), "Implementing best-first branch and bound algorithms on hypercube multicomputers", Technical Report PCL-86-02, Dept. of Computer Science, University of New Hampshire, Durham, New Hampshire 03824.

Quinton, P. (1987), "An Introduction to Systolic Architectures", Lecture Notes in Computer Science, 272, pp 387-401.

Reddaway, S.F. (1973), "DAP - a distributed array processor", First Annual Symp. on Computer Architecture, Florida, pp 61-65.

Roucairol, C. (1986), "Experiments with parallel algorithms for the asymmetric salesman problem", presented at EURO VIII, Lisbon, Portugal.

Roucairol, C. (1987), "A parallel branch and bound algorithm for the quadratic assignment problem", Discrete Applied Mathematics 18(1987), pp 211-255.

Roucairol, C. (1989), "Parallel branch and bound algorithms - an overview", in Parallel and Distributed Algorithms (eds. M. Cosnard et al.), Elsevier Science Publishers B.V. , North-Holland, 1989, pp 153-163.

Russell, R.M. (1978), "The CRAY-1 Computer System", Comm. of the ACM, 21(1), January 1978, pp 63-72.

Schnabel, R.B. (1984), "Parallel computing in Optimisation", in Proceedings of the NATO Advanced Study Institute on Computational Mathematical Programming, Bad Windsheim, Germany F.R., July 1984, pp 358-381.

Schwartz, J.T. (1980), "Ultracomputers", ACM Trans. Prog. Lang. Syst. 2, pp 484-521.

Seitz, C.L. (1985), "The Cosmic Cube", Commun. of the ACM, 28(1), pp 22-33.

Seitz, C.L., Athas, W.C., Dally, W.J. et al. (1988), "Message-
Passing Concurrent Computers: Their Architecture and Programming", Reading MA: Addison-Wesley.

Shapiro, J. (1968), "Group Theoretic Algorithms for the Integer Programming Problem II: Extension to a General Algorithm", Operations Research 16(5), pp 928-947.

Slotnick, D.L., Borck, W.C. and McReynolds, R.C. (1962), "The SOLOMON computer", AFIPS Conf. Proc. 22, pp 97-107.

Smith, B.J. (1978), "A pipelined shared resource MIMD computer", IEEE Proc. 1978 Int. Conf. on Parallel Processing, pp 6-8.

Sporer, M., Moss, F.H. and Mathais, C.J. (1988), "An introduction to the architecture of the Stellar Graphics supercomputer", COMPCON, IEEE, March 1988, pp 464-467.

Sridharan, R. (1991), "A Lagrangian heuristic for the capacitated plant location problem with side constraints", JORS 42(7), pp 579-586.

Sternberg, S.R. (1985), "An overview of image algebra and related architectures", in "Integrated Technology for Parallel Image Processing" (ed. S. Levialdi), Academic Press, London, pp 79-100.

Stewart Jr., W.R. and Golden, B.L. (1984), "A Lagrangian relaxation heuristic for vehicle routing", EJOR 15(1), pp 84-88.

Sullivan, H., Bashkow, T.R. and Klappholz (1977), "A large scale homogeneous, fully distributed parallel machine", Fourth Symposium on Computer Architecture, March 1977, pp 105-124.

Swan, R.J., Fuller, S.H. and Siewiorek, D.P. (1977), "Cm*: a modular multi-microprocessor", Proc. National Computer Conference 46, pp 637-644.

te Riele, H.J.J., Dekker, Th.J. and van der Vorst, H.A. (1987), "Algorithms and Applications on Vector and Parallel Computers", Amsterdam: North-Holland.

Trienekens, H.W.J.M. (1986), "Parallel branch and bound on a MIMD system", Report 8640/A, Econometric Institute, Erasmus University, Rotterdam.

Villano, U. (1990), "A Microcomputer Architecture for High-Speed Mono- and Bi-dimensional FFT computation", submitted to IEEE Trans. on Parallel and Distributed Syst., July 1990.

Vornberger, O. (1988), "Load balancing in a network of transputers", Lecture Notes in Computer Science 312, Distributed Algorithms, Springer-Verlag, pp 116-126.

Wah, B.W. and Ma, Y.W.E. (1984), "MANIP - a multicomputer architecture for solving combinatorial extremum-search problems", IEEE Trans. Comput., 33(5), pp 377-390.

Watanabe, T. (1984), "Architecture of supercomputers - NEC SX system", NEC Res. Dev. 73, pp 1-6.

Watanabe, T. (1987), "Architecture and Performance of the NEC Supercomputer SX System", Parallel Computing, 5, pp 247-255.

Watson, W.J. (1972), "The TI ASC - A highly modular and flexible supercomputer architecture", Proc. AFIPS Fall Joint Computer Conf., pp 221-228.

Whitby-Strevens, C. (1985), "The Transputer", Proc. 12th Int. Symposium on Computer Architecture, Boston, Mass., June 1985, pp 292-300.

White, W. (1966), "On a Group Theoretic Approach to Linear Integer Programming", Operations Research Center Report 66-27, the University of California at Berkeley, 1966.

Williams H.P. (1978), "Model building in mathematical programming", Wiley, New York, 1978.

Wilson , A.W. Jr. (1987), "Hierarchical cache/bus architecture for shared memory multiprocessors", Proc. 14th Int. Symposium on Computer Architecture, June 1987, Pittsburg, Penn., pp 244-252.

Wulf, W.A. and Bell, C.G. (1972), "C.mmp: a multi-mini-processor", Proc. AFIPS Fall Joint Computer Conference 41 (2), pp 765-777.

Wulf, W.A., Levin, R. and Harbison, S.P. (1981), "HYDRA/C.mmp: An Experimental Computer System", New York NY: McGraw-Hill, 1981, p 277.

Young, R.D. (1965), "A Primal (All Integer), Integer Programming Algorithm", Journal of Research of the National Bureau of Standards 69B, pp 213-250.

3L Limited (1988), Parallel FORTRAN User Guide, Peel House, Ladywell, Livingston, Scotland.

Bibliography

deCarlini, U. and U. Villano, "Transputers and Parallel Architectures: message-passing distributed systems", Ellis Horwood Ltd., Chichester, 1991.

Fountain,T., "Processor Arrays: Architecture and Applications", Academic Press, London, 1987.

Galletly, J., "Occam 2", Pitman Publishing, 1990.

Garey, M.R. and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, San Francisco, 1979.

Garfinkel, R.S. and G.L. Nemhauser, "Integer Programming", John Wiley, London, 1972.

Hockney, R.W. and C.R. Jessope, "Parallel Computers 2: Architecture, Programming and Algorithms", Adam Hilger (IOP Publishing), Bristol, U.K., second edition 1988.

Lewis, T.G. and H. El-Rewini, "Introduction to Parallel Computing", Prentice-Hall International, 1992.

Nemhauser, G.A. and L.A. Wolsey, "Integer and Combinatorial Optimization", John Wiley, New York, 1988.

Salkin, H.M., "Integer Programming", Addison-Wesley, Reading, Massachusetts, 1975.

Taha, H.A., "Operations Research: An Introduction", Macmillan Publishing Company, New York, Fifth Edition, 1992.

Where sources are referred to in the text, this is indicated by the use of {}, i.e.

"... see {Nemhauser and Wolsey, page 254} for further details"

Appendix 1: Further Details Relating to the MIP-Solving Algorithms Discussed in Chapter Two.

Appendix 1A: Derivation of the Gomory Cut for PIP Problems

To define a Gomory cut and show that it is appropriate for the purpose of reducing the solution space of PIP problems without removing any integer feasible solutions, consider the following, adapted from (Garfinkel and Nemhauser, pp 157-158):

The optimal tableau for a PIP problem may be represented by the formulation

$$x_{Bi} + \sum_{j \in R} a_{ij} x_j = b_i \qquad \forall i = 0, 1, 2, \ldots, m \qquad (1)$$

$x_j \geq 0$ and integer

where

R	= set of non-basic variables
x_{Bi}	= basic variable for row i
x_j	= non-basic variables
a_{ij}	= coefficient of variable x_{ij} in row i
b_i	= right hand side constant.

Since $x_{ij} \geq 0$ by definition,

$$x_{Bi} + \sum_{j \in R} [a_{ij}]\, x_j \leq b_i \qquad (2)$$

($[a_{ij}]$ is the integer part of a_{ij}).

Also, since all the x variables must be integers, the left hand side of (2) must be integer, therefore

$$x_{Bi} + \sum_{j \in R} [a_{ij}]\, x_j \leq [b_i] \qquad (3)$$

(1)-(3) gives

$$\sum_{j\in R}(a_{ij}-[a_{ij}])x_j \geq b_i-[b_i] \qquad (4)$$

If we now define

$b_i = [b_i] + g_i$ and $a_{ij} = [a_{ij}] + f_{ij}$

then (4) becomes

$$\sum_{j\in R} f_{ij}x_j \geq g_i \qquad (5)$$

This can be written as

$$\sum_{j\in R} f_{ij}x_j = g_i + s \qquad (6)$$

where $s \geq 0$ is a slack variable for the new constraint.

This is the Gomory Cut.

Note that if $g_i > 0$ then (5) is violated, since $x_j = 0$ if j is a member of set R. Thus the cut can be used to exclude solutions where $g_i > 0$ (i.e. where basic variables take non-integer values). No integer feasible solution will be omitted by using the cut, since in its derivation, the integrality of the x variables is assumed.

Appendix 1B: Derivation of the Gomory Cut for MIP Problems

The Gomory Mixed Integer Cut is derived as follows:

Let	R_1	=	$\{j \mid j$ indexes the non-basics that are integers$\}$;
	R_2	=	$\{j \mid j$ indexes the non-basics that are continuous$\}$;
	x_{Bi}	=	basic variable for row i;
	x_j	=	non-basic integer variables;
	y_j	=	non-basic continuous variables;
	a_{ij}	=	coefficient of integer variable x_j in row i;
	e_{ij}	=	coefficient of continuous variable y_j in row i; and
	b_i	=	right hand side constant for row i.

Then the equation (1) from the PIP case may be rewritten as:

$$x_{Bi} + \sum_{j \in R_1} a_{ij} x_j + \sum_{j \in R_2} e_{ij} y_j = b_i \qquad \forall\, i = 0,1,2,\ldots,m \qquad (7)$$

Since, as previously defined,

$b_i = [b_i] + g_i$ and $a_{ij} = [a_{ij}] + f_{ij}$

(7) can be rewritten as

$$x_{Bi} + \sum_{j \in R_1} [a_{ij}] x_j + \sum_{j \in R_1} f_{ij} x_j + \sum_{j \in R_2} e_{ij} y_j = [b_i] + g_i \qquad (8)$$

For x_{Bi} to be integer, either

$$x_{Bi} \le [b_i] - \sum_{j \in R_1} [a_{ij}] x_j ,$$

or

$$x_{Bi} \ge [b_i] - \sum_{j \in R_1} [a_{ij}] x_j + 1$$

Thus, substituting in (8), either

$$\sum_{j\in R_1} f_{ij}x_j + \sum_{j\in R_2} e_{ij}y_j \geq g_i \qquad (9)$$

or

$$\sum_{j\in R_1} f_{ij}x_j + \sum_{j\in R_2} e_{ij}y_j \leq g_i - 1 \qquad (10)$$

Now, let R_2' = {j|j is a member of set R_2 , $e_{ij} > 0$ }; and
R_2 = {j|j is a member of set R_2 , $e_{ij} < 0$ }.

Then, if (9) is true, so is

$$\sum_{j\in R_1} f_{ij}x_j + \sum_{j\in R_2'} e_{ij}y_j \geq g_i \qquad (11)$$

Since, by definition, $f_{ij} \geq 0$ for all j in R_1, if (10) is true, so is

$$\sum_{j\in R_2} e_{ij}y_j \leq g_i - 1$$

and hence, so is

$$\sum_{j\in R_2} e_{ij}y_j \leq g_i - 1 \qquad (12)$$

Multiplying (12) by $g_i/(g_i-1)$, which is negative, gives

$$-\sum_{j\in R_2} \frac{g_i e_{ij}y_j}{(1 - g_i)} \geq g_i \qquad (13)$$

Since (11) and (13) are mutually exclusive, a joint condition can be expressed as follows:

$$\sum_{j \in R_1} f_{ij} x_j + \sum_{j \in R_2^+} e_{ij} y_j - \sum_{j \in R_2^-} \frac{g_i e_{ij} y_j}{(1 - g_i)} \geq g_i \qquad (14)$$

This is the Gomory Cut for MIP problems.

(14) can be improved if there exist members j of the set R_1 such that $f_{ij} > g_i$.

If so, let $R_1^+ = \{j | j$ is a member of set R_1, $f_{ij} \leq g_i\}$; and

$R_1^- = \{j | j$ is a member of set R_1 , $f_{ij} > g_i\}$.

Subtracting

$$\sum_{j \in R_1^-} x_j$$

from both sides of equation (8) gives

$$x_{Bi} + \sum_{j \in R_1^+} [a_{ij}] x_j + \sum_{j \in R_1^-} ([a_{ij}] - 1) x_j + \sum_{j \in R_1^+} f_{ij} x_j$$

$$+ \sum_{j \in R_1^-} (f_{ij} - 1) x_j + \sum_{j \in R_2} e_{ij} y_j = [b_i] + g_i \qquad (15)$$

Now, in order for x_{Bi} to be integer, either

$$\sum_{j \in R_1^+} f_{ij} x_j + \sum_{j \in R_1^-} (f_{ij} - 1) x_j + \sum_{j \in R_2} e_{ij} y_j \geq g_i \qquad (16)$$

or

$$\sum_{j \in R_1^+} f_{ij} x_j + \sum_{j \in R_1^-} (f_{ij} - 1) x_j + \sum_{j \in R_2} e_{ij} y_j \leq g_i - 1 \qquad (17)$$

Since $(f_{ij} - 1)$ will always be negative for j in R_1^- and e_{ij} will always be negative for j in R_2^-, if (16) is true, so is

$$\sum_{j \in R_1^+} f_{ij} x_j + \sum_{j \in R_2^+} e_{ij} y_j \geq g_i \qquad (18)$$

Similarly, if (17) is true, so is

$$\sum_{j \in R_1^-} (f_{ij} - 1) x_j + \sum_{j \in R_2^-} e_{ij} y_j \leq g_i - 1 \qquad (19)$$

Multiplying (19) by $g_i/(g_i-1)$, which is negative, gives

$$-\frac{g_i}{(1 - g_i)} \left(\sum_{j \in R_1^-} (f_{ij} - 1) x_j + \sum_{j \in R_2^-} e_{ij} y_j \right) \geq g_i \qquad (20)$$

Thus, since (18) and (20) are mutually exclusive, a joint condition can be expressed as follows:

$$\sum_{j \in R_1^+} f_{ij} x_j + \sum_{j \in R_1^-} \frac{g_i (1 - f_{ij}) x_j}{(1 - g_i)} + \sum_{j \in R_2^+} e_{ij} y_j - \sum_{j \in R_2^-} \frac{g_i e_{ij} y_j}{(1 - g_i)} \geq g_i$$

This is the tightest Gomory Cut for MIP problems.

[illegible] 1C: [illegible] of [illegible]' Reformulation for MIP Problems

Benders' algorithm allows the solution of an MIP by rewriting it as a PIP.

Consider the following, adapted from (Garfinkel and Nemhauser, pp 135-143):

Consider the MIP problem

$$
\begin{aligned}
&x_0^* = \text{MAX } x_0 = cx + dy \\
&Ax + By \le b \qquad\qquad (1) \\
&x \ge 0 \text{ and integer, } y \ge 0
\end{aligned}
$$

For any non-negative value, x_{val}, of the integer vector x, the MIP problem (1) reduces to the LP (2):

$$
\begin{aligned}
&x_0^*(x_{val}) = cx_{val} + \text{MAX } dy \\
&By \le b - Ax_{val} \qquad\qquad (2) \\
&y \ge 0
\end{aligned}
$$

The dual of LP (2) is:

$$
\begin{aligned}
&u_0^*(x_{val}) = cx_{val} + \text{MIN } u(b - Ax_{val}) \\
&uB \ge d \qquad\qquad (3) \\
&u \ge 0
\end{aligned}
$$

Looking at the solution to the dual LP (3) provides insights into the choice of x_{val} necessary to give an optimal solution to MIP problem (1). By LP duality theory, if MIP (1) is to have an optimal solution, the dual LP (3) must not be infeasible or unbounded. Since the dual constraints are completely independent of x_{val}, if the dual LP (3) is infeasible, then the MIP problem (1) is either infeasible or unbounded. If the dual LP (3) is unbounded, this implies that its objective function is decreasing along some extreme ray (see Fig A1.1 below). Thus, there exists an extreme point u^m and a direction v such that every point on the extreme ray

$u^m + \theta v \qquad (\theta \ge 0)$

is feasible to the dual LP (3).

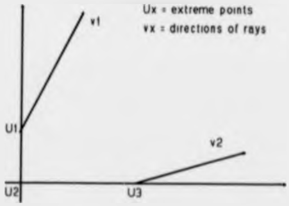


Fig. A1.1: The case where the Dual LP (3) is unbounded.

Thus, when the dual LP (3) is unbounded, its objective function may be written in terms of the extreme ray as:

$u_0 = (u^t + \theta v)(b - Ax_{m+1})$

where $T = \{u^t | u^t$ is an extreme point of dual LP (3)$\}$ and where u_0 decreases with θ.

Removing constant terms, this can be rewritten as

$u_0 = \theta v(b - Ax_{m+1})$, which decreases with θ.

Since $\theta \geq 0$, then $v(b - Ax_{m+1}) < 0$ and so if Q is a set of directions such that

$Q = \{v^q | u^t + \theta v^q, \theta \geq 0$ is an extreme ray for some $u^t \in T\}$

it is sufficient to impose the constraint

$v^q(b - Ax) \geq 0$ for every $v^q \in Q$

in order to rule out x_{m+1} candidates which will cause the dual LP (3) to become unbounded. Since this is the only case that we are interested in, we may rewrite the dual LP (3) as

$$u_0^*(x_{val}) = cx_{val} + \underset{u^t \epsilon T}{MIN} u^t(b - Ax)$$

$$v^q(b - Ax) \geq 0 \quad for\ every\ v^q \epsilon Q.$$

Since $x_0^* = MAX\ x_0^*(x)$, $x \geq 0$, integer and admissible, it can also be written as $x_0^* = MAX\ u_0^*(x)$, $x \geq 0$, integer and admissible.

That is:

$$x_0^* = MAX(cx + \underset{u^t \epsilon T}{MIN} u^t(b - Ax))$$

$$v^q(b - Ax) \geq 0 \quad for\ every\ v^q \epsilon Q.$$

Introducing a variable

$$Z = cx + \underset{u^t \epsilon T}{MIN} u^t(b - Ax)$$

this can be written as

MAX $\quad Z$

$Z \leq cx + u^t(b - Ax)$ for every $u^t \in T$

$0 \leq v^q(b - Ax)$ for every $v^q \in Q$ (4)

$x \geq 0$ and integer

This is Benders' reformulation of the original MIP problem (1).

[illegible]

Gomory showed that by relaxing the non-negativity (bu not integrality) constraints on certain variables, any IP can be represented by a minimisation problem defined as a group.

Consider the following, adapted from (Salkin, pp 282-284):

Define the standard representation of a PIP problem as

Maximise	cx	
Subject to	$Ax = b$	(1)
and	$x \geq 0$ and integer	

We shall rewrite this problem as

Maximise	$c_B x_B + c_N x_N$	
subject to	$Bx_B + Nx_N = b$	(2)
and	$x_B \geq 0$, $x_N \geq 0$ and integer	

where B is a basis whose columns are from A, N are the remaining non-basic columns from A and the terms are rearranged so that x_B are the basic variables associated with B and x_N are the non-basic variables associated with N. The costs corresponding to the basic and non-basic variables are c_B and c_N respectively.

Since B is a basis, it has an inverse, and thus we may solve the constraints for (2) to get

$$x_B = B^{-1}b - B^{-1}Nx_N \qquad (3)$$

Thus, problem (2) may be rewritten as

Maximise	$c_B B^{-1}b - (c_B B^{-1}N - c_N)x_N$	
Subject to	$x_B = B^{-1}b - B^{-1}Nx_N$	(4)
and	$x_B \geq 0$ and integer,	
	$x_N \geq 0$ and integer	

Since the term $c_B B^{-1}b$ is a constant, it may be dropped from the objective function. Also, constraining x_B to be an integer is equivalent to stating that $x_B \equiv 0$ modulo 1. Thus, from (4)

$B^{-1}Nx_N \equiv B^{-1}b$ (mod 1) (5)

Thus, the problem may again be rewritten as

Minimise	$(c_BB^{-1}N - c_N)x_N$		
subject to	$B^{-1}Nx_N \equiv B^{-1}b$	(mod 1)	(6)
and	$x_N \geq 0$ and integer		
	$x_B = B^{-1}b - B^{-1}Nx_N \geq 0$		

If x_N satisfies (5), it yields an integral x_B which satisfies (2) and vice versa. Thus, the original problem (1) may be solved by solving (6).

Note that the costs associated with variable x_N in (6) are of the form $(c_BB^{-1}a_j - c_j$ where a_j is some column from A in N and c_j is the corresponding cost.

Taking the dual of the LP of (1) gives

Minimise	wb
subject to	wA = c
and	$w \geq 0$

The value of the jth surplus variable w_{sj} in wA = c is

$w_{sj} = wa_j - c_j = c_BB^{-1}a_j - c_j$ when $w = c_BB^{-1}$

Thus the costs associated with variable x_N in (6) are just the reduced costs or the values of the dual surplus variables.

If the basis B yields a dual feasible solution, i.e. if $c_BB^{-1}N - c_N \geq 0$ and the non-negativity constraints on x_B are dropped, then the problem (6) can be rewritten as

Minimise	cx_N		
subject to	$B^{-1}Nx_N \equiv B^{-1}b$	(mod 1)	(7)
and	$x_N \geq 0$ and integer		

since $B^{-1}Nx_N$ is equivalent to

$$\sum_{j=1}^{n} \alpha_j x_{J(j)}$$

where α_j is the jth column of $B^{-1}N$ and $x_{J(j)}$ is the jth ($1 \leq j \leq n$) non-basic variable.

Let $B^{-1}b$ be denoted by the column α_0, then the congruence relationship becomes

$$\sum_{j=1}^{n} \alpha_j x_{J(j)} \equiv \alpha_0 \qquad (\text{mod}\, 1) \qquad (8)$$

Since two vectors are congruent (modulo 1) if and only if the corresponding elements are congruent (modulo 1), there are actually m congruence relations in (8), we may add or subtract multiples of $x_{J(j)} \equiv 0$ (mod 1) to each equation without destroying the congruence relationship so that every column α_j has non-negative entries less than one. Similarly, the elements in α_0 may be reduced to non-negative fractions by adding or subtracting multiples of $0 \equiv 1$ (mod 1) to each component.

Let the columns that have the fractional parts be denoted as $\hat{\alpha}_j$ $(j=0,1,\ldots,n)$.

Thus, the problem may again be rewritten as

$$\textit{Minimise} \qquad \sum_{j=1}^{n} \sigma_j x_{J(j)}$$

$$\textit{subject to} \qquad \sum_{j=1}^{n} \hat{\alpha}_j x_{J(j)} \equiv \hat{\alpha}_0 \qquad (\text{mod}\, 1) \qquad (9)$$

and $x_{J(j)} \geq 0$ and integer $(j=1,\ldots\ldots,n)$

where $\sigma_j \geq 0$ are the costs and each column $\hat{\alpha}_j$ $(j=0,1,\ldots\ldots,n)$ satisfies $0 \leq \hat{\alpha}_j < e$ (where e is a column of ones).

Problem (9) is referred to as the **Group Minimisation Problem**

Appendix 1E: Implicit Enumeration Criteria for the Direct Search Algorithm

Before discussing some of the implicit enumeration criteria appearing in the literature on Direct Search Enumeration, some further terms must be defined in addition to those defined in section 2.4.2.

The subproblem at point x^l is problem P

Minimise $\quad z = cx^l$

Subject to $\quad Ax^l \leq b$

and $\quad 0 \leq x^l \leq e$

where $x_j = 0$ or 1 for $j=1,\ldots,n$ and e is a column of ones.

If the o variables set at zero are dropped and the l columns whose variables are fixed at one are subtracted from the right hand side vector b, then P can be rewritten as

Minimise $\quad z - z^l = c^l x^l$

Subject to $\quad A^l x^l \leq b^l$

and $\quad 0 \leq x^l \leq e$

where $x_j = 0$ or 1 for $j \in F$.

$x^l = (x_j)$ is the vector corresponding to free variables, F is the corresponding set of indices in x^l, c^l and $A^l = (A_j)$ are costs and columns of A, b^l is the updated right hand side, and z^l is the sum of the costs of the l variables fixed to one.

This subproblem is named P^l. The associated LP problem is named LP^l and its optimal solution is $\hat{z}$ (which includes the constant z^l). The present best integer solution is z^*.

Some of the implicit enumeration criteria of the direct search algorithm are as follows:

__Ceiling Tests__

The objective function value at node x^l is z^l. This value can be decreased by at most the sum (over the free variables) of the negative costs. Thus, an improved integer solution (since we are dealing with minimisation problems) found from node x^l is only possible if

$$z^l + \sum_{j \in F^-} c_j \leq z^* \qquad (1)$$

where $F^- = \{j \in F \mid c_j < 0\}$.

This means that a free variable with a positive cost should not be set to one if it increases s^l so that (1) cannot be satisfied. Thus, cancel, at level l, any $j \in F$ for which

$$c_j + z^l + \sum_{j \in F^-} c_j \ge z^* \qquad (2)$$

<u>Infeasibility Test</u>

The constraints of the subproblem are of the form

$s^l = b^l - A^l x^l \ge 0$ (3)

where $s^l = (s_i)$ are non-negative slack variables.

Since $x_j \le 1$, the largest possible value for slack s_i is

$$P_i = b_i^l - \sum_{j \in F_-} a_{ij} \qquad (4)$$

where $F_- = \{j \in F \mid a_{ij} < 0\}$.

Therefore, for a zero-one solution $P_i \ge 0$ for $i=1,\dots,m$ (5)
From (4), $P_i \ge b_i^l$ and thus for $b_i^l \ge 0$, (5) is automatically satisfied. Using (4), we can find a P_i for each constraint which has $b_i^l < 0$. If any of these P_i values is negative, a backwards step is justified.

<u>Cancellation Test [illegible]</u>

If a free variable has a large positive coefficient in some row, then setting this variable to one reduces b_i^l and may result in a $P_i < 0$. Therefore, if, for any $j \in F$,

$$P_i - \max_{a_{ij}>0} a_{ij} < 0 \quad for\ some\ i \quad (1 \le i \le m) \qquad (6)$$

cancel x_j at level l.

Note that if (6) indicates a cancellation, a column a_j is omitted from A^l and so the values of P_i (as defined by (4)) may be changed, and (5) and (6) retested.

__Cancellation One Test__

If one of the negative coefficients of free variables was omitted when computing P_i (i.e. a free variable was temporarily set to zero), this __could__ result in $P_i < 0$. If this happens, then for a zero-one solution to be possible, the omitted variable must take the value one.

Therefore if, for any $j \in F$,

$$P_i + \underset{a_{ij}<0}{MIN}\, a_{ij} < 0 \quad \textit{for some } i \quad (1 \leq i \leq m) \qquad (7)$$

then x_j must have value one in any zero-one solution produced from node x^t. If another test results in cancelling x_j, a backwards step is justified.

If $b_i^t \geq 0$, observe that $P_i + a_{ij} \geq b_i^t \geq 0$ for any $a_{ij} < 0$, or (7) would never indicate that a variable must have value one.

__Linear Programming Tests__

Obvious points relating to the LP relaxation of the subproblem are:

(1) if any LP solution LP^t is integer (in the free x variables), this solution is optimal for P^t).

(2) if LP^t has no feasible solution, there is no zero-one solution to P^t.

(3) the optimum value of the objective function $\hat{z}$ for the LP^t is a lower bound on the value z for __any__ zero-one solution to the subproblem (i.e. $\hat{z} \leq z$).

(4) if, at any dual simplex iteration, the value of the LP objective function z exceeds the current best integer solution z^*, a backwards step is allowed.

Appendix 1F: Derivation of Constraints for the Algorithm for Zero-One MIP Problems

The Lemke and Spielberg Direct Search Algorithm for zero-one MIPs derives constraints that are valid at any node using only the zero-one variables.

Consider the MIP problem

Minimise	$z = cx + dy$	
Subject to	$Ax + By \le b$	(1)
and	$0 \le x \le 1$ and integer, $y \ge 0$	

For any fixed zero-one vector x, the MIP problem (1) reduces to the LP (2):

Minimise	$z - cx = dy$	
Subject to	$By \le b - Ax$	(2)
and	$y \ge 0$	

The dual of LP (2) is the dual LP (3):

Minimise	$u(b - Ax)$	
Subject to	$uB + d \ge 0$	(3)
and	$u \ge 0$	

If (x,y) is to give an improved MIP solution, then $cx + dy < z'$ and $Ax + By \le b$. Multiplying the first equation by a non-negative vector u and adding it to the first yields

$(z' + ub) - (uB + d)y - (uA + c)x > 0$ (4)

At each node that is explicitly enumerated, an LP of form P is solved. The Simplex computations generate extreme points of the polyhedron U, where $U = \{u|uB + d \ge 0, u \ge 0\}$. (If U is empty then the MIP problem has no solution).

Since this means that $uB + d \ge 0$, for non-negative y we have
$(c + uA)x < z' + ub$ (5)

Therefore, whenever an LP of form P is solved, it yields dual extreme points u and constraints of type (5) in only the x variables. Since U is independent of x, these inequalities are valid at any node.

Since it is possible to have a large number of dual extreme points and thus generate a large number of constraints at each node, a good rule of thumb is to generate only the constraint coming from the optimal dual extreme point.

A necessary and sufficient condition on x to admit feasible solutions y to the MIP is that

$$v(b - Ax) \geq 0 \qquad (6)$$

where v is a direction of a ray in U.

This constraint is derived in a similar way to the constraint mentioned in the derivation of Benders' decomposition algorithm, as previously mentioned in section 2.2.1.

Appendix 10: Derivation of the [illegible] Penalties used to decide where to branch

If, when using the Branch and Bound method to solve a MIP problem, the solution to an LP relaxation indicates that branching is called for, a choice will usually have to be made as to which variable to branch upon. Beale describes the following method for choosing the branching variable ([Beale, 1979]).

An estimate of the degradation to the LP solution value caused by making each branch is made. The calculation of an estimated degradation is based on finding the amount by which the variable value will change and multiplying this by a per unit movement penalty.

If a supposedly-integer variable x_i take a continuous value C_i, where $C_i = |C_i| + f_i$, let imposing a new lower bound of $|C_i| + 1$ on x_i decrease the objective function by the 'up penalty' of p_i for every unit increase from the current value of x_i. (This is an 'up penalty' since the variable value must be moved up to the new lower bound). Similarly, let the 'down penalty' incurred by placing an upper bound of $|C_i|$ on x_i be p_i' for every unit decrease from the current value of x_i.

An estimated decrease to the LP solution of

$D_i' = p_i' f_i$

would occur if an upper bound was imposed and a decrease of

$D_i = p_i (1-f_i)$

would occur if a lower bound was imposed.

The values of p_i and p_i' are calculated as follows.

Write the problem constraints in terms of the variable that is to be branched upon, x_i, i.e. let the problem be formulated as

$$MAX \qquad \sum_{j \neq i} a_{0j} x_j = b_0 - a_{0i} x_i$$

$$\text{Subject to} \quad \sum_{j \neq i} a_{kj} x_j = b_k - a_{ki} x_i \qquad \forall \text{ rows } k > 0$$

where

x_j	= decision variables
a_{kj}	= coefficient of variable x_j in row k
b_k	= right hand side constant for row k

If the value of x_i is increased by $(1-f_i)$, there is no effect on the objective function value or on any of the other variables if the value of each b_k is simultaneously decreased in each row k by $z_k = a_{ki}(1-f_i)$. Similarly, if the value of x_i is decreased by f_i, there is no effect as long as each b_k is simultaneously decreased by $z_k = -a_{ki}f_i$.

Thus, to estimate the degradation to the LP solution imposed by changing the value of the variable x_i, we can hold the value of x_i constant and decrease b_k by z_k for all rows k.

Let π_k be the shadow price for the row k. Thus, if the movements z_k are small, the degradation to the LP solution imposed by branching could be estimated by

$$D = \sum_k \pi_k z_k \qquad (1)$$

i.e., p_i and p_i' are calculated by making use of the row coefficients for the variable x_i and the shadow prices π_k on the rows k.

It would be unwise, however, to simply use equation (1) to calculate the per unit degradation for the row, since the shadow price for a row only measures the cost of making very small changes and could thus greatly underestimate the costs of the larger changes that could occur. Equation (1) is thus only considered to give a lower bound on the value of the degradation. Indeed, if

x_i is a basic variable (the only situation where we would be considering branching), then the degradation will always be zero if equation (1) is used.

To get a more useful estimate of degradation, the XPRESS-MP optimiser ([Dash Associates, 1989]) adapts equation (1) to create the following heuristic method for estimating degradations to the LP solution.

Let the actual change made to the right hand side constant b_k of a row k (in order to model the effects of branching on a variable x_i) depend upon:
the type of the row; and
the sign of the coefficient of the branching variable x_i in that row.

Table A1.1 shows how the heuristic method models the effects of changes to the value of x_i for each row of the problem.

a_{ki} in row k:	Positive	Positive	Negative	Negative
Row k type:	'≤' row	'≥' row	'≤' row	'=' row
Increase x_i by $(1-f_i)$	Decrease b_k by $a_{ki}*(1-f_i)$	Increase b_k by $a_{ki}*(1-f_i)$	Increase b_k by $\|a_{ki}\|*(1-f_i)$	Decrease b_k by $\|a_{ki}\|*(1-f_i)$
Decrease x_i by f_i	Increase b_k by $a_{ki}*f_i$	Decrease b_k by $a_{ki}*f_i$	Decrease b_k by $\|a_{ki}\|*f_i$	Increase b_k by $\|a_{ki}\|*f_i$

Table A1.1. Modelling the change in the value of x_i.

N.B. When a problem has been input to the XPRESS-MP optimiser, any '≥' type rows are converted to "≤" type rows. Thus, the only types of rows that we are concerned with are '≤' or "=" type rows (along with the unconstrained objective function row).

Since, as mentioned above, the shadow price should only be used to measure the cost of making small changes to b_k, the XPRESS-MP optimiser heuristic also makes use of a 'pseudo-shadow-price' ps_k for each row k in order to measure the cost of making the changes described in Table A1.1.

The shadow price and pseudo-shadow-price for a row are used to construct a function

$cost_k$ = MIN(-ps_k,π_k) when a_{ki} is positive and x_i is being increased
                  or when a_{ki} is negative and x_i is being decreased;
      = MAX(ps_k,π_k) when a_{ki} is negative and x_i is being increased
                  or when a_{ki} is positive and x_i is being decreased.

The heuristic gives values to ps_k as follows in order to calculate appropriate values of $cost_k$.

If the value of x_i is to be increased in row k, ps_k is set to a positive, user-specified tolerance tol_k (default value 0.01) unless the row is unconstrained, in which case ps_k is set to zero (since changing the value of x_i will have no effect on an unconstrained row). If the value of x_i is to be decreased in row k, then ps_k is set to zero unless the row is of type "=", in which case it is set to a positive, user defined tolerance tol_k (default value 0.01).

The heuristic uses these settings for ps_k because the shadow prices for the "≤" type rows are allowed to be zero or positive, whilst the shadow prices for the "=" type rows are allowed to be positive, zero or negative.

The overall effect, as can be seen from Table A1.2, is to force $cost_k$ to be non-zero if possible.

a_{ki} in row k:	Positive	Positive	Negative	Negative
Row k type:	"≤" row	"=" row	"≤" row	"=" row
Unit Cost of increasing x_i (COSTUP row)	$MIN(-tol_k, \pi_k) * a_{ki}$	$MIN(-tol_k, \pi_k) * a_{ki}$	$MAX(tol_k, \pi_k) * a_{ki}$	$MAX(tol_k, \pi_k) * a_{ki}$
Unit cost of decreasing x_i (COSTDN row)	$MAX(0, \pi_k) * a_{ki}$	$MAX(tol_k, \pi_k) * a_{ki}$	$MIN(0, \pi_k) * a_{ki}$	$MIN(-tol_k, \pi_k) * a_{ki}$
Shadow price values	+ve or zero	-ve, zero, or +ve	+ve or zero	-ve, zero, or +ve

Table A1.2. Components of the per unit cost of changing the value of x_i.

The heuristic calculates p_i (the total per unit penalty for increasing x_i) by calculating the sum of the COSTUP row in Table A1.2 and p_i' (the total per unit penalty for decreasing x_i) by calculating the sum of the COSTDN row of Table A1.2.

Note that the value of p_i is calculated for use with the 'negative' distance f_i and is hence always negative, whereas the value of p_i' is calculated for use with the 'positive' distance $(1-f_i)$ and is hence always positive.

Per Unit Penalties for Special Ordered Sets

The heuristic method used to calculate the per unit penalties for branching on an integer-constrained variable is easily extended to calculating penalties for branching on Special Ordered Sets of Type One and Two (as defined in section 7.3).

When estimating the degradation to the LP solution caused by making a branch on a Special Ordered Set of Type One, a set member variable is chosen (as described in section 7.3.1) and its value altered when a branch is made on the set. This is a similar situation to that for branching on general integer variables and thus the same heuristic method can be used by the XPRESS-MP optimiser.

When estimating the degradation to the LP solution caused by making a branch on a Special Ordered Set of Type Two, two set member variables are chosen (as described in section 7.3.1) and their values altered when a branch is made on the set. Since the effects of changing the values of two variables can still be modelled by altering the right hand side of the constraints in which they occur, the situation is again similar to that faced when branching on general integers.

For both types of Special Ordered Set, the effects of altering the right hand side are modelled for each row. The component degradations caused by altering the individual rows are then summed to give the estimate of the total degradation incurred if the set is branched upon.

The degradation caused by altering an individual row depends (as when branching on general integers) upon the sign of the altered variable(s) in the row and the type of the row. The entries in the COSTUP and COSTDN rows of Table A1.2 above can thus be used to give an indication of what the per unit movement penalties should be for the different combinations of row type and coefficient sign that might occur.

It should be noted however, that the row coefficients of the changed variable(s) are already used in the construction of the 'corrected' and 'uncorrected' vectors used to indicate the actual movement on the row for a Special Ordered Set (as described in section 7.3.1). The actual per unit movement penalties used for each row when dealing with Special Ordered Sets must thus be as in Table A1.3 below.

a_{ki} in row k:	Positive	Positive	Negative	Negative
Row k type:	'$\leq$' row	'$=$' row	'$\leq$' row	'$=$' row
Per Unit Cost of increasing x_i on row k	MIN(-tol$_k$, π_k)	MIN(-tol$_k$, π_k)	MAX(tol$_k$, π_k)	MAX(tol$_k$, π_k)
Per Unit cost of decreasing x_i on row k	MAX(0, π_k)	MAX(tol$_k$, π_k)	MIN(0, π_k)	MIN (-tol$_k$, π_k)

Table A1.3. Per unit cost of changing the value of x_i for row k.

Appendix 2: Test Results for Code Discussed in Chapter Five.

This Appendix contains the results of attacking the set of test problems using the initial n-transputer parallel algorithm developed (as discussed in Chapter Five).

KEY

tptrs	=	number of slave transputers used
time	=	time in seconds to solve problem
nodes	=	nodes taken to proven optimal solution
speedup	=	speedup based on time with one slave transputer

The nodes figure is composed of the following components:

sol nodes	=	number of integer feasible solutions found
inf nodes	=	number of infeasible nodes
cut 1	=	nodes generated to cut off a parent when a new incumbent solution has been found
cut 2	=	nodes that have worse solutions than the present best and are therefore cut off
cut 3	=	nodes that are returned to be added to the candidate list but which are no longer eligible because the cutoff has changed since the LP relaxation was sent out and the returning node is not good enough
cut 4	=	nodes where the LP relaxation has not been solved due to some error in the LP solver on the slaves - the node is cut off in this case and the final solution cannot be proven to be optimal
cut 5	=	nodes that return a new integer feasible solution to become the incumbent but which cannot because another, better incumbent has been found since the LP relaxation was sent out
attack	=	number of nodes added to candidate list

The attack figure is composed of the following components:

never	=	number of nodes added to candidate list but never attacked
once	=	number of nodes added to candidate list and attacked once
twice	=	number of nodes added to candidate list and attacked twice
max tptr	=	maximum number of slave transputers actually used
av tptr	=	average number of slave transputers used

Appendix 2A: Test Results for Code Using Unpacked Data Structures

Test Results for Small Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	4.01	3.57	3.24	3.62	3.62	3.57	3.57	3.57
nodes	7	11	13	13	13	13	13	13
speedup	1.00	1.12	1.24	1.11	1.11	1.12	1.12	1.12
sol nodes	1	2	2	3	3	3	3	3
inf nodes	0	0	0	0	0	0	0	0
cut 1	0	2	4	1	1	1	1	1
cut 2	3	1	0	2	2	2	2	2
cut 3	0	0	1	1	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	0	0	0	0	0	0
attack	3	5	6	6	6	6	6	6
never	0	1	2	0	0	0	0	0
once	0	0	0	1	1	1	1	1
twice	3	4	4	5	5	5	5	5
max tptr	1	2	3	4	4	4	4	4
av tptr	1.00	1.56	1.78	2.50	2.50	2.50	2.50	2.50

Table 2A:1: Test Results for Problem AZA.

tptrs	1	2	3	4	5	6	7	8
time	526.84	311.98	316.86	319.39	300.38	290.18	296.38	280.12
nodes	1901	1689	1845	1809	1713	1649	1701	1555
speedup	1.00	1.69	1.66	1.65	1.75	1.81	1.78	1.88
sol nodes	7	3	2	3	3	3	2	3
inf nodes	50	33	32	30	31	29	31	28
cut 1	201	126	221	202	152	120	146	64
cut 2	693	682	668	670	671	673	672	683
cut 3	0	1	0	0	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	950	844	922	904	856	824	850	777
never	0	50	98	93	71	56	68	29
once	201	26	25	16	10	8	10	6
twice	749	768	799	795	775	760	772	742
max tptr	1	2	3	4	4	4	4	4
av tptr	1.00	1.66	1.79	1.78	1.81	1.82	1.82	1.83

Table 2A:2: Test Results for Problem AZB.

tptrs	1	2	3	4	5	6	7	8
time	60.96	38.00	27.14	24.66	24.88	24.93	25.05	24.71
nodes	145	159	143	141	145	145	141	145
speedup	1.00	1.60	2.25	2.47	2.45	2.45	2.43	2.47
sol nodes	3	4	2	1	2	2	1	2
inf nodes	6	7	6	6	6	6	6	6
cut 1	4	11	12	8	12	12	8	12
cut 2	60	58	52	56	53	53	56	53
cut 3	0	0	0	0	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	72	79	71	70	72	72	70	72
never	0	4	6	4	6	6	4	6
once	4	3	0	0	0	0	0	0
twice	68	72	65	66	66	66	66	66
max tptr	1	2	3	4	5	5	5	5
av tptr	1.00	1.70	2.23	2.63	2.65	2.62	2.66	2.66

Table 2A:3: Test Results for Problem AZC.

Test Results for Small Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	2.42	2.14	2.58	2.63	2.58	2.59	2.63	2.74
nodes	15	21	23	23	23	23	23	23
speedup	1.00	1.13	0.94	0.92	0.94	0.93	0.92	0.88
sol nodes	2	3	3	3	3	3	3	3
inf nodes	2	3	4	4	4	4	4	4
cut 1	0	0	0	0	0	0	0	0
cut 2	4	3	3	3	3	3	3	3
cut 3	0	1	2	2	2	2	2	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	0	0	0	0	0	0
attack	7	10	11	11	11	11	11	11
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	7	10	11	11	11	11	11	11
max tptr	1	2	3	3	3	3	3	3
av tptr	1.00	1.57	1.74	1.78	1.78	1.78	1.78	1.78

Table 2A:4: Test Results for Problem HPW15.

Test Results for Medium Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	34.66	24.71	50.41	50.09	17.20	16.98	73.05	17.14
nodes	99	87	181	181	57	57	249	57
speedup	1.00	1.40	0.69	0.69	2.02	2.04	0.47	2.02
sol nodes	2	1	2	2	1	1	2	1
inf nodes	0	0	0	0	0	0	0	0
cut 1	40	32	74	74	18	18	96	18
cut 2	8	11	15	15	9	9	26	9
cut 3	0	0	0	0	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	49	43	90	90	28	28	124	28
never	0	14	31	31	8	8	41	8
once	40	4	12	12	2	2	14	2
twice	9	25	47	47	18	18	69	18
max tptr	1	2	3	3	3	3	3	3
av tptr	1.00	1.53	1.53	1.54	1.62	1.62	1.53	1.62

Table 2A:5: Test Results for Problem INGT274.

tptrs	1	2	3	4	5	6	7	8
time	1386.98	691.84	549.86	418.59	353.56	328.91	291.32	280.67
nodes	499	491	571	557	571	617	597	615
speedup	1.00	2.00	2.52	3.31	3.92	4.22	4.76	4.94
sol nodes	3	2	7	7	6	6	6	6
inf nodes	223	216	259	249	257	286	273	284
cut 1	1	1	0	2	2	0	1	0
cut 2	23	26	20	20	21	16	19	17
cut 3	0	1	0	1	0	0	0	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	1	0	0
attack	249	245	285	278	285	308	298	307
never	0	0	0	0	0	0	0	0
once	1	1	0	2	2	0	1	0
twice	248	244	285	276	283	308	297	307
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.91	2.81	3.66	4.50	5.26	5.85	6.26

Table 2A:6: Test Results for Problem MRX.

Test Results for Medium Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	1436.74	762.91	523.77	425.84	355.75	312.31	275.94	255.46
nodes	515	543	549	577	577	577	549	549
speedup	1.00	1.88	2.74	3.37	4.04	4.60	5.21	5.62
sol nodes	4	6	4	4	6	5	7	5
inf nodes	234	47	8	5	7	8	53	48
cut 1	1	4	8	5	2	1	1	4
cut 2	19	14	14	13	13	15	11	14
cut 3	0	1	0	0	0	0	1	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	1	2	1	0	2	2
attack	257	271	274	288	288	288	274	274
never	0	0	2	1	1	0	0	0
once	1	4	4	3	0	1	1	4
twice	256	267	268	284	287	287	273	270
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.91	2.78	3.61	4.47	5.18	5.79	6.34

Table 2A:7: Test Results for Problem MR1.

Test Results for Large Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	613.46	339.72	389.80	235.46	167.80	208.94	231.90	192.79
nodes	113	105	177	131	101	141	161	127
speedup	1.00	1.81	1.57	2.61	3.66	2.94	2.65	3.18
sol nodes	5	4	7	3	3	4	4	3
inf nodes	8	4	6	3	3	3	1	2
cut 1	25	18	38	33	26	39	49	40
cut 2	19	27	37	23	14	21	24	14
cut 3	0	0	1	2	4	2	2	5
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	2	1	2	1	0
attack	56	52	88	65	50	70	80	63
never	0	4	9	9	9	13	16	13
once	25	10	20	15	8	13	17	14
twice	31	38	59	41	33	44	47	36
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.83	2.62	3.26	3.80	4.22	4.21	4.15

Table 2A:8: Test Results for Problem CHAL.

tptrs	1	2	3	4	5	6	7	8
time	339.66	164.23	242.49	169.61	468.19	125.34	156.10	125.50
nodes	115	103	203	165	387	129	157	131
speedup	1.00	2.07	1.40	2.00	0.73	2.71	2.18	2.71
sol nodes	6	4	6	6	6	2	2	2
inf nodes	7	4	5	4	19	3	4	3
cut 1	31	31	66	61	73	53	62	53
cut 2	14	11	21	10	92	6	8	6
cut 3	0	0	3	1	0	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	2	1	1	4	0	2	1
attack	57	51	101	82	193	64	78	65
never	0	8	17	22	26	21	23	20
once	31	15	32	17	21	11	16	13
twice	26	28	52	43	146	32	39	32
max tptr	1	2	3	4	5	6	6	6
av tptr	1.00	1.79	2.31	2.87	2.83	2.88	2.95	2.96

Table 2A:9: Test Results for Problem GY.

Test Results for Large Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	77.00	315.17	91.45	91.78	370.81	147.80	134.73	171.42
nodes	59	257	81	79	307	125	117	135
speedup	1.00	0.24	0.84	0.84	0.21	0.52	0.57	0.45
sol nodes	2	3	1	1	2	1	1	2
inf nodes	0	0	0	0	0	0	0	0
cut 1	28	117	35	33	136	53	53	56
cut 2	0	8	4	3	15	7	4	9
cut 3	0	1	1	3	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	1	0	0
attack	29	128	40	39	153	62	58	67
never	0	38	14	14	39	19	21	22
once	28	41	7	5	58	15	11	12
twice	1	49	19	20	56	28	26	33
max tptr	1	2	3	4	3	3	4	3
av tptr	1.00	1.48	1.85	1.85	1.51	1.81	1.89	1.76

Table 2A:10: Test Results for Problem INGT1345.

Test Results for Small MIP Problems

tptrs	1	2	3	4	5	6	7	8
time	23.40	18.95	12.30	14.06	13.95	12.80	13.95	13.96
nodes	111	149	105	115	115	105	115	115
speedup	1.00	1.23	1.90	1.66	1.68	1.83	1.68	1.68
sol nodes	10	2	1	2	2	1	2	2
inf nodes	2	12	9	10	10	9	10	10
cut 1	34	59	43	46	46	43	46	46
cut 2	10	2	0	0	0	0	0	0
cut 3	0	0	0	0	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	55	74	52	57	57	52	57	57
never	0	27	21	22	22	21	22	22
once	34	5	1	2	2	1	2	2
twice	21	42	30	33	33	30	33	33
max tptr	1	2	3	3	3	3	3	3
av tptr	1.00	1.53	1.74	1.71	1.71	1.76	1.71	1.71

Table 2A:11: Test Results for problem DAAC.

tptrs	1	2	3	4	5	6	7	8
time	3.19	2.41	2.41	2.48	2.42	2.42	2.42	2.47
nodes	3	3	3	3	3	3	3	3
speedup	1.00	1.32	1.32	1.29	1.32	1.32	1.32	1.29
sol nodes	1	1	1	1	1	1	1	1
inf nodes	0	0	0	0	0	0	0	0
cut 1	0	0	0	0	0	0	0	0
cut 2	1	0	0	0	0	0	0	0
cut 3	0	1	1	1	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	1	1	1	1	1	1	1	1
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	1	1	1	1	1	1	1	1
max tptr	1	2	2	2	2	2	2	2
av tptr	1.00	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 2A:12: Test Results for Problem G31.

tptrs	1	2	3	4	5	6	7	8
time	5.55	3.90	3.13	3.13	3.13	3.14	3.13	3.07
nodes	5	5	5	5	5	5	5	5
speedup	1.00	1.42	1.77	1.77	1.77	1.77	1.77	1.81
sol nodes	1	1	1	1	1	1	1	1
inf nodes	0	0	0	0	0	0	0	0
cut 1	0	0	0	0	0	0	0	0
cut 2	2	1	0	0	0	0	0	0
cut 3	0	1	2	2	2	2	2	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	2	2	2	2	2	2	2	2
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	2	2	2	2	2	2	2	2
max tptr	1	2	3	3	3	3	3	3
av tptr	1.00	1.40	1.80	1.80	1.80	1.80	1.80	1.80

Table 2A:13: Test Results for Problem G32.

tptrs	1	2	3	4	5	6	7	8
time	189.22	246.23	137.09	163.40	152.74	152.80	163.46	166.10
nodes	735	1839	1139	1357	1257	1219	1323	1281
speedup	1.00	0.77	1.38	1.16	1.24	1.24	1.16	1.14
sol nodes	15	5	2	1	2	2	2	3
inf nodes	12	74	22	27	23	26	26	27
cut 1	125	781	527	629	579	536	609	557
cut 2	216	60	19	22	24	46	25	54
cut 3	0	0	0	0	1	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	367	919	569	678	628	609	661	640
never	0	351	250	299	276	253	288	265
once	125	79	27	31	27	30	33	27
twice	242	489	292	348	325	326	340	348
max tptr	1	2	3	4	3	4	3	3
av tptr	1.00	1.56	1.80	1.78	1.79	1.78	1.78	1.76

Table 2A:14: Test Results for Problem OK.

tptrs	1	2	3	4	5	6	7	8
time	1.76	1.05	1.04	0.83	0.82	0.82	0.83	0.83
nodes	31	27	23	25	25	25	25	25
speedup	1.00	1.68	1.69	2.12	2.15	2.15	2.12	2.12
sol nodes	2	1	1	1	1	1	1	1
inf nodes	10	8	6	7	7	7	7	7
cut 1	1	5	4	4	4	4	4	4
cut 2	3	0	1	1	1	1	1	1
cut 3	0	0	0	0	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	15	13	11	12	12	12	12	12
never	0	2	2	2	2	2	2	2
once	1	1	0	0	0	0	0	0
twice	14	10	9	10	10	10	10	10
max tptr	1	2	3	4	4	4	4	4
av tptr	1	1.5	1.63	1.9	1.9	1.9	1.9	1.9

Table 2A:15: Test Results for Problem SETX.

Test Results for Medium MIP Problems

tptrs	1	2	3	4	5	6	7	8
time	405.07	124.35	108.59	63.06	50.04	58.88	58.83	56.96
nodes	237	159	211	161	141	193	203	209
speedup	1.00	3.26	3.73	6.42	8.09	6.88	6.89	7.11
sol nodes	16	8	9	5	7	5	4	3
inf nodes	0	0	0	0	0	0	0	2
cut 1	53	44	71	61	46	76	78	84
cut 2	50	27	24	15	18	14	17	14
cut 3	0	1	1	0	0	2	3	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	1	0	0	0	0	0
attack	118	79	105	80	70	96	101	104
never	0	8	19	25	20	32	34	34
once	53	28	33	11	6	12	10	16
twice	65	43	53	44	44	52	57	54
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.86	2.63	3.15	3.91	4.41	4.82	4.88

Table 2A:16: Test Results for Problem BAG882.

tptrs	1	2	3	4	5	6	7	8
time	3384	1807	1220	1044	865	790	793	752
nodes	1649	1757	1665	1781	1657	1667	1731	1667
speedup	1.00	1.87	2.77	3.24	3.91	4.28	4.27	4.50
sol nodes	1	5	1	6	1	1	2	1
inf nodes	791	833	791	838	791	791	820	791
cut 1	4	7	8	16	5	10	9	13
cut 2	29	33	32	30	32	31	33	29
cut 3	0	0	0	0	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	1	1	0	1	2	0
attack	824	878	832	890	828	833	865	833
never	0	3	4	7	2	5	4	6
once	4	1	0	2	1	0	1	1
twice	820	874	828	881	825	828	860	826
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.85	2.7	3.34	4.10	4.61	4.76	4.97

Table 2A:17: Test Results for Problem TAX1.

tptrs	1	2	3	4	5	6	7	8
time	129.07	24.39	25.10	20.70	21.15	20.71	21.59	22.14
nodes	165	37	49	43	47	51	51	57
speedup	1.00	5.29	5.14	6.24	6.10	6.23	5.98	5.83
sol nodes	7	1	1	1	1	1	1	1
inf nodes	58	8	13	13	13	14	15	16
cut 1	4	3	7	4	8	8	5	9
cut 2	14	6	3	1	0	0	1	1
cut 3	0	1	1	3	2	3	4	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	82	18	24	21	23	25	25	28
never	0	1	3	2	4	4	2	4
once	4	1	1	0	0	0	1	1
twice	78	16	20	19	19	21	22	23
max tptr	1	2	3	4	5	6	7	7
av tptr	1.00	1.74	2.21	2.74	2.79	3.28	3.39	3.5

Table 2A:18: Test Results for Problem TAX2.

tptrs	1	2	3	4	5	6	7	8
time	40.27	29.16	27.95	27.90	28.01	27.90	28.23	27.95
nodes	23	29	39	41	43	43	43	43
speedup	1.00	1.38	1.44	1.44	1.44	1.44	1.43	1.44
sol nodes	4	4	5	5	5	5	5	5
inf nodes	4	4	6	5	6	6	6	6
cut 1	3	4	6	7	7	7	7	7
cut 2	1	1	0	3	3	3	3	3
cut 3	0	2	3	1	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	11	14	19	20	21	21	21	21
never	0	0	2	3	3	3	3	3
once	3	4	2	1	1	1	1	1
twice	8	10	15	16	17	17	17	17
max tptr	1	2	3	4	5	5	5	5
av tptr	1.00	1.80	2.21	2.53	2.56	2.56	2.56	2.56

Table 2A:19: Test Results for Problem CRAC.

tptrs	1	2	3	4	5	6	7	8
time	211.30	100.35	82.61	58.95	56.45	61.03	46.03	43.89
nodes	31	33	35	31	35	43	37	33
speedup	1.00	2.11	2.56	3.58	3.74	3.46	4.59	4.81
sol nodes	8	9	8	6	6	6	2	3
inf nodes	0	0	0	0	0	0	0	0
cut 1	2	6	3	6	5	7	8	5
cut 2	6	1	4	2	1	1	1	0
cut 3	0	0	1	0	3	1	5	5
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	2	2	3	7	3	4
attack	15	16	17	15	17	21	18	16
never	0	2	0	1	1	2	3	2
once	2	2	3	4	3	3	2	1
twice	13	12	14	10	13	16	13	13
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.81	2.75	3.24	4.17	4.75	5.28	5.71

Table 2A:20: Test Results for Problem DOM1.

tptrs	1	2	3	4	5	6	7	8
time	307.50	160.60	82.00	68.28	80.68	73.38	55.37	51.30
nodes	103	101	55	55	111	121	93	89
speedup	1.00	1.92	3.75	4.50	3.81	4.19	5.55	6.00
sol nodes	3	4	2	1	4	2	3	2
inf nodes	0	0	0	0	0	2	0	5
cut 1	20	23	3	2	31	43	27	28
cut 2	29	23	22	23	19	13	13	5
cut 3	0	0	1	1	2	0	3	5
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	0	1	0	1	1	0
attack	51	50	27	27	55	60	46	44
never	0	3	1	1	7	17	10	12
once	20	17	1	0	17	9	7	4
twice	31	30	25	26	31	34	29	28
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.86	2.75	3.43	4.16	4.76	5.20	5.07

Table 2A:21: Test Results for Problem MCA.

tptrs	1	2	3	4	5	6	7	8
time	55911	25545	11624	8517	7014	8724	5035	4895
nodes	3917	3581	2207	2237	2223	3371	2107	2401
speedup	1.00	2.19	4.81	6.56	7.97	6.41	11.10	11.42
sol nodes	12	21	9	13	8	10	7	10
inf nodes	0	0	0	0	0	0	0	0
cut 1	628	509	169	275	218	364	136	213
cut 2	1319	1258	924	821	884	1305	906	972
cut 3	0	1	2	9	1	2	3	5
cut 4	0	0	0	0	0	0	0	0
cut 5	0	2	0	1	1	5	2	1
attack	1958	1790	1103	1118	1111	1685	1053	1200
never	0	3	7	19	14	10	9	14
once	628	503	155	237	190	344	118	185
twice	1330	1284	941	862	907	1331	926	1001
max tptr	1	2	3	4	5	6	7	8
av tptr	1.00	1.96	2.91	3.83	4.79	5.86	6.62	7.50

Table 2A:22: Test Results for Problem MO0788.

Appendix 2B: Test Results for Code Using Packed Data Structures

Test Results for Small Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	3.62	2.96	2.97	3.07	2.8	2.64	2.75	2.69
nodes	7	11	13	13	13	13	15	13
speedup	1.00	1.22	1.22	1.18	1.29	1.37	1.32	1.35
sol nodes	1	2	2	2	2	2	2	2
inf nodes	0	0	0	0	0	0	0	0
cut 1	0	1	1	0	0	0	2	0
cut 2	3	3	2	2	1	1	1	1
cut 3	0	0	1	1	2	2	1	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	1	2	2	2	2	2
attack	3	5	6	6	6	6	7	6
never	0	0	0	0	0	0	1	0
once	0	1	1	0	0	0	0	0
twice	3	4	5	6	6	6	6	6
max tptr	1	2	3	4	5	6	6	6
av tptr	1	1.8	2.5	2.69	2.92	2.85	3	3

Table 2B:1: Test Results for Problem AZA.

tptrs	1	2	3	4	5	6	7	8
time	388.4	216.4	154.7	149.5	142.7	152.4	148.1	146.9
nodes	1901	1871	1665	1529	1511	1537	1547	1523
speedup	1.00	1.80	2.51	2.60	2.72	2.55	2.62	2.64
sol nodes	7	4	1	1	2	1	2	2
inf nodes	50	35	29	28	28	28	29	28
cut 1	201	222	132	62	43	65	70	60
cut 2	693	674	671	674	683	675	672	672
cut 3	0	1	0	0	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	1	0
attack	950	935	832	764	755	768	773	761
never	0	79	50	30	20	32	34	29
once	201	64	32	2	3	1	2	2
twice	749	792	750	732	732	735	737	730
max tptr	1	2	3	4	5	6	5	6
av tptr	1	1.73	2.33	2.55	2.65	2.56	2.62	2.6

Table 2B:2: Test Results for Problem AZB.

tptrs	1	2	3	4	5	6	7	8
time	49.7	26.64	19.39	16.75	15.76	13.95	14.12	13.73
nodes	145	145	143	137	139	141	141	137
speedup	1.00	1.87	2.56	2.97	3.15	3.56	3.52	3.62
sol nodes	3	2	2	2	2	2	2	2
inf nodes	6	6	6	6	6	6	6	6
cut 1	4	10	9	3	5	8	8	5
cut 2	60	55	55	58	57	54	55	56
cut 3	0	0	0	0	0	1	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	72	72	71	68	69	70	70	68
never	0	5	4	1	2	4	4	2
once	4	0	1	1	1	0	0	1
twice	68	67	66	66	66	66	66	65
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.76	2.46	3.05	3.47	4.02	4.02	4.1

Table 2B:3: Test Results for Problem AZC.

Test Results for Small Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	1.82	1.76	1.59	1.04	0.93	0.93	0.98	0.93
nodes	15	23	23	23	23	23	23	23
speedup	1.00	1.03	1.14	1.75	1.96	1.96	1.86	1.96
sol nodes	2	3	3	1	1	1	1	1
inf nodes	2	4	4	3	3	3	3	3
cut 1	0	0	0	7	7	7	7	7
cut 2	4	3	3	0	0	0	0	0
cut 3	0	1	1	1	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	1	0	0	0	0	0
attack	7	11	11	11	11	11	11	11
never	0	0	0	3	3	3	3	3
once	0	0	0	1	1	1	1	1
twice	7	11	11	7	7	7	7	7
max tptr	1	2	3	4	5	5	5	5
av tptr	1	1.7	2.04	2.31	2.44	2.44	2.44	2.44

Table 2B:4: Test Results for Problem HPW15.

Test Results for Medium Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	28.23	34.83	48.06	42.62	45.09	42.51	46.41	45.2
nodes	99	167	245	213	225	209	229	225
speedup	1.00	0.81	0.59	0.66	0.63	0.66	0.61	0.62
sol nodes	2	2	2	2	2	1	2	2
inf nodes	0	0	0	0	0	0	0	0
cut 1	40	67	101	86	91	85	92	91
cut 2	8	14	18	18	19	19	20	19
cut 3	0	1	2	1	1	0	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	49	83	122	106	112	104	114	112
never	0	28	44	36	39	38	39	39
once	40	11	13	14	13	9	14	13
twice	9	44	65	56	60	57	61	60
max tptr	1	2	3	3	3	3	3	3
av tptr	1	1.53	1.7	1.69	1.68	1.68	1.66	1.68

Table 2B:5: Test Results for Problem INGT274.

tptrs	1	2	3	4	5	6	7	8
time	1323	695.3	502.4	382.9	318.7	292.2	246.7	217.2
nodes	499	519	561	559	571	617	595	593
speedup	1.00	1.90	2.63	3.46	4.15	4.53	5.36	6.09
sol nodes	3	5	6	7	6	8	5	5
inf nodes	223	230	251	251	258	286	268	268
cut 1	1	1	3	1	1	0	4	2
cut 2	23	24	21	21	21	15	20	20
cut 3	0	0	0	0	0	0	1	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	249	259	280	279	285	308	297	296
never	0	0	1	0	0	0	1	0
once	1	1	1	1	1	0	2	2
twice	248	258	278	278	284	308	294	294
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.95	2.87	3.8	4.7	5.55	6.41	7.27

Table 2B:6: Test Results for Problem MRX.

Test Results for Medium Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	1372	699.2	485.9	380	306.3	271.6	232.4	213.9
nodes	515	521	537	555	541	569	555	573
speedup	1.00	1.96	2.82	3.61	4.48	5.05	5.91	6.42
sol nodes	4	5	7	7	6	6	4	6
inf nodes	234	234	246	255	246	263	256	265
cut 1	1	3	0	2	4	2	1	0
cut 2	19	17	16	14	14	14	16	15
cut 3	0	2	0	0	1	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	1	1
attack	257	260	268	277	270	284	277	286
never	0	0	0	0	1	1	0	0
once	1	3	0	2	2	0	1	0
twice	256	257	268	275	267	283	276	286
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.95	2.89	3.78	4.63	5.57	6.42	7.17

Table 2B:7: Test Results for Problem MR1.

Test Results for Large Combinatorial Problems

tptrs	1	2	3	4	5	6	7	8
time	539.5	331.7	250.4	194.8	192.1	170.3	183.8	151.3
nodes	113	125	135	125	169	183	211	183
speedup	1.00	1.63	2.16	2.77	2.81	3.17	2.93	3.57
sol nodes	5	5	7	4	5	6	7	5
inf nodes	8	7	7	4	4	6	8	5
cut 1	25	21	22	25	38	49	59	43
cut 2	19	27	29	29	34	23	26	30
cut 3	0	1	3	1	4	8	4	8
cut 4	0	0	0	0	0	0	0	0
cut 5	0	2	0	0	0	0	2	1
attack	56	62	67	62	84	91	105	91
never	0	1	0	7	10	10	15	13
once	25	19	22	11	18	29	29	17
twice	31	42	45	44	56	52	61	61
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.92	2.86	3.52	4.44	5.19	6.07	6.62

Table 2B:8: Test Results for Problem CHAL.

tptrs	1	2	3	4	5	6	7	8
time	284.6	235.7	155.6	136.6	109.3	78.32	304.6	143.6
nodes	115	221	211	207	199	157	519	297
speedup	1.00	1.21	1.83	2.08	2.60	3.63	0.93	1.98
sol nodes	6	5	4	6	4	1	7	12
inf nodes	7	6	3	4	3	4	22	6
cut 1	31	88	86	79	81	65	101	120
cut 2	14	12	9	10	8	4	121	10
cut 3	0	0	1	3	1	3	7	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	3	2	3	2	2	0
attack	57	110	105	103	99	78	259	148
never	0	9	19	22	26	22	40	45
once	31	70	48	35	29	21	21	30
twice	26	31	38	46	44	35	198	73
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.87	2.7	3.34	4.05	4.7	4.88	4.92

Table 2B:9: Test Results for Problem GY.

<u>Test Results for Large Combinatorial Problems</u>

tptrs	1	2	3	4	5	6	7	8
time	62.18	171.9	52.35	48.39	50.59	50.53	47.51	52.07
nodes	59	193	61	63	63	63	61	65
speedup	1.00	0.36	1.19	1.28	1.23	1.23	1.31	1.19
sol nodes	2	2	1	1	1	1	1	1
inf nodes	0	0	0	0	0	0	0	0
cut 1	28	83	24	30	29	29	28	30
cut 2	0	11	5	0	1	1	0	0
cut 3	0	1	1	1	1	1	2	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	29	96	30	31	31	31	30	32
never	0	29	9	13	12	12	12	13
once	28	25	6	4	5	5	4	4
twice	1	42	15	14	14	14	14	15
max tptr	1	2	3	4	4	4	4	4
av tptr	1	1.58	1.92	2.12	1.97	1.97	2.12	2.06

Table 2B:10: Test Results for Problem INGT1345.

<u>Test Results for Small MIP Problems</u>

tptrs	1	2	3	4	5	6	7	8
time	15.33	6.1	5.22	4.95	4.78	4.89	4.17	4.12
nodes	111	75	79	95	97	99	89	89
speedup	1.00	2.51	2.94	3.10	3.21	3.13	3.68	3.72
sol nodes	10	3	5	2	2	3	1	1
inf nodes	2	4	8	5	7	8	4	4
cut 1	34	24	18	39	36	35	38	38
cut 2	10	7	8	1	2	2	0	0
cut 3	0	0	1	1	2	2	2	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	55	37	39	47	48	49	44	44
never	0	6	6	17	15	16	18	18
once	34	12	6	5	6	3	2	2
twice	21	19	27	25	27	30	24	24
max tptr	1	2	3	4	5	6	7	7
av tptr	1	1.75	2.26	2.59	2.77	3	2.86	2.86

Table 2B:11: Test Results for Problem DAAC.

tptrs	1	2	3	4	5	6	7	8
time	3.02	2.25	2.25	2.31	2.25	2.25	2.25	2.25
nodes	3	3	3	3	3	3	3	3
speedup	1.00	1.34	1.34	1.31	1.34	1.34	1.34	1.34
sol nodes	1	1	1	1	1	1	1	1
inf nodes	0	0	0	0	0	0	0	0
cut 1	0	0	0	0	0	0	0	0
cut 2	1	0	0	0	0	0	0	0
cut 3	0	1	1	1	1	1	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	1	1	1	1	1	1	1	1
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	1	1	1	1	1	1	1	1
max tptr	1	2	2	2	2	2	2	2
av tptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 2B:12: Test Results for Problem G31.

<u>Test Results for Small MIP Problems</u>

tptrs	1	2	3	4	5	6	7	8
time	5.16	3.51	2.8	2.75	2.8	2.8	2.8	2.81
nodes	5	5	5	5	5	5	5	5
speedup	1.00	1.47	1.84	1.88	1.84	1.84	1.84	1.84
sol nodes	1	1	1	1	1	1	1	1
inf nodes	0	0	0	0	0	0	0	0
cut 1	0	0	0	0	0	0	0	0
cut 2	2	1	0	0	0	0	0	0
cut 3	0	1	2	2	2	2	2	2
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	2	2	2	2	2	2	2	2
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	2	2	2	2	2	2	2	2
max tptr	1	2	3	3	3	3	3	3
av tptr	1	1.6	1.8	1.8	1.8	1.8	1.8	1.8

Table 2B:13: Test Results for Problem G32.

tptrs	1	2	3	4	5	6	7	8
time	127.2	90.52	66.3	64.49	49.27	53.17	50.42	56.46
nodes	735	1051	1073	1235	909	1039	959	1097
speedup	1.00	1.40	1.92	1.97	2.58	2.39	2.52	2.25
sol nodes	15	6	5	2	2	2	2	3
inf nodes	12	16	21	23	12	14	15	20
cut 1	125	336	430	569	379	462	424	503
cut 2	216	167	81	23	60	42	38	22
cut 3	0	1	0	1	2	0	1	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	367	525	536	617	454	519	479	548
never	0	119	185	260	178	215	203	235
once	125	98	60	49	23	32	18	33
twice	242	308	291	308	253	272	258	280
max tptr	1	2	3	4	5	6	7	6
av tptr	1	1.7	2.21	2.61	2.72	2.72	2.78	2.77

Table 2B:14: Test Results for Problem OK.

tptrs	1	2	3	4	5	6	7	8
time	1.26	0.76	0.55	0.6	0.6	0.55	0.55	0.61
nodes	31	31	27	31	33	31	33	33
speedup	1.00	1.66	2.29	2.10	2.10	2.29	2.29	2.07
sol nodes	2	3	1	1	1	1	1	1
inf nodes	10	9	7	9	9	8	9	9
cut 1	3	3	5	5	7	7	7	7
cut 2	1	0	0	0	0	0	0	0
cut 3	0	0	1	1	0	0	0	0
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	0	0	0	0	0	0
attack	15	15	13	15	16	15	16	16
never	0	1	2	2	3	3	3	3
once	3	1	1	1	1	1	1	1
twice	12	13	10	12	12	11	12	12
max tptr	1	2	3	4	5	4	5	5
av tptr	1	1.68	2.14	2.35	2.5	2.33	2.5	2.5

Table 2B:15: Test Results for Problem SETEX.

Test Results for Medium MIP Problems

tptrs	1	2	3	4	5	6	7	8
time	360.3	123.5	90.08	68.99	57.56	51.47	47.68	37.79
nodes	237	187	205	205	209	227	251	213
speedup	1.00	2.92	4.00	5.22	6.26	7.00	7.56	9.53
sol nodes	16	11	8	8	6	8	8	9
inf nodes	0	1	2	0	0	1	0	0
cut 1	53	53	64	60	69	76	88	74
cut 2	50	27	24	28	22	21	26	18
cut 3	0	2	5	6	8	4	0	4
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	1	0	4	4	2
attack	118	93	102	102	104	113	125	106
never	0	3	5	7	10	16	21	17
once	53	47	54	46	49	44	46	40
twice	65	43	43	49	45	53	58	49
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.94	2.89	3.81	4.7	5.5	6.43	7.22

Table 2B:16: Test Results for Problem BAG882.

tptrs	1	2	3	4	5	6	7	8
time	3032	1530	1026	783.3	683.4	541.2	482.7	423.6
nodes	1649	1655	1657	1661	1993	1649	1735	1661
speedup	1.00	1.98	2.96	3.87	4.44	5.60	6.28	7.16
sol nodes	1	1	1	1	7	1	5	1
inf nodes	791	791	791	791	882	791	825	791
cut 1	4	4	4	7	61	0	6	4
cut 2	29	31	31	29	40	29	30	31
cut 3	0	0	0	0	1	3	0	3
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	2	3	6	1	2	1
attack	824	827	828	830	996	824	867	830
never	0	2	1	2	23	0	2	2
once	4	0	2	3	15	0	2	0
twice	820	825	825	825	958	824	863	828
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.94	2.89	3.81	4.54	5.58	6.36	7.25

Table 2B:17: Test Results for Problem TAX1.

tptrs	1	2	3	4	5	6	7	8
time	105	21.53	19.39	19.56	17.91	16.58	17.08	17.14
nodes	165	39	43	57	55	51	51	53
speedup	1.00	4.88	5.42	5.37	5.86	6.33	6.15	6.13
sol nodes	7	1	1	1	1	1	1	1
inf nodes	58	8	12	12	10	14	14	15
cut 1	4	3	4	12	11	7	5	4
cut 2	14	7	3	3	3	0	1	0
cut 3	0	1	2	1	3	4	5	7
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	0	0	0	0	0	0
attack	82	19	21	28	27	25	25	26
never	0	0	1	5	5	3	2	2
once	4	3	2	2	1	1	1	0
twice	78	16	18	21	21	21	22	24
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.81	2.38	2.93	3.45	3.91	4.35	4.63

Table 2B:18: Test Results for Problem TAX2.

<u>Test Results for Large MIP Problems</u>

tptrs	1	2	3	4	5	6	7	8
time	32.85	24.99	23.45	23.56	23.62	23.62	23.62	23.62
nodes	23	33	45	35	31	31	31	31
speedup	1.00	1.31	1.40	1.39	1.39	1.39	1.39	1.39
sol nodes	4	4	7	5	4	4	4	4
inf nodes	4	5	9	6	4	4	4	4
cut 1	3	5	3	1	0	0	0	0
cut 2	1	1	3	3	3	3	3	3
cut 3	0	1	1	3	5	5	5	5
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	0	0	0	0	0	0
attack	11	16	22	17	15	15	15	15
never	0	0	0	0	0	0	0	0
once	3	5	3	1	0	0	0	0
twice	8	11	19	16	15	15	15	15
max tptr	1	2	3	4	5	6	6	6
av tptr	1	1.89	2.6	3.06	3.29	3.35	3.35	3.35

Table 2B:19: Test Results for Problem CRAC.

tptrs	1	2	3	4	5	6	7	8
time	197.7	106.8	69.8	53.3	48.8	48.8	48.9	48.8
nodes	31	33	31	29	33	37	41	45
speedup	1.00	1.85	2.83	3.71	4.05	4.05	4.04	4.05
sol nodes	8	9	7	5	6	6	6	6
inf nodes	0	0	0	0	0	0	0	0
cut 1	2	2	4	6	6	6	7	7
cut 2	6	2	2	1	0	1	1	1
cut 3	0	0	1	0	0	0	0	1
cut 4	0	0	0	0	0	0	0	0
cut 5	0	4	2	3	5	6	7	8
attack	15	16	15	14	16	18	20	22
never	0	0	0	1	1	0	0	1
once	2	2	4	4	4	6	7	5
twice	13	14	11	9	11	12	13	16
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.94	2.74	3.30	4.22	4.97	5.71	6.42

Table 2B:20: Test Results for Problem DOM1.

tptrs	1	2	3	4	5	6	7	8
time	276.1	128.4	82.83	81.07	69.21	54.6	47.35	48.45
nodes	103	85	77	123	125	119	117	127
speedup	1.00	2.15	3.33	3.41	3.99	5.06	5.83	5.70
sol nodes	3	3	3	5	4	4	4	4
inf nodes	0	0	0	1	1	1	1	1
cut 1	20	15	13	32	35	33	33	37
cut 2	29	24	20	19	19	17	13	15
cut 3	0	1	2	3	3	4	8	6
cut 4	0	0	0	0	0	0	0	0
cut 5	0	0	1	2	1	1	0	1
attack	51	42	38	61	62	59	58	63
never	0	1	1	1	5	5	5	6
once	20	13	11	30	25	23	23	25
twice	31	28	26	30	32	31	30	32
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.94	2.87	3.79	4.53	5.4	6.21	6.91

Table 2B:21: Test Results for Problem MCA.

tptrs	1	2	3	4	5	6	7	8
time	52228	26508	14505	13585	9398	6901	6952	7380
nodes	3917	4001	3133	4129	3399	2839	3527	4435
speedup	1.00	1.97	3.60	3.84	5.56	7.57	7.51	7.08
sol nodes	12	14	14	19	14	10	13	16
inf nodes	0	0	0	0	0	0	0	0
cut 1	628	660	381	687	450	287	489	747
cut 2	1319	1323	1167	1350	1226	1115	1246	1445
cut 3	0	3	3	6	7	6	13	8
cut 4	0	0	0	0	0	0	0	0
cut 5	0	1	2	3	3	2	3	2
attack	1958	2000	1566	2064	1699	1419	1763	2217
never	0	1	1	1	4	2	6	5
once	628	658	379	685	442	283	477	737
twice	1330	1341	1186	1378	1253	1134	1280	1475
max tptr	1	2	3	4	5	6	7	8
av tptr	1	1.99	2.98	3.97	4.96	5.95	6.94	7.93

Table 28:22: Test Results for Problem MO0788.

Appendix 2C: Comparison of Solution Times for Code Using Packed and Unpacked Data Structures

This Appendix contains graphs showing the solution times obtained by attacking the set of test problems using four different n-transputer codes.

RUN1 uses the code discussed at the beginning of Chapter 5 (the results for which are held in Appendix 2A). This code does not use packed data structures, and sends messages one word at a time.

RUN2 uses a similar code to RUN1, except that packed data structures are used to reduce the disk-reading and message-passing times. The messages are still sent one word at a time.

RUN3 again uses packed data structures, but data is sent and received word by word by the appropriate processes on transputers, but passed between transputers as four messages whose length depends on the size of the problem attacked and the number of integer entities it contains.

RUN4 also uses packed data structures, but data is sent, passed and received as four messages whose length depends on the size of the problem attacked and the number of integer entities it contains.

Test runs three and four can be used to examine the overheads accumulated at the slave and master processors by sending messages of different lengths.

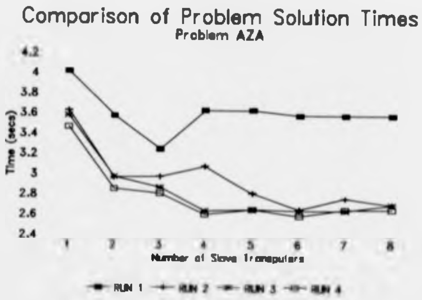


Fig. 2C:1: Solution Time Comparisons for Problem AZA.

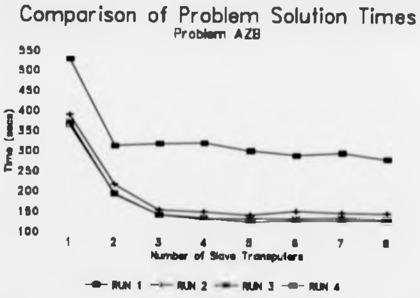


Fig. 2C:2: Solution Time Comparisons for Problem AZB.

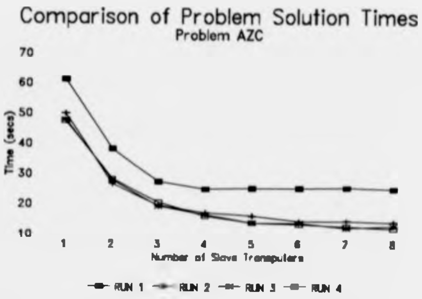


Fig. 2C:3: Solution Time Comparisons for Problem AZC.

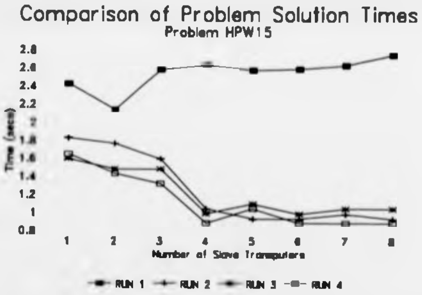


Fig. 2C:4: Solution Time Comparisons for Problem HPW15.

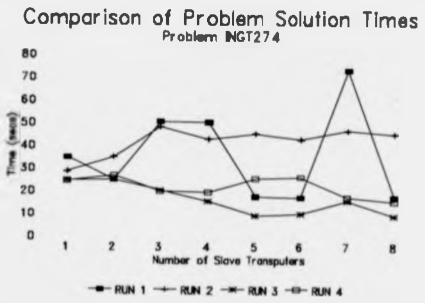


Fig. 2C:5: Solution Time Comparisons for Problem INGT274.

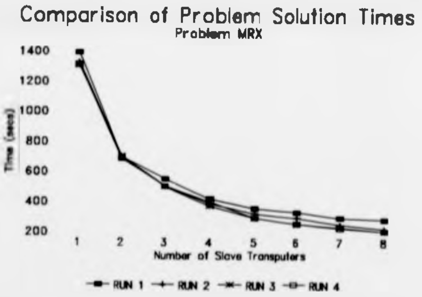


Fig. 2C:6: Solution Time Comparisons for Problem MRX.

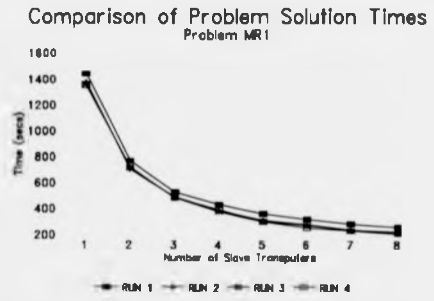


Fig. 2C:7: Solution Time Comparisons for Problem MR1.

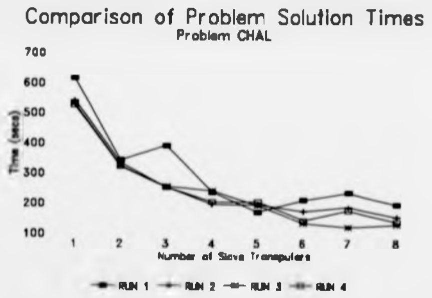


Fig. 2C:8: Solution Time Comparisons for Problem CHAL.

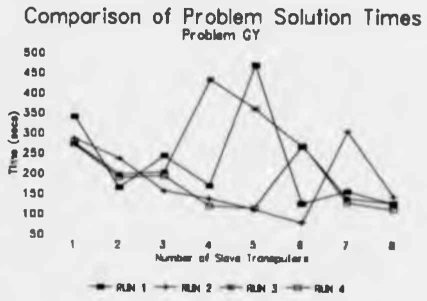


Fig. 2C:9: Solution Time Comparisons for Problem GY.

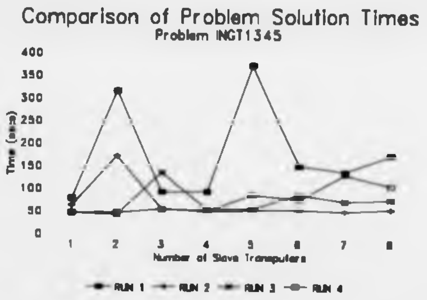


Fig. 2C:10: Solution Time Comparisons for Problem INGT1345.

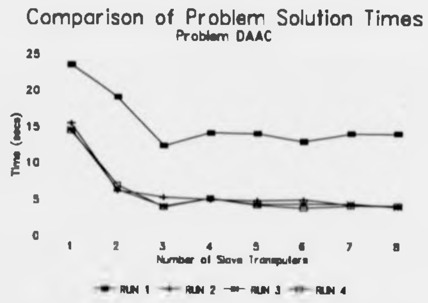


Fig. 2C:11: Solution Time Comparisons for Problem DAAC.

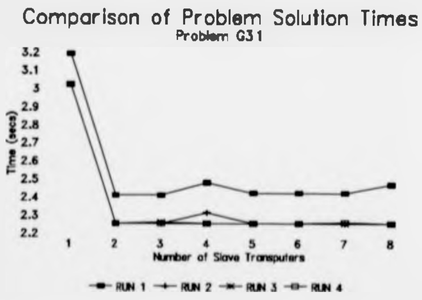


Fig. 2C:12: Solution Time Comparisons for Problem G31.

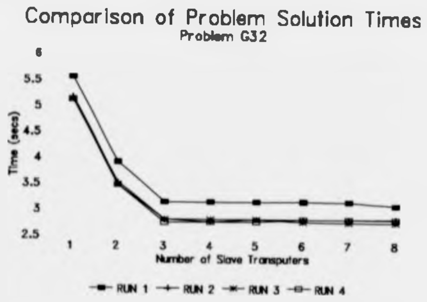


Fig. 2C:13: Solution Time Comparisons for Problem G32.

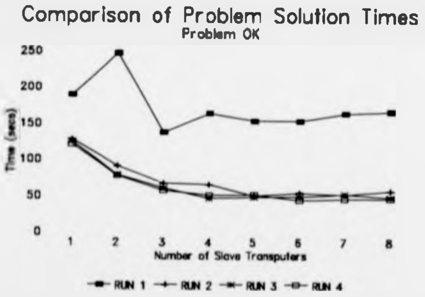


Fig. 2C:14: Solution Time Comparisons for Problem OK.

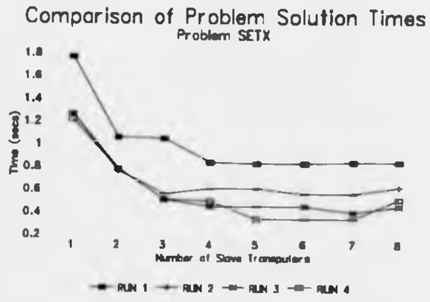


Fig. 2C:15: Solution Time Comparisons for Problem SETX.

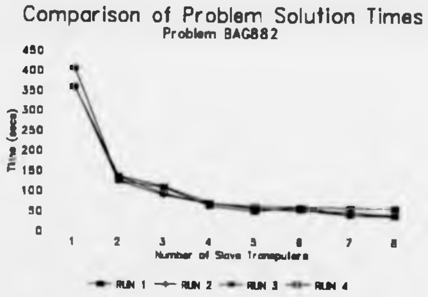


Fig. 2C:16: Solution Time Comparisons for Problem BAG882.

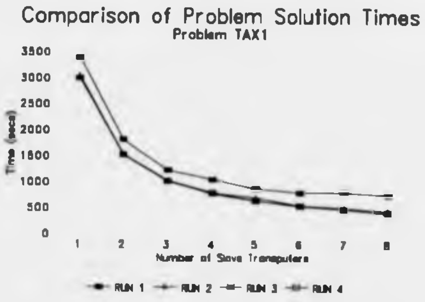


Fig. 2C:17: Solution Time Comparisons for Problem TAX1.

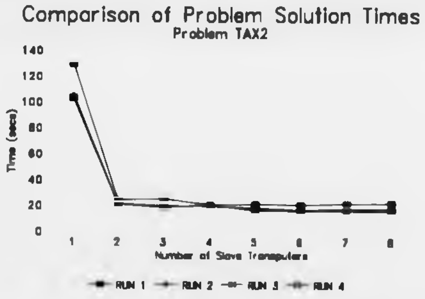


Fig. 2C:18: Solution Time Comparisons for Problem TAX2.

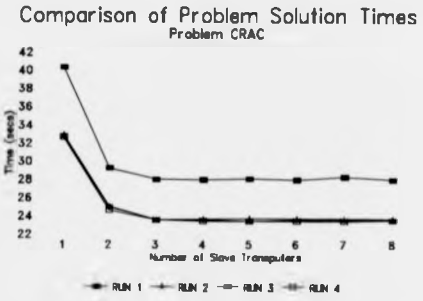


Fig. 2C:19: Solution Time Comparisons for Problem CRAC.

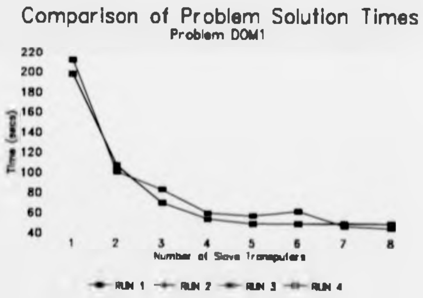


Fig. 2C:20: Solution Time Comparisons for Problem DOM1.

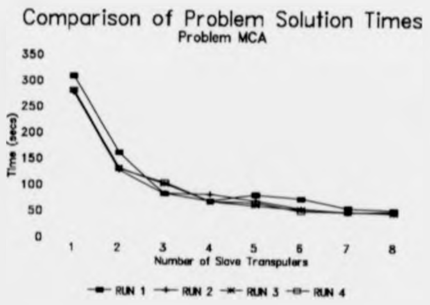


Fig. 2C:21: Solution Time Comparisons for Problem MCA.

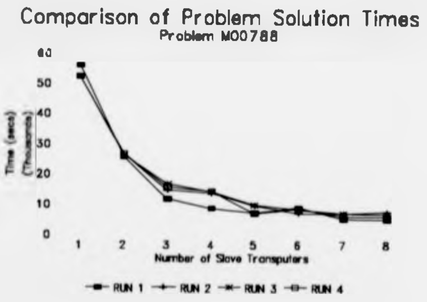


Fig. 2C:22: Solution Time Comparisons for Problem MO0788.

Appendix 3: Test Results for Code Discussed in Chapter Six.

This Appendix contains the results of attacking the set of test problems using the various node selection strategies discussed in Chapter Six.

KEY

tptrs	=	number of slave transputers used
time	=	time in seconds to solve problem
nodes	=	nodes taken to proven optimal solution
speedup	=	speedup based on time with one slave transputer

The nodes figure is composed of the following components:

sol nodes	=	number of integer feasible solutions found
inf nodes	=	number of infeasible nodes
cut 1	=	nodes generated to cut off a parent when a new incumbent solution has been found
cut 2	=	nodes that have worse solutions than the present best and are therefore cut off
cut 3	=	nodes that are returned to be added to the candidate list but which are no longer eligible because the cutoff has changed since the LP relaxation was sent out and the returning node is not good enough
cut 4	=	nodes where the LP relaxation has not been solved due to some error in the LP solver on the slaves - the node is cut off in this case and the final solution cannot be proven to be optimal
cut 5	=	nodes that return a new integer feasible solution to become the incumbent but which cannot because another, better incumbent has been found since the LP relaxation was sent out
attack	=	number of nodes added to candidate list

The attack figure is composed of the following components:

never	=	number of nodes added to candidate list but never attacked
once	=	number of nodes added to candidate list and attacked once
twice	=	number of nodes added to candidate list and attacked twice
max tptr	=	maximum number of slave transputers actually used
av tptr	=	average number of slave transputers used

Test results using Node Selection Strategy One

tptrs	1	2	3	4	5	6	7	8
time	3.62	2.73	2.74	2.69	2.69	2.69	2.7	2.69
nodes	9	11	13	13	13	13	13	13
speedup	1.00	1.33	1.32	1.35	1.35	1.35	1.34	1.35
sol	1	2	2	2	2	2	2	2
inf	0	0	0	0	0	0	0	0
cut1	2	3	2	1	0	0	0	0
cut2	2	0	1	1	1	1	1	1
cut3	0	1	1	1	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	1	2	2	2	2	2
attack	4	5	6	6	6	6	6	6
never	1	1	0	0	0	0	0	0
once	0	1	2	1	0	0	0	0
twice	3	3	4	5	6	6	6	6
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.75	2.45	2.67	2.92	3	3	3

Table 3:1: Results for AZA using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	333.95	198.11	149.46	151.32	141.37	148.63	141.43	148.41
nodes	1467	1467	1467	1467	1467	1467	1467	1467
speedup	1.00	1.69	2.23	2.21	2.36	2.25	2.36	2.25
sol	1	1	1	1	1	1	1	1
inf	28	28	28	28	28	28	28	28
cut1	28	28	28	28	28	28	28	28
cut2	677	677	677	677	677	677	677	677
cut3	0	0	0	0	0	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	733	733	733	733	733	733	733	733
never	14	14	14	14	14	14	14	14
once	0	0	0	0	0	0	0	0
twice	719	719	719	719	719	719	719	719
maxtptr	1	2	3	4	5	6	6	7
avtptr	1	1.7	2.31	2.51	2.62	2.55	2.63	2.58

Table 3:2: Results for AZB using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	47.45	25.76	18.73	15.82	15.43	13.46	14.06	13.62
nodes	131	131	133	135	135	135	135	135
speedup	1.00	1.84	2.53	3.00	3.08	3.53	3.37	3.48
sol	1	1	1	1	1	1	1	1
inf	6	6	6	6	6	6	6	6
cut1	0	0	2	4	4	4	4	4
cut2	59	59	58	57	57	57	56	56
cut3	0	0	0	0	0	0	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	65	65	66	67	67	67	67	67
never	0	0	1	2	2	2	2	2
once	0	0	0	0	0	0	0	0
twice	65	65	65	65	65	65	65	65
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.82	2.48	3.15	3.37	3.9	3.88	4.15

Table 3:3: Results for AZC using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	1.86	0.93	0.87	0.93	0.94	0.94	0.88	0.94
nodes	17	15	17	21	21	21	21	21
speedup	1.00	2.00	2.14	2.00	1.98	1.98	2.11	1.98
sol	2	1	1	1	1	1	1	1
inf	3	2	2	2	2	2	2	2
cut1	1	2	4	7	7	7	7	7
cut2	3	2	1	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	8	7	8	10	10	10	10	10
never	0	0	1	2	2	2	2	2
once	1	2	2	3	3	3	3	3
twice	7	5	5	5	5	5	5	5
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.62	1.92	2.14	2.14	2.14	2.14	2.14

Table 3:4: Results for HPW15 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	48.5	19.94	23.73	26.42	27.24	23.73	24.22	31.09
nodes	185	111	133	139	133	133	129	165
speedup	1.00	2.43	2.04	1.84	1.78	2.04	2.00	1.56
sol	1	2	2	2	2	2	2	2
inf	0	0	0	0	0	0	0	0
cut1	92	51	62	64	57	61	60	77
cut2	0	2	3	4	7	3	3	3
cut3	0	1	0	0	1	1	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	92	55	66	69	66	66	64	82
never	39	17	23	23	18	22	27	27
once	14	17	16	18	21	17	6	23
twice	39	21	27	28	27	27	31	32
maxtptr	1	2	3	3	3	3	3	3
avtptr	1	1.45	1.58	1.51	1.53	1.58	1.57	1.49

Table 3:5: Results for INGT274 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	1360.9	713.26	530.14	407.87	311.42	280.29	251.28	222.99
nodes	515	537	605	605	549	603	615	611
speedup	1.00	1.91	2.57	3.34	4.37	4.86	5.42	6.10
sol	5	6	6	5	5	6	5	5
inf	225	233	273	273	244	272	283	281
cut1	4	5	6	6	1	5	1	0
cut2	24	25	18	18	24	19	17	17
cut3	0	0	0	0	0	0	1	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	1	0	1	1
attack	257	268	302	302	274	301	307	305
never	1	1	2	1	0	1	0	0
once	2	3	2	4	1	3	1	0
twice	254	264	298	297	273	297	306	305
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.95	2.89	3.8	4.68	5.6	6.43	7.23

Table 3:6: Results for MRX using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	1368.3	719.09	508.88	379.09	305.05	272.92	241.4	213.77
nodes	515	541	569	549	541	577	577	579
speedup	1.00	1.90	2.69	3.61	4.49	5.01	5.67	6.40
sol	4	4	5	5	6	5	4	5
inf	232	245	261	253	245	266	266	268
cut1	3	6	4	1	5	3	3	2
cut2	19	16	15	15	13	14	14	13
cut3	0	0	0	0	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	1	0	1	1
attack	257	270	284	274	270	288	288	289
never	1	2	1	0	1	0	1	1
once	1	2	2	1	3	3	1	0
twice	255	266	281	273	266	285	286	288
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.94	2.89	3.79	4.68	5.58	6.35	7.15

Table 3:7: Results for MR1 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	388.32	274.62	218.05	127.65	118.14	112.93	116.61	116.33
nodes	77	109	133	93	107	119	127	143
speedup	1.00	1.41	1.78	3.04	3.29	3.44	3.33	3.34
sol	2	3	5	1	3	3	3	2
inf	4	6	6	4	4	3	3	3
cut1	17	25	37	26	35	43	43	50
cut2	16	18	18	13	11	11	12	13
cut3	0	2	0	2	1	0	3	3
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	1	0	0	0	1
attack	38	54	66	46	53	59	63	71
never	2	1	10	5	9	13	12	13
once	13	23	17	16	17	17	19	24
twice	23	30	39	25	27	29	32	34
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.98	2.84	3.57	4.25	4.92	5.58	6.33

Table 3:8: Results for CHAL using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	582.81	568.7	410.9	337.96	302.25	282.43	397.33	277.15
nodes	305	599	617	621	635	635	923	651
speedup	1.00	1.02	1.42	1.72	1.93	2.06	1.47	2.10
sol	2	3	4	1	2	4	4	5
inf	14	27	27	27	27	27	40	27
cut1	126	259	266	275	281	279	409	284
cut2	11	10	10	5	5	4	5	5
cut3	0	0	1	1	1	1	3	3
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	2	2	3	1	2
attack	152	299	308	310	317	317	461	325
never	62	123	126	132	132	131	195	132
once	2	13	14	11	17	17	19	20
twice	88	163	168	167	168	169	247	173
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.85	2.66	3.33	3.96	4.41	4.58	4.76

Table 3:9: Results for GY using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	135.23	93.92	83.87	90.41	88.54	85.47	80.25	63.88
nodes	87	105	105	119	115	109	103	79
speedup	1.00	1.44	1.61	1.50	1.53	1.58	1.69	2.12
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	43	52	50	59	56	53	50	37
cut2	0	0	1	0	0	0	0	0
cut3	0	0	1	0	1	1	1	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	43	52	52	59	57	54	51	39
never	18	24	23	28	26	24	23	15
once	7	4	4	3	4	5	4	7
twice	18	24	25	28	27	25	24	17
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.7	2.09	2.18	2.27	2.25	2.3	2.19

Table 3:10: Results for INGT1345 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	6.98	4.73	4.12	3.74	3.84	3.95	3.85	3.79
nodes	59	63	65	67	67	79	79	79
speedup	1.00	1.48	1.69	1.87	1.82	1.77	1.81	1.84
sol	1	1	1	1	1	1	1	1
inf	2	4	4	4	4	5	5	5
cut1	22	23	23	25	24	32	32	32
cut2	5	4	4	3	3	1	0	0
cut3	0	0	1	1	2	1	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	29	31	32	33	33	39	39	39
never	10	11	10	11	10	15	15	15
once	2	1	3	3	4	2	2	2
twice	17	19	19	19	19	22	22	22
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.65	2.12	2.4	2.6	2.96	3.02	3.02

Table 3:11: Results for DAAC using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	3.02	2.25	2.26	2.26	2.25	2.25	2.25	2.3
nodes	3	3	3	3	3	3	3	3
speedup	1.00	1.34	1.34	1.34	1.34	1.34	1.34	1.31
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	1	0	0	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	1	1	1	1	1	1	1	1
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	1	1	1	1	1	1	1	1
maxtptr	1	2	2	2	2	2	2	2
avtptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 3:12: Results for G31 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	5.16	3.46	2.74	2.8	2.8	2.8	2.8	2.81
nodes	5	5	5	5	5	5	5	5
speedup	1.00	1.49	1.88	1.84	1.84	1.84	1.84	1.84
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	2	1	0	0	0	0	0	0
cut3	0	1	2	2	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	2	2	2	2	2	2	2	2
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	2	2	2	2	2	2	2	2
maxtptr	1	2	3	3	3	3	3	3
avtptr	1	1.6	1.8	1.8	1.8	1.8	1.8	1.8

Table 3:13: Results for G32 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	80.24	48.67	39.38	34.33	33.95	34.71	34.71	34.33
nodes	603	603	609	609	611	611	605	613
speedup	1.00	1.65	2.04	2.34	2.36	2.31	2.31	2.34
sol	1	1	1	1	1	1	1	1
inf	4	7	7	7	7	7	7	7
cut1	233	230	232	233	233	233	229	235
cut2	64	64	64	64	64	64	64	64
cut3	0	0	1	0	1	1	2	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	301	301	304	304	305	305	302	306
never	115	115	114	115	114	114	113	115
once	3	0	4	3	5	5	3	5
twice	183	186	186	186	186	186	186	186
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.68	2.2	2.63	2.76	2.74	2.74	2.82

Table 3:14: Results for OK using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	1.16	0.77	0.55	0.55	0.5	0.55	0.55	0.55
nodes	27	29	25	23	25	25	25	25
speedup	1.00	1.51	2.11	2.11	2.32	2.11	2.11	2.11
sol	2	1	1	1	1	1	1	1
inf	9	9	8	6	8	8	8	8
cut1	3	4	3	4	3	3	3	3
cut2	0	0	0	0	0	0	0	0
cut3	0	0	0	1	0	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	0	1	1	1	1
attack	13	14	12	11	12	12	12	12
never	0	2	1	2	1	1	1	1
once	3	0	1	0	1	1	1	1
twice	10	12	10	9	10	10	10	10
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.68	2	2.21	2.27	2.27	2.27	2.27

Table 3:15: Results for SETX using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	183.84	58.45	41.64	35.37	32.35	29.99	28.07	27.13
nodes	155	101	105	111	135	139	151	163
speedup	1.00	3.15	4.41	5.20	5.68	6.13	6.55	6.78
sol	6	3	3	3	3	4	4	3
inf	0	0	0	0	0	0	0	0
cut1	52	35	37	39	52	51	58	65
cut2	20	13	11	11	10	11	10	8
cut3	0	0	0	1	0	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	2	2	3	3	3	5
attack	77	50	52	55	67	69	75	81
never	26	17	18	18	22	20	22	25
once	0	1	1	3	8	11	14	15
twice	51	32	33	34	37	38	39	41
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.88	2.82	3.64	4.69	5.43	6.25	7.02

Table 3:16: Results for BAG882 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	3054	1536.8	1039.8	790.54	639.44	546.95	478.73	429.3
nodes	1649	1655	1659	1659	1663	1659	1667	1661
speedup	1.00	1.99	2.94	3.86	4.78	5.58	6.38	7.11
sol	1	1	1	1	1	1	1	1
inf	791	791	791	791	791	791	791	791
cut1	4	6	7	6	8	4	9	6
cut2	29	29	29	29	31	31	29	30
cut3	0	0	0	0	0	2	0	2
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	3	1	1	4	1
attack	824	827	829	829	831	829	833	830
never	0	1	2	1	1	0	1	1
once	4	4	3	4	6	4	7	4
twice	820	822	824	824	824	825	825	825
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.94	2.86	3.76	4.7	5.48	6.36	7.14

Table 3:17: Results for TAX1 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	81.51	20.43	17.8	17.25	17.13	16.75	17.19	17.08
nodes	123	35	39	41	45	53	51	55
speedup	1.00	3.99	4.58	4.73	4.76	4.87	4.74	4.77
sol	2	1	1	1	1	1	1	1
inf	42	8	9	11	13	13	14	15
cut1	10	2	5	4	5	11	4	5
cut2	8	6	3	2	0	0	1	0
cut3	0	1	2	3	4	2	6	7
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	61	17	19	20	22	26	25	27
never	1	1	2	2	2	5	1	1
once	8	0	1	0	1	1	2	3
twice	52	16	16	18	19	20	22	23
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.76	2.29	2.86	3.33	3.83	4.49	4.94

Table 3:18: Results for TAX2 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	26.81	24.05	23.46	23.56	23.56	23.61	23.61	23.56
nodes	21	33	37	37	31	31	31	31
speedup	1.00	1.11	1.14	1.14	1.14	1.14	1.14	1.14
sol	1	2	3	4	4	4	4	4
inf	1	2	5	5	4	4	4	4
cut1	9	12	6	5	0	0	0	0
cut2	0	0	3	3	3	3	3	3
cut3	0	1	2	2	5	5	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	10	16	18	18	15	15	15	15
never	4	6	2	2	0	0	0	0
once	1	0	2	1	0	0	0	0
twice	5	10	14	15	15	15	15	15
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.76	2.48	3.03	3.29	3.35	3.35	3.35

Table 3:19: Results for CRAC using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	80.96	49.49	41.2	31.8	31.75	29.77	29.33	29.38
nodes	23	25	27	21	25	27	27	31
speedup	1.00	1.64	1.97	2.55	2.55	2.72	2.76	2.76
sol	2	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	10	11	11	7	8	8	7	8
cut2	0	0	0	0	0	0	0	0
cut3	0	0	0	2	3	4	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	1	1	1	1	2
attack	11	12	13	10	12	13	13	15
never	5	5	5	3	4	4	3	3
once	0	1	1	1	0	0	1	2
twice	6	6	7	6	8	9	9	10
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.86	2.62	3.07	3.76	4.32	4.8	5.48

Table 3:20: Results for DOM1 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	209.04	108.75	76.34	55.09	46.03	41.03	38.34	36.09
nodes	87	87	81	81	77	83	77	77
speedup	1.00	1.92	2.74	3.79	4.54	5.09	5.45	5.79
sol	2	2	2	2	1	1	1	1
inf	2	2	2	2	2	2	2	2
cut1	32	32	25	26	23	26	22	20
cut2	8	8	10	10	11	9	9	9
cut3	0	0	2	1	1	2	4	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	1	2	1	2
attack	43	43	40	40	38	41	38	38
never	16	16	12	13	11	13	10	10
once	0	0	1	0	1	0	2	0
twice	27	27	27	27	26	28	26	28
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.96	2.87	3.75	4.41	5.19	5.64	6.37

Table 3:21: Results for MCA using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
time	42995	21426	14305	10837	8423.4	7179.4	6325.8	5553.3
nodes	3663	3647	3653	3677	3541	3635	3767	3755
speedup	1.00	2.01	3.01	3.97	5.10	5.99	6.80	7.74
sol	10	9	9	10	8	8	9	9
inf	0	0	0	0	0	0	0	0
cut1	832	823	825	826	773	805	859	848
cut2	990	989	990	1000	982	1000	1006	1011
cut3	0	2	2	2	5	2	7	8
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	1	3	3	3	2
attack	1831	1823	1826	1838	1770	1817	1883	1877
never	3	4	4	4	6	2	1	0
once	826	815	817	818	761	801	857	848
twice	1002	1004	1005	1016	1003	1014	1025	1029
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.99	2.98	3.97	4.95	5.94	6.93	7.91

Table 3:22: Results for MO0788 using Node Selection Strategy 1.

<u>Test results using Node Selection Strategy Two</u>

tptrs	1	2	3	4	5	6	7	8
time	3.62	2.97	2.91	2.75	2.69	2.7	2.7	2.69
nodes	9	11	13	13	13	13	13	13
speedup	1.00	1.22	1.24	1.32	1.35	1.34	1.34	1.35
sol	1	2	2	2	2	2	2	2
inf	0	0	0	0	0	0	0	0
cut1	2	2	2	1	0	0	0	0
cut2	2	1	1	1	1	1	1	1
cut3	0	0	1	1	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	2	2	2	2	2
attack	4	5	6	6	6	6	6	6
never	1	0	0	0	0	0	0	0
once	0	2	2	1	0	0	0	0
twice	3	3	4	5	6	6	6	6
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.78	2.45	2.67	2.92	3	3	3

Table 3:23: Results for AZA using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	331.75	190.32	159.77	152.36	142.3	141.16	141.71	150.87
nodes	1467	1467	1467	1467	1467	1467	1467	1467
speedup	1.00	1.74	2.08	2.18	2.33	2.35	2.34	2.20
sol	1	1	1	1	1	1	1	1
inf	28	28	28	28	28	28	28	28
cut1	28	28	28	28	28	28	28	28
cut2	677	677	677	677	677	677	677	677
cut3	0	0	0	0	0	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	733	733	733	733	733	733	733	733
never	14	14	14	14	14	14	14	14
once	0	0	0	0	0	0	0	0
twice	719	719	719	719	719	719	719	719
maxtptr	1	2	3	4	5	6	7	6
avtptr	1	1.75	2.26	2.51	2.67	2.71	2.7	2.63

Table 3:24: Results for AZB using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	46.85	25.1	18.4	15.54	15.16	13.95	14.39	14
nodes	131	131	133	135	135	135	135	135
speedup	1.00	1.87	2.55	3.01	3.09	3.36	3.26	3.35
sol	1	1	1	1	1	1	1	1
inf	6	6	6	6	6	6	6	6
cut1	0	0	2	4	4	4	4	4
cut2	59	59	57	57	57	57	56	56
cut3	0	0	1	0	0	0	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	65	65	66	67	67	67	67	67
never	0	0	1	2	2	2	2	2
once	0	0	0	0	0	0	0	0
twice	65	65	65	65	65	65	65	65
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.89	2.54	3.18	3.44	3.85	4.03	4.13

Table 3:25: Results for AZC using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	1.86	0.94	0.88	0.93	0.88	0.88	0.93	0.93
nodes	17	15	17	21	21	21	21	21
speedup	1.00	1.98	2.11	2.00	2.11	2.11	2.00	2.00
sol	2	1	1	1	1	1	1	1
inf	3	2	2	2	2	2	2	2
cut1	1	2	4	7	7	7	7	7
cut2	3	2	1	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	8	7	8	10	10	10	10	10
never	0	0	1	2	2	2	2	2
once	1	2	2	3	3	3	3	3
twice	7	5	5	5	5	5	5	5
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.62	1.92	2.14	2.14	2.14	2.14	2.14

Table 3:26: Results for HPW15 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	48.45	41.36	46.68	68.71	60.8	64.59	64.76	69.26
nodes	185	217	259	379	329	341	357	379
speedup	1.00	1.17	1.04	0.71	0.80	0.75	0.75	0.70
sol	1	2	2	2	2	2	2	2
inf	0	0	0	0	0	0	0	0
cut1	92	105	126	185	161	163	170	185
cut2	0	0	1	1	2	5	5	3
cut3	0	2	1	2	0	1	2	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	92	108	129	189	164	170	178	189
never	39	44	55	77	67	62	70	77
once	14	17	16	31	27	39	30	31
twice	39	47	58	81	70	69	78	81
maxtptr	1	2	3	4	3	4	4	3
avtptr	1	1.54	1.72	1.72	1.65	1.69	1.73	1.7

Table 3:27: Results for INGT274 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	1353.8	710.29	538.54	400.96	305.16	278.3	243.59	216.35
nodes	513	547	615	597	555	601	609	603
speedup	1.00	1.91	2.51	3.38	4.44	4.86	5.56	6.26
sol	6	4	5	3	3	4	3	4
inf	224	231	283	264	237	262	271	263
cut1	4	19	4	16	17	19	13	18
cut2	23	20	16	15	20	16	16	16
cut3	0	0	0	0	0	0	1	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	1	0	1	1
attack	256	273	307	298	277	300	304	301
never	1	3	0	5	4	4	3	4
once	2	13	4	6	9	11	7	10
twice	253	257	303	287	264	285	294	287
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.96	2.88	3.77	4.72	5.56	6.49	7.31

Table 3:28: Results for MRX using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	1358.3	708.81	494.22	379.04	310.33	265.56	227.17	212.18
nodes	515	539	565	565	575	577	559	577
speedup	1.00	1.92	2.75	3.58	4.38	5.11	5.98	6.40
sol	3	3	3	3	5	4	3	5
inf	227	237	249	255	253	257	244	264
cut1	9	14	17	11	17	14	19	6
cut2	19	16	14	14	11	14	12	12
cut3	0	0	0	0	1	0	2	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	1	0	0	1
attack	257	269	282	282	287	288	279	288
never	2	3	3	1	2	2	3	1
once	5	8	11	9	13	10	13	4
twice	250	258	268	272	272	276	263	283
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.96	2.89	3.81	4.7	5.59	6.4	7.13

Table 3:29: Results for MR1 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	363.99	251.29	199.16	118.97	111.94	114.41	112	126.88
nodes	77	107	129	85	107	119	127	153
speedup	1.00	1.45	1.83	3.06	3.25	3.18	3.25	2.87
sol	2	3	6	2	3	3	3	1
inf	3	5	5	2	3	3	3	5
cut1	21	31	42	25	38	44	44	57
cut2	13	12	11	13	9	8	12	9
cut3	0	3	1	1	1	2	2	3
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	2
attack	38	53	64	42	53	59	63	76
never	3	4	9	7	10	13	12	14
once	15	23	24	11	18	18	20	29
twice	20	26	31	24	25	28	31	33
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.89	2.8	3.47	4.29	4.91	5.66	5.99

Table 3:30: Results for CHAL using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	588.58	564.53	413.81	479.11	182.85	114.52	403.65	396.07
nodes	307	597	617	897	363	231	917	927
speedup	1.00	1.04	1.42	1.23	3.22	5.14	1.46	1.49
sol	3	4	5	3	5	2	5	6
inf	14	27	27	40	14	5	40	40
cut1	126	256	264	394	154	94	403	405
cut2	11	11	10	7	6	6	6	6
cut3	0	1	3	2	2	4	4	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	3	1	5	1	2
attack	153	298	308	448	181	115	458	463
never	61	122	125	187	70	36	189	190
once	4	12	14	20	14	22	25	25
twice	88	164	169	241	97	57	244	248
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.86	2.64	3.4	4.06	4.4	4.42	4.62

Table 3:31: Results for GY using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	135.33	101.45	83.48	83.87	83.76	72.56	70.47	77.77
nodes	87	109	105	109	109	93	89	103
speedup	1.00	1.33	1.62	1.61	1.62	1.87	1.92	1.74
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	43	53	49	53	53	44	40	50
cut2	0	0	1	0	0	0	0	0
cut3	0	1	2	1	1	2	4	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	43	54	52	54	54	46	44	51
never	18	24	23	24	24	20	18	23
once	7	5	3	5	5	4	4	4
twice	18	25	26	25	25	22	22	24
maxtptr	1	2	3	4	4	4	5	4
avtptr	1	1.62	2.12	2.29	2.29	2.2	2.33	2.25

Table 3:32: Results for INGT1345 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	6.92	4.83	4.12	3.79	3.57	3.62	3.79	3.79
nodes	59	63	65	67	67	75	79	79
speedup	1.00	1.43	1.68	1.83	1.94	1.91	1.83	1.83
sol	1	1	1	1	1	1	1	1
inf	2	4	4	4	4	4	5	5
cut1	22	23	23	25	24	31	32	32
cut2	5	4	4	3	3	0	0	0
cut3	0	0	1	1	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	29	31	32	33	33	37	39	39
never	10	11	10	11	10	14	15	15
once	2	1	3	3	4	3	2	2
twice	17	19	19	19	19	20	22	22
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.6	2.1	2.4	2.84	2.91	3.02	3.02

Table 3:33: Results for DAAC using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	2.97	2.25	2.26	2.25	2.25	2.25	2.26	2.25
nodes	3	3	3	3	3	3	3	3
speedup	1.00	1.32	1.31	1.32	1.32	1.32	1.31	1.32
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	1	0	0	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	1	1	1	1	1	1	1	1
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	1	1	1	1	1	1	1	1
maxtptr	1	2	2	2	2	2	2	2
avtptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 3:34: Results for G31 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	5.1	3.46	2.8	2.8	2.75	2.75	2.8	2.81
nodes	5	5	5	5	5	5	5	5
speedup	1.00	1.47	1.82	1.82	1.85	1.85	1.82	1.81
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	2	1	0	0	0	0	0	0
cut3	0	1	2	2	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	2	2	2	2	2	2	2	2
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	2	2	2	2	2	2	2	2
maxtptr	1	2	3	3	3	3	3	3
avtptr	1	1.6	1.8	1.8	1.8	1.8	1.8	1.8

Table 3:35: Results for G32 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	81.68	49.92	37.46	35.1	36.3	33.12	34.54	34.38
nodes	603	605	607	605	611	605	611	611
speedup	1.00	1.64	2.18	2.33	2.25	2.47	2.36	2.38
sol	1	1	1	1	1	1	1	1
inf	4	6	7	7	7	7	7	7
cut1	233	232	232	229	233	229	232	232
cut2	64	64	64	64	64	64	64	64
cut3	0	0	0	2	1	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	301	302	303	302	305	302	305	305
never	115	115	115	113	114	113	113	113
once	3	2	2	3	5	3	6	6
twice	183	185	186	186	186	186	186	186
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.69	2.22	2.59	2.67	2.8	2.73	2.75

Table 3:36: Results for OK using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	1.09	0.71	0.49	0.5	0.49	0.49	0.5	0.49
nodes	27	29	25	25	25	25	25	25
speedup	1.00	1.54	2.22	2.18	2.22	2.22	2.18	2.22
sol	2	1	1	1	1	1	1	1
inf	9	9	6	7	8	8	8	8
cut1	3	4	6	4	3	3	3	3
cut2	0	0	0	0	0	0	0	0
cut3	0	0	0	1	0	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	1	0	0	1	1	1	1
attack	13	14	12	12	12	12	12	12
never	0	2	3	2	1	1	1	1
once	3	0	0	0	1	1	1	1
twice	10	12	9	10	10	10	10	10
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.68	1.95	2.52	2.27	2.27	2.27	2.27

Table 3:37: Results for SETX using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	480.48	155.81	132.77	53.12	39.71	30.43	28.17	27.08
nodes	313	223	273	147	139	139	151	163
speedup	1.00	3.08	3.62	9.05	12.10	15.79	17.06	17.74
sol	8	5	4	5	7	3	4	3
inf	0	0	0	0	0	0	0	0
cut1	55	54	59	34	35	51	56	64
cut2	94	53	71	30	23	12	14	9
cut3	0	0	1	2	2	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	2	3	3	4	2	5
attack	156	111	136	73	69	69	75	81
never	5	3	4	9	14	20	19	24
once	45	48	51	16	7	11	18	16
twice	106	60	81	48	48	38	38	41
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.93	2.95	3.76	4.64	5.36	6.32	7.03

Table 3:38: Results for BAG882 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	1065.7	571.5	404.75	323.84	278.52	241.45	222.17	211.85
nodes	791	817	839	855	859	835	885	883
speedup	1.00	1.86	2.63	3.29	3.83	4.41	4.80	5.03
sol	4	4	4	4	7	6	5	4
inf	227	237	235	240	248	235	254	271
cut1	133	138	149	156	144	146	154	138
cut2	32	28	26	22	24	19	18	15
cut3	0	0	2	4	3	7	6	7
cut4	0	0	0	0	0	0	0	0
cut5	0	2	4	2	4	5	6	7
attack	395	408	419	427	429	417	442	441
never	20	24	25	29	24	24	27	31
once	93	90	99	98	96	98	100	76
twice	282	294	295	300	309	295	315	334
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.92	2.81	3.73	4.54	5.27	6.17	6.89

Table 3:39: Results for TAX1 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	223.1	114.13	80.02	63.99	54.16	49	44	41.36
nodes	285	285	285	285	285	285	285	285
speedup	1.00	1.95	2.79	3.49	4.12	4.55	5.07	5.39
sol	1	1	1	1	1	1	1	1
inf	125	125	125	125	125	125	125	125
cut1	0	0	0	0	0	0	0	0
cut2	17	17	17	17	17	16	16	17
cut3	0	0	0	0	0	1	1	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	142	142	142	142	142	142	142	142
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	142	142	142	142	142	142	142	142
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.9	2.72	3.58	4.32	4.94	5.65	6.22

Table 3:40: Results for TAX2 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	26.75	24.44	23.45	23.56	23.56	23.62	23.62	23.62
nodes	21	33	37	37	31	31	31	31
speedup	1.00	1.09	1.14	1.14	1.14	1.13	1.13	1.13
sol	1	3	3	4	4	4	4	4
inf	1	4	5	5	4	4	4	4
cut1	9	9	5	5	0	0	0	0
cut2	0	0	3	3	3	3	3	3
cut3	0	1	2	2	5	5	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	1	0	0	0	0	0
attack	10	16	18	18	15	15	15	15
never	4	4	2	2	0	0	0	0
once	1	1	1	1	0	0	0	0
twice	5	11	15	15	15	15	15	15
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.79	2.53	3.03	3.29	3.35	3.35	3.35

Table 3:41: Results for CRAC using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	80.85	49.48	41.3	31.69	31.8	29.87	29.33	29.33
nodes	23	25	27	21	25	27	27	31
speedup	1.00	1.63	1.96	2.55	2.54	2.71	2.76	2.76
sol	2	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	10	11	11	7	8	8	7	8
cut2	0	0	0	0	0	0	0	0
cut3	0	0	0	2	3	4	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	1	1	1	1	2
attack	11	12	13	10	12	13	13	15
never	5	5	5	3	4	4	3	3
once	0	1	1	1	0	0	1	2
twice	6	6	7	6	8	9	9	10
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.86	2.62	3.07	3.76	4.32	4.8	5.48

Table 3:42: Results for DOM1 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	241.78	125.45	75.62	55.86	45.97	40.81	37.62	36.96
nodes	95	95	81	81	77	87	77	71
speedup	1.00	1.93	3.20	4.33	5.26	5.92	6.43	6.54
sol	3	4	2	2	1	1	1	1
inf	4	4	2	2	2	2	2	2
cut1	33	32	24	24	23	30	22	15
cut2	8	7	11	11	11	7	9	11
cut3	0	0	2	2	1	2	4	6
cut4	0	0	0	0	0	0	0	0
cut5	0	1	0	0	1	2	1	1
attack	47	47	40	40	38	43	38	35
never	14	14	11	11	11	14	10	7
once	5	4	2	2	1	2	2	1
twice	28	29	27	27	26	27	26	27
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.94	2.84	3.77	4.43	5.21	5.67	6.77

Table 3:43: Results for MCA using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
time	26017	13098	8816	6532.7	5324.9	4429	3720.5	3909.3
nodes	2097	2101	2117	2043	2107	2077	1939	2307
speedup	1.00	1.99	2.95	3.98	4.89	5.87	6.99	6.66
sol	7	7	6	7	8	6	7	8
inf	0	0	0	0	0	0	0	0
cut1	452	453	456	411	451	426	340	413
cut2	590	589	593	601	590	599	616	725
cut3	0	1	2	2	2	5	5	6
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	1	3	3	2	2
attack	1048	1050	1058	1021	1053	1038	969	1153
never	125	122	119	107	116	117	87	109
once	202	209	218	197	219	192	166	195
twice	721	719	721	717	718	729	716	849
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.99	2.98	3.97	4.95	5.93	6.92	7.91

Table 3:44: Results for MO0788 using Node Selection Strategy 2.

<u>Test results using Node Selection Strategy Three</u>

tptrs	1	2	3	4	5	6	7	8
time	6.31	2.96	2.97	2.69	2.74	2.75	2.69	2.75
nodes	15	11	13	13	13	13	13	13
speedup	1.00	2.13	2.12	2.35	2.30	2.29	2.35	2.29
sol	2	2	2	2	2	2	2	2
inf	0	0	0	0	0	0	0	0
cut1	0	1	1	2	0	0	0	0
cut2	6	3	2	0	1	1	1	1
cut3	0	0	1	1	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	1	2	2	2	2	2
attack	7	5	6	6	6	6	6	6
never	0	0	0	1	0	0	0	0
once	0	1	1	0	0	0	0	0
twice	7	4	5	5	6	6	6	6
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.8	2.5	2.36	2.92	3	3	3

Table 3:45: Results for AZA using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	332.13	1012.2	164.11	177.58	148.02	140.17	140.44	139.45
nodes	1459	8385	1699	1869	1579	1531	1529	1549
speedup	1.00	0.33	2.02	1.87	2.24	2.37	2.36	2.38
sol	2	26	4	4	4	1	1	1
inf	29	218	30	44	30	28	28	28
cut1	7	396	111	76	35	62	60	73
cut2	692	3551	703	811	719	674	675	673
cut3	0	2	1	0	1	1	1	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	1	0	1	0	0	0
attack	729	4192	849	934	789	765	764	774
never	1	144	49	37	17	31	29	35
once	5	108	13	2	1	0	2	3
twice	723	3940	787	895	771	734	733	736
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.75	2.31	2.63	2.72	2.72	2.7	2.72

Table 3:46: Results for AZB using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	45.14	34.88	22.3	20.65	17.25	13.84	18.02	14.77
nodes	133	187	161	177	161	137	185	141
speedup	1.00	1.29	2.02	2.19	2.62	3.26	2.50	3.06
sol	2	4	4	5	3	2	5	2
inf	6	10	7	9	7	6	8	6
cut1	2	2	4	13	10	4	8	8
cut2	57	78	66	62	60	56	69	54
cut3	0	0	0	0	0	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	1	0	2	0
attack	66	93	80	88	80	68	92	70
never	1	1	2	6	4	2	3	4
once	0	0	0	1	2	0	2	0
twice	65	92	78	81	74	66	87	66
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.83	2.49	2.99	3.37	3.95	4.16	3.99

Table 3:47: Results for AZC using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	1.7	1.59	1.54	0.99	0.93	0.94	0.93	0.94
nodes	15	23	23	23	23	23	23	23
speedup	1.00	1.07	1.10	1.72	1.83	1.81	1.83	1.81
sol	2	2	3	1	1	1	1	1
inf	2	4	4	3	3	3	3	3
cut1	1	2	0	7	7	7	7	7
cut2	3	2	3	0	0	0	0	0
cut3	0	0	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	2	1	0	0	0	0	0
attack	7	11	11	11	11	11	11	11
never	0	0	0	3	3	3	3	3
once	1	2	0	1	1	1	1	1
twice	6	9	11	7	7	7	7	7
maxtptr	1	2	3	4	5	5	5	5
avtptr	1	1.76	2.09	2.31	2.44	2.44	2.44	2.44

Table 3:48: Results for HPW15 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	39.66	33.61	25.21	22.41	15.43	23.57	46.69	31.48
nodes	155	171	117	109	83	113	233	157
speedup	1.00	1.18	1.57	1.77	2.57	1.68	0.85	1.26
sol	2	1	2	2	1	2	1	1
inf	0	0	0	0	0	0	0	0
cut1	76	78	44	41	38	40	98	69
cut2	0	7	13	10	2	14	17	8
cut3	0	0	0	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	0	0	0	0
attack	77	85	58	54	41	56	116	78
never	23	33	18	18	16	17	43	32
once	30	12	8	5	6	6	12	5
twice	24	40	32	31	19	33	61	41
maxtptr	1	2	3	3	3	3	3	4
avtptr	1	1.52	1.62	1.65	1.67	1.64	1.69	1.69

Table 3:49: Results for INGT274 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	1373.8	717.49	496.91	380.31	310.33	284.79	242.66	218.71
nodes	523	543	567	561	559	601	601	593
speedup	1.00	1.91	2.76	3.61	4.43	4.82	5.66	6.28
sol	6	7	5	5	5	3	4	4
inf	228	235	244	247	244	271	271	263
cut1	6	9	15	9	10	11	8	10
cut2	22	21	20	20	20	16	16	19
cut3	0	0	0	0	1	0	2	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	261	271	283	280	279	300	300	296
never	2	3	4	2	3	4	2	2
once	2	3	7	5	4	3	4	6
twice	257	265	272	273	272	293	294	288
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.95	2.88	3.8	4.7	5.5	6.48	7.27

Table 3:50: Results for MRX using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	1347.7	710.52	502.07	369.1	302.47	261.66	225.3	208.5
nodes	509	531	563	545	543	549	547	579
speedup	1.00	1.90	2.68	3.65	4.46	5.15	5.98	6.46
sol	3	5	6	5	5	6	5	4
inf	228	242	257	242	242	249	242	259
cut1	5	2	5	10	9	5	13	10
cut2	19	17	14	15	14	15	14	14
cut3	0	0	0	1	2	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	2
attack	254	265	281	272	271	274	273	289
never	2	1	1	3	3	1	3	1
once	1	0	3	4	3	3	7	8
twice	251	264	277	265	265	270	263	280
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.95	2.87	3.81	4.69	5.53	6.41	7.26

Table 3:51: Results for MR1 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	579.9	352.51	281.77	176.09	179.44	143.47	175.98	153.68
nodes	123	139	165	127	161	155	229	209
speedup	1.00	1.65	2.06	3.29	3.23	4.04	3.30	3.77
sol	4	4	7	4	6	4	7	4
inf	8	6	8	4	3	2	5	6
cut1	28	31	35	31	39	47	76	68
cut2	22	28	32	24	26	19	20	22
cut3	0	0	0	1	4	5	7	3
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	0	3	1	0	2
attack	61	69	82	63	80	77	114	104
never	2	3	2	8	11	10	19	20
once	24	25	31	15	17	27	38	28
twice	35	41	49	40	52	40	57	56
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.91	2.85	3.57	4.56	5.17	6.1	6.75

Table 3:52: Results for CHAL using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	198.45	198.67	187.07	154.23	118.75	82.23	152.92	126.55
nodes	73	173	205	219	203	163	303	279
speedup	1.00	1.00	1.06	1.29	1.67	2.41	1.30	1.57
sol	4	5	10	9	5	2	12	5
inf	4	9	6	9	3	4	9	4
cut1	15	59	58	66	75	68	98	113
cut2	14	13	25	19	12	4	26	10
cut3	0	0	3	3	5	3	3	6
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	4	2	1	4	2
attack	36	86	102	109	101	81	151	139
never	2	11	15	20	24	25	33	43
once	11	37	28	26	27	18	32	27
twice	23	38	59	63	50	38	86	69
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.9	2.78	3.5	4.1	4.71	5.11	5.43

Table 3:53: Results for GY using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	203.45	253.26	43.89	96.45	93.98	98.54	52.95	105.07
nodes	177	271	55	119	121	123	67	129
speedup	1.00	0.80	4.64	2.11	2.16	2.06	3.84	1.94
sol	2	3	1	2	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	86	118	24	55	60	60	30	61
cut2	1	13	2	0	0	0	1	2
cut3	0	2	1	3	0	1	2	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	88	135	27	59	60	61	33	64
never	21	47	8	23	26	27	11	26
once	44	24	8	9	8	6	8	9
twice	23	64	11	27	26	28	14	29
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.52	2	2.11	2.1	2.08	2.11	2

Table 3:54: Results for INGT1345 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	7.91	7.58	5.94	4.99	4.34	4.34	4.56	4.29
nodes	55	91	89	97	81	79	81	79
speedup	1.00	1.04	1.33	1.59	1.82	1.82	1.73	1.84
sol	2	3	3	1	1	1	2	2
inf	2	10	6	6	5	5	5	6
cut1	12	24	28	40	33	29	31	28
cut2	12	9	7	1	2	2	2	2
cut3	0	0	1	1	0	3	1	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	27	45	44	48	40	39	40	39
never	4	10	12	17	16	14	14	13
once	4	4	4	6	1	1	3	2
twice	19	31	28	25	23	24	23	24
maxtptr	1	2	3	4	5	6	5	7
avtptr	1	1.78	2.16	2.65	2.56	2.78	2.58	2.86

Table 3:55: Results for DAAC using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	3.02	2.26	2.25	2.25	2.25	2.31	2.25	2.26
nodes	3	3	3	3	3	3	3	3
speedup	1.00	1.34	1.34	1.34	1.34	1.31	1.34	1.34
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	1	0	0	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	1	1	1	1	1	1	1	1
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	1	1	1	1	1	1	1	1
maxtptr	1	2	2	2	2	2	2	2
avtptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 3:56: Results for G31 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	5.1	3.52	2.8	2.8	2.75	2.75	2.74	2.75
nodes	5	5	5	5	5	5	5	5
speedup	1.00	1.45	1.82	1.82	1.85	1.85	1.86	1.85
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	2	1	0	0	0	0	0	0
cut3	0	1	2	2	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	2	2	2	2	2	2	2	2
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	2	2	2	2	2	2	2	2
maxtptr	1	2	3	3	3	3	3	3
avtptr	1	1.6	1.8	1.8	1.8	1.8	1.8	1.8

Table 3:57: Results for G32 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	96.61	57.67	69.31	48.17	50.92	39.21	52.4	51.91
nodes	533	559	1135	851	1015	667	999	1047
speedup	1.00	1.68	1.39	2.01	1.90	2.46	1.84	1.86
sol	5	8	3	4	1	3	1	1
inf	5	20	25	19	18	15	17	14
cut1	39	50	508	319	469	214	465	493
cut2	218	202	32	81	19	99	15	14
cut3	0	0	0	3	1	3	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	266	279	567	425	507	333	499	523
never	5	12	232	147	229	104	227	237
once	29	26	44	25	11	6	11	19
twice	232	241	291	253	267	223	261	267
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.72	2.21	2.65	2.71	2.87	2.69	2.77

Table 3:58: Results for OK using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	0.98	0.76	0.55	0.55	0.5	0.5	0.49	0.43
nodes	25	31	29	27	25	25	25	27
speedup	1.00	1.29	1.78	1.78	1.96	1.96	2.00	2.28
sol	2	3	1	1	1	1	1	1
inf	7	10	8	7	7	8	8	7
cut1	2	1	5	6	4	4	4	6
cut2	2	0	0	0	0	0	0	0
cut3	0	0	0	0	1	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	2	1	0	0	0	0	0
attack	12	15	14	13	12	12	12	13
never	1	0	2	3	2	1	1	3
once	0	1	1	0	0	2	2	0
twice	11	14	11	10	10	9	9	10
maxtptr	1	2	3	4	5	4	4	5
avtptr	1	1.73	2.12	2.19	2.43	2.33	2.33	2.38

Table 3:59: Results for SETX using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	686.51	296.66	151.87	44.27	59.82	59.21	38.45	47.4
nodes	371	343	299	143	219	215	187	235
speedup	1.00	2.31	4.52	15.51	11.48	11.59	17.85	14.48
sol	20	11	16	5	7	10	6	9
inf	0	0	0	0	0	0	0	0
cut1	20	37	64	50	67	43	56	64
cut2	146	123	64	14	31	46	22	37
cut3	0	0	2	2	4	7	6	2
cut4	0	0	0	0	0	0	0	0
cut5	0	1	4	1	1	2	4	6
attack	185	171	149	71	109	107	93	117
never	6	10	10	20	24	15	23	23
once	8	17	44	10	19	13	10	18
twice	171	144	95	41	66	79	60	76
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.95	2.89	3.71	4.71	5.56	6.41	7.29

Table 3:60: Results for BAG882 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	762.37	386.51	224.64	271.99	196.14	169.94	171.03	109.08
nodes	487	491	397	677	533	539	575	381
speedup	1.00	1.97	3.39	2.80	3.89	4.49	4.46	6.99
sol	7	5	5	4	4	4	4	1
inf	146	150	103	191	162	156	172	88
cut1	45	67	76	120	81	94	78	94
cut2	46	22	13	21	14	9	21	2
cut3	0	2	2	2	3	4	7	3
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	3	3	6	3
attack	243	245	198	338	266	269	287	190
never	13	19	25	34	23	27	18	29
once	19	29	26	52	35	40	42	36
twice	211	197	147	252	208	202	227	125
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.92	2.79	3.69	4.61	5.39	6.05	6.87

Table 3:61: Results for TAX1 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	225.03	114.58	81.67	64.21	54.05	47.84	44.33	41.86
nodes	295	285	295	293	287	285	285	285
speedup	1.00	1.96	2.76	3.50	4.16	4.70	5.08	5.38
sol	2	1	2	2	1	1	1	1
inf	126	125	126	125	125	125	125	125
cut1	4	0	4	5	2	0	0	0
cut2	16	17	16	15	16	17	17	17
cut3	0	0	0	0	0	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	147	142	147	146	143	142	142	142
never	2	0	2	2	1	0	0	0
once	0	0	0	1	0	0	0	0
twice	145	142	145	143	142	142	142	142
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.89	2.7	3.53	4.33	5.09	5.61	6.12

Table 3:62: Results for TAX2 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	26.81	25.48	23.4	23.56	23.62	23.62	23.62	23.62
nodes	21	45	43	37	31	31	31	31
speedup	1.00	1.05	1.15	1.14	1.14	1.14	1.14	1.14
sol	1	3	6	4	4	4	4	4
inf	1	4	7	5	4	4	4	4
cut1	9	15	5	5	0	0	0	0
cut2	0	0	3	3	3	3	3	3
cut3	0	1	1	2	5	5	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	10	22	21	18	15	15	15	15
never	4	7	2	2	0	0	0	0
once	1	1	1	1	0	0	0	0
twice	5	14	18	15	15	15	15	15
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.9	2.55	3.03	3.29	3.35	3.35	3.35

Table 3:63: Results for CRAC using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	80.79	71.84	75.75	51.14	38.34	51.14	50.81	45.65
nodes	23	35	47	35	33	45	47	49
speedup	1.00	1.12	1.07	1.58	2.11	1.58	1.59	1.77
sol	2	3	4	3	3	4	5	4
inf	0	0	0	0	0	0	0	0
cut1	10	14	17	10	9	12	11	12
cut2	0	0	1	1	0	2	3	1
cut3	0	0	0	0	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	4	4	4	4	7
attack	11	17	23	17	16	22	23	24
never	5	7	8	4	4	3	2	3
once	0	0	1	2	1	6	7	6
twice	6	10	14	11	11	13	14	15
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.9	2.8	3.44	4.08	5.03	5.78	6.43

Table 3:64: Results for DOM1 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	316.65	198.62	182.41	148.85	48.94	44.93	39.99	48.72
nodes	97	107	143	143	65	67	75	107
speedup	1.00	1.59	1.74	2.13	6.47	7.05	7.92	6.50
sol	6	6	7	4	1	1	3	4
inf	4	5	6	6	2	2	2	6
cut1	12	18	40	44	7	9	13	21
cut2	27	24	17	15	19	17	15	15
cut3	0	0	0	1	3	4	4	8
cut4	0	1	2	2	0	0	0	0
cut5	0	0	0	0	1	1	1	0
attack	48	53	71	71	32	33	37	53
never	3	5	12	14	3	4	5	7
once	6	8	16	16	1	1	3	7
twice	39	40	43	41	28	28	29	39
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.97	2.85	3.76	4.5	5.05	6.05	7.13

Table 3:65: Results for MCA using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	36011	17851	11530	10383	7518.5	6917.3	4990.9	4688.6
nodes	2735	2711	2563	2969	2719	2973	2639	2851
speedup	1.00	2.02	3.12	3.47	4.79	5.21	7.22	7.68
sol	13	10	9	17	14	14	10	11
inf	0	0	0	0	0	0	0	0
cut1	465	463	393	407	400	432	459	502
cut2	890	881	877	1057	938	1025	843	897
cut3	0	1	2	3	7	14	7	15
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	1	1	2	1	1
attack	1367	1355	1281	1484	1359	1486	1319	1425
never	65	67	56	52	62	64	69	72
once	335	329	281	303	276	304	321	358
twice	967	959	944	1129	1021	1118	929	995
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.99	2.98	3.98	4.96	5.95	6.94	7.92

Table 3:66: Results for MO0788 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	762.37	386.51	224.64	271.99	196.14	169.94	171.03	109.08
nodes	487	491	397	677	533	539	575	381
speedup	1.00	1.97	3.39	2.80	3.89	4.49	4.46	6.99
sol	7	5	5	4	4	4	4	1
inf	146	150	103	191	162	156	172	88
cut1	45	67	76	120	81	94	78	94
cut2	46	22	13	21	14	9	21	2
cut3	0	2	2	2	3	4	7	3
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	3	3	6	3
attack	243	245	198	338	266	269	287	190
never	13	19	25	34	23	27	18	29
once	19	29	26	52	35	40	42	36
twice	211	197	147	252	208	202	227	125
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.92	2.79	3.69	4.61	5.39	6.05	6.87

Table 3:61: Results for TAX1 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	225.03	114.58	81.67	64.21	54.05	47.84	44.33	41.86
nodes	295	285	295	293	287	285	285	285
speedup	1.00	1.96	2.76	3.50	4.16	4.70	5.08	5.38
sol	2	1	2	2	1	1	1	1
inf	126	125	126	125	125	125	125	125
cut1	4	0	4	5	2	0	0	0
cut2	16	17	16	15	16	17	17	17
cut3	0	0	0	0	0	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	147	142	147	146	143	142	142	142
never	2	0	2	2	1	0	0	0
once	0	0	0	1	0	0	0	0
twice	145	142	145	143	142	142	142	142
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.89	2.7	3.53	4.33	5.09	5.61	6.12

Table 3:62: Results for TAX2 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	26.81	25.48	23.4	23.56	23.62	23.62	23.62	23.62
nodes	21	45	43	37	31	31	31	31
speedup	1.00	1.05	1.15	1.14	1.14	1.14	1.14	1.14
sol	1	3	6	4	4	4	4	4
inf	1	4	7	5	4	4	4	4
cut1	9	15	5	5	0	0	0	0
cut2	0	0	3	3	3	3	3	3
cut3	0	1	1	2	5	5	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	10	22	21	18	15	15	15	15
never	4	7	2	2	0	0	0	0
once	1	1	1	1	0	0	0	0
twice	5	14	18	15	15	15	15	15
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.9	2.55	3.03	3.29	3.35	3.35	3.35

Table 3:63: Results for CRAC using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	80.79	71.84	75.75	51.14	38.34	51.14	50.81	45.65
nodes	23	35	47	35	33	45	47	49
speedup	1.00	1.12	1.07	1.58	2.11	1.58	1.59	1.77
sol	2	3	4	3	3	4	5	4
inf	0	0	0	0	0	0	0	0
cut1	10	14	17	10	9	12	11	12
cut2	0	0	1	1	0	2	3	1
cut3	0	0	0	0	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	4	4	4	4	7
attack	11	17	23	17	16	22	23	24
never	5	7	8	4	4	3	2	3
once	0	0	1	2	1	6	7	6
twice	6	10	14	11	11	13	14	15
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.9	2.8	3.44	4.08	5.03	5.78	6.43

Table 3:64: Results for DOM1 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	316.65	198.62	182.41	148.85	48.94	44.93	39.99	48.72
nodes	97	107	143	143	65	67	75	107
speedup	1.00	1.59	1.74	2.13	6.47	7.05	7.92	6.50
sol	6	6	7	4	1	1	3	4
inf	4	5	6	6	2	2	2	6
cut1	12	18	40	44	7	9	13	21
cut2	27	24	17	15	19	17	15	15
cut3	0	0	0	1	3	4	4	8
cut4	0	1	2	2	0	0	0	0
cut5	0	0	0	0	1	1	1	0
attack	48	53	71	71	32	33	37	53
never	3	5	12	14	3	4	5	7
once	6	8	16	16	1	1	3	7
twice	39	40	43	41	28	28	29	39
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.97	2.85	3.76	4.5	5.05	6.05	7.13

Table 3:65: Results for MCA using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
time	36011	17851	11530	10383	7518.5	6917.3	4990.9	4688.6
nodes	2735	2711	2563	2969	2719	2973	2639	2851
speedup	1.00	2.02	3.12	3.47	4.79	5.21	7.22	7.68
sol	13	10	9	17	14	14	10	11
inf	0	0	0	0	0	0	0	0
cut1	465	463	393	407	400	432	459	502
cut2	890	881	877	1057	938	1025	843	897
cut3	0	1	2	3	7	14	7	15
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	1	1	2	1	1
attack	1367	1355	1281	1484	1359	1486	1319	1425
never	65	67	56	52	62	64	69	72
once	335	329	281	303	276	304	321	358
twice	967	959	944	1129	1021	1118	929	995
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.99	2.98	3.98	4.96	5.95	6.94	7.92

Table 3:66: Results for MO0788 using Node Selection Strategy 3.

Test results using Node Selection Strategy Four

tptrs	1	2	3	4	5	6	7	8
time	3.62	2.91	2.97	2.69	2.64	2.75	2.69	2.69
nodes	7	11	13	13	13	13	13	13
speedup	1.00	1.24	1.22	1.35	1.37	1.32	1.35	1.35
sol	1	2	2	2	2	2	2	2
inf	0	0	0	0	0	0	0	0
cut1	0	1	1	2	1	0	0	0
cut2	3	3	2	0	1	1	1	1
cut3	0	0	1	1	1	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	1	2	2	2	2	2
attack	3	5	6	6	6	6	6	6
never	0	0	0	1	0	0	0	0
once	0	1	1	0	1	0	0	0
twice	3	4	5	5	5	6	6	6
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.8	2.5	2.36	2.58	3	2.85	2.85

Table 3:67: Results for AZA using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	391.67	207.4	161.05	143.96	150.82	145.77	151.76	150.28
nodes	1901	1743	1637	1531	1545	1551	1541	1509
speedup	1.00	1.89	2.43	2.72	2.60	2.69	2.58	2.61
sol	7	4	2	1	1	3	2	2
inf	50	32	28	28	28	29	28	28
cut1	201	160	121	63	69	69	67	43
cut2	693	674	667	674	673	675	674	682
cut3	0	1	1	0	2	0	0	0
cut4	0	0	0	0	0	0	0	0
cut5	0	1	0	0	0	0	0	0
attack	950	871	818	765	772	775	770	754
never	0	57	48	31	34	33	32	20
once	201	46	25	1	1	3	3	3
twice	749	768	745	733	737	739	735	731
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.73	2.27	2.59	2.58	2.65	2.56	2.58

Table 3:68: Results for AZB using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	49.49	26.64	20.43	16.7	15.87	13.95	13.57	14.23
nodes	145	145	143	137	139	141	137	139
speedup	1.00	1.86	2.42	2.96	3.12	3.55	3.65	3.48
sol	3	2	2	2	2	2	2	3
inf	6	6	7	6	6	6	6	6
cut1	4	10	3	3	5	8	5	5
cut2	60	55	59	58	57	54	56	55
cut3	0	0	1	0	0	1	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	72	72	71	68	69	70	68	69
never	0	5	1	1	2	4	2	2
once	4	0	1	1	1	0	1	1
twice	68	67	69	66	66	66	65	66
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.76	2.4	3.05	3.43	4.02	4.38	4.02

Table 3:69: Results for AZC using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	1.81	1.76	1.59	1.04	0.93	0.94	0.93	0.93
nodes	15	23	23	23	23	23	23	23
speedup	1.00	1.03	1.14	1.74	1.95	1.93	1.95	1.95
sol	2	3	3	1	1	1	1	1
inf	2	4	4	3	3	3	3	3
cut1	0	0	0	7	7	7	7	7
cut2	4	3	3	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	1	1	0	0	0	0	0
attack	7	11	11	11	11	11	11	11
never	0	0	0	3	3	3	3	3
once	0	0	0	1	1	1	1	1
twice	7	11	11	7	7	7	7	7
maxtptr	1	2	3	4	5	5	5	5
avtptr	1	1.7	2.04	2.31	2.44	2.44	2.44	2.44

Table 3:70: Results for HPW15 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	28.23	32.79	76.34	48.61	35.16	41.09	47.73	42.56
nodes	99	157	371	245	175	203	243	213
speedup	1.00	0.86	0.37	0.58	0.80	0.69	0.59	0.66
sol	2	2	2	2	2	1	2	2
inf	0	0	0	0	0	0	0	0
cut1	40	62	147	101	72	81	102	86
cut2	8	15	37	18	14	20	18	18
cut3	0	0	0	2	0	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	49	78	185	122	87	101	121	106
never	0	26	66	44	32	35	41	36
once	40	10	15	13	8	11	20	14
twice	9	42	104	65	47	55	60	56
maxtptr	1	2	3	3	3	3	3	3
avtptr	1	1.56	1.73	1.69	1.64	1.69	1.66	1.69

Table 3:71: Results for INGT274 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	1323	709.26	511.96	378.93	319.23	282.26	249.64	223.66
nodes	499	531	571	555	581	597	605	615
speedup	1.00	1.87	2.58	3.49	4.14	4.69	5.30	5.92
sol	3	5	7	7	6	7	6	5
inf	223	235	259	248	258	272	278	284
cut1	1	3	0	2	6	2	1	0
cut2	23	23	20	19	20	18	18	18
cut3	0	0	0	2	1	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	249	265	285	277	290	298	302	307
never	0	1	0	0	0	0	0	0
once	1	1	0	2	6	2	1	0
twice	248	263	285	275	284	296	301	307
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.94	2.87	3.83	4.67	5.55	6.43	7.25

Table 3:72: Results for MRX using Node Selection Strategy 4.

cptrs	1	2	3	4	5	6	7	8
time	1372.2	714.14	489.99	379.26	308.1	274.42	238.37	220.73
nodes	515	537	537	551	543	579	575	575
speedup	1.00	1.89	2.80	3.62	4.45	5.00	5.76	6.22
sol	4	6	7	6	4	4	4	7
inf	234	246	246	251	260	260	267	266
cut1	1	1	0	5	2	4	1	1
cut2	19	16	16	12	15	15	14	17
cut3	0	0	0	1	0	0	6	0
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	1	0	1	2	1
attack	257	268	268	275	271	289	287	287
never	0	0	0	1	1	0	5	3
once	1	1	0	3	0	1	3	1
twice	256	267	268	271	270	288	286	286
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.93	2.85	3.77	4.7	5.49	6.41	7.34

Table 3:73: Results for NR1 using Node Selection Strategy 4.

cptrs	1	2	3	4	5	6	7	8
time	539.8	327.50	250.95	194.88	167.74	180.07	190.77	175.02
nodes	113	125	133	125	147	187	209	215
speedup	1.00	1.65	2.15	2.77	3.22	2.98	2.83	3.07
sol	5	5	7	4	5	5	8	6
inf	8	7	7	4	4	7	3	6
cut1	29	22	21	25	34	56	51	72
cut2	19	26	29	29	27	25	34	35
cut3	0	1	3	1	8	5	7	10
cut4	0	0	0	0	0	0	0	0
cut5	3	2	0	0	0	1	1	1
attack	56	82	66	62	73	98	104	117
never	0	3	0	7	7	13	13	20
once	25	30	21	11	20	30	27	31
twice	31	41	45	44	46	55	64	66
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.93	2.86	3.52	4.56	5.25	5.6	6.54

Table 3:74: Results for CHAL using Node Selection Strategy 4.

cptrs	1	2	3	4	5	6	7	8
time	284.87	311.86	230.85	347.89	351.74	149.06	132.17	382.94
nodes	115	181	255	423	551	271	261	657
speedup	1.00	0.91	1.23	0.82	0.81	1.91	2.15	0.74
sol	6	7	9	7	16	5	7	3
inf	7	8	8	18	19	8	8	33
cut1	31	109	78	84	137	84	88	130
cut2	14	15	31	95	92	36	34	149
cut3	0	1	0	4	7	2	7	8
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	4	3	1	2	2
attack	57	140	127	211	275	135	131	328
never	0	11	15	23	40	27	32	53
once	31	87	48	38	57	30	20	24
twice	26	42	64	150	178	78	79	251
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.9	2.72	3.57	4.23	4.69	5.3	5.16

Table 3:75: Results for OY using Node Selection Strategy 4

tptrs	1	2	3	4	5	6	7	8
time	62.01	421.83	68.1	48.39	47.94	47.73	50.59	47.89
nodes	59	423	81	63	61	61	63	61
speedup	1.00	0.15	0.91	1.28	1.29	1.30	1.23	1.29
sol	2	3	2	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	28	164	36	30	28	28	29	28
cut2	0	45	2	0	0	0	1	0
cut3	0	0	1	1	2	2	1	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	29	211	40	31	30	30	31	30
never	0	63	15	13	12	12	12	12
once	28	38	6	4	4	4	5	4
twice	1	110	19	14	14	14	14	14
maxtptr	1	2	3	4	4	4	4	4
avtptr	1	1.54	1.93	2.12	2.12	2.12	1.97	2.12

Table 3:76: Results for INGT1345 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	15.22	6.1	5.28	5.66	5.33	5.11	4.12	5.5
nodes	111	75	79	109	103	95	89	103
speedup	1.00	2.50	2.88	2.69	2.86	2.98	3.69	2.77
sol	10	3	5	2	2	1	1	1
inf	2	4	8	8	8	6	4	8
cut1	34	24	18	41	38	38	38	41
cut2	10	7	8	2	3	2	0	0
cut3	0	0	1	2	0	1	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	1	0	0	0
attack	55	37	39	54	51	47	44	51
never	0	6	6	19	17	17	18	18
once	34	12	6	3	4	4	2	5
twice	21	19	27	32	30	26	24	28
maxtptr	1	2	3	4	5	6	7	6
avtptr	1	1.75	2.26	2.65	2.85	2.84	2.86	2.81

Table 3:77: Results for DAAC using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	2.97	2.3	2.25	2.25	2.25	2.25	2.25	2.3
nodes	3	3	3	3	3	3	3	3
speedup	1.00	1.29	1.32	1.32	1.32	1.32	1.32	1.29
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	1	0	0	0	0	0	0	0
cut3	0	1	1	1	1	1	1	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	1	1	1	1	1	1	1	1
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	1	1	1	1	1	1	1	1
maxtptr	1	2	2	2	2	2	2	2
avtptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 3:78: Results for G31 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	5.11	3.52	2.8	2.81	2.75	2.8	2.75	2.81
nodes	5	5	5	5	5	5	5	5
speedup	1.00	1.45	1.83	1.82	1.86	1.83	1.86	1.82
sol	1	1	1	1	1	1	1	1
inf	0	0	0	0	0	0	0	0
cut1	0	0	0	0	0	0	0	0
cut2	2	1	0	0	0	0	0	0
cut3	0	1	2	2	2	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	2	2	2	2	2	2	2	2
never	0	0	0	0	0	0	0	0
once	0	0	0	0	0	0	0	0
twice	2	2	2	2	2	2	2	2
maxtptr	1	2	3	3	3	3	3	3
avtptr	1	1.6	1.8	1.8	1.8	1.8	1.8	1.8

Table 3:79: Results for G32 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	127.27	84.29	57.79	59.21	44.16	55.16	58.44	54.1
nodes	735	921	921	1149	765	1105	1157	1049
speedup	1.00	1.51	2.20	2.15	2.88	2.31	2.18	2.35
sol	15	6	3	2	3	1	2	3
inf	12	17	13	20	12	19	23	16
cut1	125	349	331	527	272	508	529	462
cut2	419	190	134	24	96	23	23	42
cut3	0	0	0	2	1	2	2	2
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	367	460	460	574	383	552	578	524
never	0	84	139	236	129	239	247	219
once	125	80	53	55	14	30	35	24
twice	242	296	268	283	240	283	296	281
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.73	2.28	2.6	2.86	2.76	2.8	2.85

Table 3:80: Results for OX using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	1.32	0.82	0.55	0.55	0.49	0.5	0.5	0.44
nodes	31	31	29	27	25	25	25	25
speedup	1.00	1.61	2.40	2.40	2.69	2.64	2.64	3.00
sol	2	1	1	1	1	1	1	1
inf	10	10	8	7	7	8	8	7
cut1	2	1	5	6	8	8	6	6
cut2	2	0	0	0	0	0	0	0
cut3	0	0	0	0	1	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	2	1	0	0	0	0	0
attack	15	15	14	13	12	12	12	12
never	0	0	1	3	2	1	1	2
once	2	1	1	0	0	2	2	0
twice	13	14	12	10	10	9	9	10
maxtptr	1	2	3	4	5	4	4	5
avtptr	1	1.73	2.12	2.19	2.43	2.33	2.33	2.43

Table 3:81: Results for SETX using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	360.15	133.41	93.1	70.09	57.18	64.98	44.05	34.33
nodes	237	201	205	215	203	287	227	215
speedup	1.00	2.70	3.87	5.14	6.30	5.54	8.18	10.49
sol	16	9	7	9	7	11	5	4
inf	0	1	0	2	0	0	0	0
cut1	53	59	60	73	60	99	81	85
cut2	50	29	31	22	31	24	16	11
cut3	0	2	3	1	3	4	10	6
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	1	1	6	2	2
attack	118	100	102	107	101	143	113	107
never	0	2	6	11	9	13	16	28
once	53	55	48	51	42	73	49	29
twice	65	43	48	45	50	57	48	50
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.95	2.85	3.74	4.69	5.59	6.45	6.99

Table 3:82: Results for BAG882 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	1002.1	532.38	281.49	213.93	277.48	247.7	191.36	151.1
nodes	619	661	511	535	781	825	653	607
speedup	1.00	1.88	3.56	4.68	3.61	4.05	5.24	6.63
sol	6	7	7	1	5	6	4	3
inf	219	230	146	138	256	260	201	165
cut1	24	46	72	124	77	103	81	109
cut2	61	45	28	3	48	35	31	15
cut3	0	0	1	2	5	3	7	8
cut4	0	0	0	0	0	0	0	0
cut5	0	3	2	0	0	6	3	4
attack	309	330	255	267	390	412	326	303
never	0	3	9	20	17	32	11	27
once	24	40	54	84	43	39	59	55
twice	285	287	192	163	330	341	256	221
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.95	2.82	3.67	4.54	5.38	6.09	6.99

Table 3:83: Results for TAX1 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	226.35	115.35	80.3	63.66	53.55	48.5	44.22	40.64
nodes	297	295	291	285	287	285	285	285
speedup	1.00	1.96	2.82	3.56	4.23	4.67	5.12	5.57
sol	4	2	2	1	1	1	1	1
inf	128	128	127	125	125	125	125	125
cut1	2	1	1	0	2	0	0	0
cut2	15	17	16	17	16	17	17	16
cut3	0	0	0	0	0	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	0	0	0	0	0	0	0
attack	148	147	145	142	143	142	142	142
never	0	0	0	0	1	0	0	0
once	2	1	1	0	0	0	0	0
twice	146	146	144	142	142	142	142	142
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.89	2.77	3.56	4.38	5.02	5.64	6.24

Table 3:84: Results for TAX2 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	32.84	24.94	23.4	23.51	23.57	23.57	23.61	23.56
nodes	23	33	45	35	31	31	31	31
speedup	1.00	1.32	1.40	1.40	1.39	1.39	1.39	1.39
sol	4	4	7	5	4	4	4	4
inf	4	5	9	6	4	4	4	4
cut1	3	5	3	1	0	0	0	0
cut2	1	1	3	3	3	3	3	3
cut3	0	1	1	3	5	5	5	5
cut4	0	0	0	0	0	0	0	0
cut5	0	1	0	0	0	0	0	0
attack	11	16	22	17	15	15	15	15
never	0	0	0	0	0	0	0	0
once	3	5	3	1	0	0	0	0
twice	8	11	19	16	15	15	15	15
maxtptr	1	2	3	4	5	6	6	6
avtptr	1	1.89	2.6	3.06	3.29	3.35	3.35	3.35

Table 3:85: Results for CRAC using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	197.57	106.72	69.64	53.39	48.78	48.88	48.72	48.77
nodes	31	33	31	29	33	37	41	45
speedup	1.00	1.85	2.84	3.70	4.05	4.04	4.06	4.05
sol	8	9	7	5	6	6	6	6
inf	0	0	0	0	0	0	0	0
cut1	2	2	4	6	6	6	7	7
cut2	6	2	2	1	0	1	1	1
cut3	0	0	1	0	0	0	0	1
cut4	0	0	0	0	0	0	0	0
cut5	0	4	2	3	5	6	7	8
attack	15	16	15	14	16	18	20	22
never	0	0	0	1	1	0	0	1
once	2	2	4	4	4	6	7	5
twice	13	14	11	9	11	12	13	16
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.94	2.74	3.3	4.22	4.97	5.71	6.42

Table 3:86: Results for DOM1 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	281.11	131.22	100.24	74.54	61.85	51.74	45.53	51.74
nodes	115	95	121	97	107	105	105	137
speedup	1.00	2.14	2.80	3.77	4.55	5.43	6.17	5.43
sol	3	3	3	3	3	4	6	3
inf	2	2	2	2	4	2	2	4
cut1	24	17	29	17	24	25	24	42
cut2	29	25	25	23	20	20	18	15
cut3	0	1	1	3	1	2	2	5
cut4	0	0	0	0	0	0	0	0
cut5	0	0	1	1	2	0	1	0
attack	57	47	60	48	53	52	52	68
never	0	0	0	1	4	2	4	7
once	24	17	29	15	16	21	16	28
twice	33	30	31	32	33	29	32	33
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.96	2.91	3.78	4.63	5.4	6.11	7.01

Table 3:87: Results for MCA using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
time	52228	26547	17265	13120	11096	8903.2	6992.1	5971
nodes	3917	3999	3861	3913	4229	3971	3607	3507
speedup	1.00	1.97	3.03	3.98	4.71	5.87	7.47	8.75
sol	12	15	15	16	15	18	15	13
inf	0	0	0	0	0	0	0	0
cut1	628	658	603	612	710	621	531	510
cut2	1319	1325	1306	1319	1382	1341	1247	1214
cut3	0	1	5	8	6	4	9	15
cut4	0	0	0	0	0	0	0	0
cut5	0	1	2	2	2	2	2	2
attack	1958	1999	1930	1956	2114	1985	1803	1753
never	0	2	0	2	4	1	6	5
once	628	654	603	608	702	619	519	500
twice	1330	1343	1327	1346	1408	1365	1278	1248
maxtptr	1	2	3	4	5	6	7	8
avtptr	1	1.99	2.98	3.97	4.96	5.95	6.94	7.93

Table 3:88: Results for MCA using Node Selection Strategy 4.

Appendix 4: Test Results for Code Discussed in Chapter Seven.

This Appendix contains the results of attacking the set of test problems using multiway branching techniques (as discussed in Chapter Five).

KEY

Tptrs	=	number of slave transputers used
Time	=	time in seconds to solve problem
Nodes	=	nodes taken to proven optimal solution

The nodes figure is composed of the following components:

Sol	=	number of integer feasible solutions found
Inf	=	number of infeasible nodes
Cut 1	=	nodes generated to cut off a parent when a new incumbent solution has been found
Cut 2	=	nodes that have worse solutions than the present best and are therefore cut off
Cut 3	=	nodes that are returned to be added to the candidate list but which are no longer eligible because the cutoff has changed since the LP relaxation was sent out and the returning node is not good enough
Cut 4	=	nodes where the LP relaxation has not been solved due to some error in the LP solver on the slaves - the node is cut off in this case and the final solution cannot be proven to be optimal
Cut 5	=	nodes that return a new integer feasible solution to become the incumbent but which cannot because another, better incumbent has been found since the LP relaxation was sent out
Attack	=	number of nodes added to candidate list
Max Tptr	=	maximum number of slave transputers actually used
Av Tptr	=	average number of slave transputers used

Appendix 4A: Test Results for Multiway Branching on General Integers

Test results using multi-way branching code with node selection 1

tptrs	1	2	3	4	5	6	7	8
Time	3.68	2.02	1.7	1.71	1.81	1.65	1.65	1.65
Nodes	38	34	42	45	45	45	45	45
Sol	2	1	1	1	1	1	1	1
Inf	4	3	4	3	3	3	3	3
Cut 1	7	6	12	17	17	17	17	16
Cut 2	15	15	13	11	11	11	11	11
Cut 3	0	0	1	1	1	1	1	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	10	9	11	12	12	12	12	12
Max Tptr	1	2	3	4	4	4	4	4
Av Tptr	1	1.71	2.3	2.32	2.32	2.32	2.32	2.41

Table 4A:1: Results for HPW15 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
Time	27.79	7.97	21.2	21.04	26.09	21.03	21.58	37.57
Nodes	173	78	184	185	203	185	184	279
Sol	2	1	1	1	2	1	1	2
Inf	0	0	0	0	0	0	0	0
Cut 1	116	53	116	120	120	120	116	162
Cut 2	3	2	11	8	17	8	11	28
Cut 3	0	0	0	0	1	0	0	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	52	22	56	56	63	56	56	86
Max Tptr	1	2	3	3	3	3	3	3
Av Tptr	1	1.52	1.75	1.71	1.69	1.72	1.74	1.74

Table 4A:2: Results for INGT274 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
Time	2413.4	1952.6	1340.6	1057.6	833.72	815.54	476.64	711.51
Nodes	699	1092	1111	1108	1049	1148	812	1192
Sol	4	4	5	3	3	5	4	5
Inf	34	56	55	54	50	55	31	56
Cut 1	113	148	167	171	167	190	204	228
Cut 2	305	497	490	485	454	490	292	481
Cut 3	0	0	0	1	3	5	2	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	3	5	7	7	5	3	5
Attack	243	384	389	387	365	398	276	413
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.9	2.73	3.54	4.31	4.82	5.42	5.79

Table 4A:3: Results for GY using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
Time	91.01	81.13	75.41	79.77	86.68	70.02	69.2	68.55
Nodes	117	151	177	190	192	154	165	165
Sol	1	1	1	1	2	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	84	103	122	133	130	104	114	115
Cut 2	0	3	4	2	3	4	2	0
Cut 3	0	1	0	0	2	0	1	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	0	0	0
Attack	32	43	50	54	54	45	47	47
Max Tptr	1	2	3	4	4	4	4	4
Av Tptr	1	1.58	2.13	2.14	2.11	2.12	2.16	2.16

Table 4A:4: Results for INGT1345 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
Time	3.03	2.25	2.25	2.25	2.25	2.31	2.25	2.25
Nodes	3	3	3	3	3	3	3	3
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	1	0	0	0	0	0	0	0
Cut 3	0	1	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	1	1	1	1	1	1	1	1
Max Tptr	1	2	2	2	2	2	2	2
Av Tptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 4A:5: Results for G31 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
Time	9.44	6.1	4.55	3.19	2.97	2.96	2.96	2.97
Nodes	7	7	7	7	7	7	7	7
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	4	3	2	1	0	0	0	0
Cut 3	0	1	2	3	3	3	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	1	1	1
Attack	2	2	2	2	2	2	2	2
Max Tptr	1	2	3	4	5	5	5	5
Av Tptr	1	1.71	2.29	2.71	2.86	2.86	2.86	2.86

Table 4A:6: Results for G32 using Node Selection Strategy 1.

tptrs	1	2	3	4	5	6	7	8
Time	49164	24448	16800	12634	10388	8842.5	7768.7	6777.1
Nodes	2884	2842	3050	3054	3318	3486	3620	3614
Sol	10	9	9	10	10	10	10	11
Inf	0	0	0	0	0	0	0	0
Cut 1	761	730	853	857	1011	1118	1186	1190
Cut 2	1178	1179	1199	1195	1218	1221	1240	1231
Cut 3	0	2	1	3	4	6	12	11
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	2	0	1
Attack	935	922	988	989	1074	1129	1172	1170
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.99	2.98	3.98	4.97	5.96	6.96	7.94

Table 4A:7: Results for MOO788 using Node Selection Strategy 1.

Test results using multi-way branching code with node selection 2

tptrs	1	2	3	4	5	6	7	8
Time	3.73	1.98	1.7	1.7	1.64	1.65	1.59	1.71
Nodes	38	34	42	45	45	45	45	45
Sol	2	1	1	1	1	1	1	1
Inf	4	3	4	3	3	3	3	3
Cut 1	7	6	12	17	17	17	16	17
Cut 2	15	15	13	11	11	11	11	11
Cut 3	0	0	1	1	1	1	2	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	10	9	11	12	12	12	12	12
Max Tptr	1	2	3	4	4	4	4	4
Av Tptr	1	1.75	2.23	2.39	2.43	2.36	2.45	2.32

Table 4A:8: Results for HPW15 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
Time	44.38	7.91	14.34	34	20.81	33.45	15	33.72
Nodes	268	78	117	283	185	280	120	283
Sol	2	1	1	2	1	2	1	2
Inf	0	0	0	0	0	0	0	0
Cut 1	181	53	70	180	120	178	70	180
Cut 2	4	2	11	14	8	14	14	14
Cut 3	0	0	1	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	81	22	34	87	56	86	35	87
Max Tptr	1	2	3	3	3	3	3	3
Av Tptr	1	1.52	1.7	1.73	1.74	1.74	1.74	1.73

Table 4A:9: Results for INGT274 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
Time	2378.1	1936.5	1337.2	1070.9	583.7	807.35	725.4	707.88
Nodes	686	1083	1112	1128	778	1138	1141	1183
Sol	5	4	6	7	6	5	5	5
Inf	34	56	55	55	32	55	53	54
Cut 1	109	149	168	181	174	194	197	230
Cut 2	298	489	488	486	292	482	480	482
Cut 3	0	0	2	1	3	3	7	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	3	3	3	5	2	3	0
Attack	240	382	390	395	266	397	396	410
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.89	2.76	3.52	4.27	4.83	5.45	5.75

Table 4A:10: Results for GY using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
Time	89.64	74.15	69.98	68.6	70.47	70.03	66.9	76.02
Nodes	117	152	168	165	161	156	161	175
Sol	1	1	1	1	1	1	1	2
Inf	0	0	0	0	0	0	0	0
Cut 1	84	106	116	115	111	106	113	121
Cut 2	0	1	2	1	3	2	1	1
Cut 3	0	1	1	1	0	2	0	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	32	43	48	47	46	45	46	50
Max Tptr	1	2	3	4	4	4	4	4
Av Tptr	1	1.63	2.1	2.16	2.14	2.12	2.15	2.09

Table 4A:11: Results for INGT1345 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
Time	3.02	2.26	2.25	2.26	2.25	2.31	2.26	2.25
Nodes	3	3	3	3	3	3	3	3
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	1	0	0	0	0	0	0	0
Cut 3	0	1	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	1	1	1	1	1	1	1	1
Max Tptr	1	2	2	2	2	2	2	2
Av Tptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 4A:12: Results for G31 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
Time	9.45	4.94	4.45	3.18	2.96	3.02	2.97	2.97
Nodes	7	7	7	7	7	7	7	7
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	4	3	2	1	0	0	0	0
Cut 3	0	1	2	3	3	3	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	1	1	1
Attack	2	2	2	2	2	2	2	2
Max Tptr	1	2	3	4	5	5	5	5
Av Tptr	1	1.71	2.29	2.71	2.86	2.86	2.86	2.86

Table 4A:13: Results for G32 using Node Selection Strategy 2.

tptrs	1	2	3	4	5	6	7	8
Time	40491	20288	13614	10297	8292.8	6947.5	5923.9	5302.8
Nodes	1954	1945	2023	2065	2089	2115	2005	2230
Sol	8	7	7	9	9	8	7	10
Inf	0	0	0	0	0	0	0	0
Cut 1	345	338	395	421	438	458	367	538
Cut 2	967	966	962	963	962	959	975	954
Cut 3	0	3	4	4	4	4	6	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	2	0	2
Attack	634	631	655	668	676	684	650	721
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.99	2.99	3.98	4.97	5.96	6.94	7.93

Table 4A:14: Results for MOO788 using Node Selection Strategy 2.

<u>Test results using multi-way branching code with node selection 3</u>

tptrs	1	2	3	4	5	6	7	8
Time	3.74	3.02	2.69	2.52	2.58	2.41	2.7	2.8
Nodes	34	50	66	64	73	69	68	68
Sol	2	3	4	5	4	4	6	6
Inf	4	5	8	10	9	10	10	10
Cut 1	3	8	19	14	21	20	14	14
Cut 2	16	20	15	12	15	11	13	13
Cut 3	0	0	1	2	1	2	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	4	4	4	4	4
Attack	9	13	17	17	19	18	18	18
Max Tptr	1	2	3	4	5	5	5	5
Av Tptr	1	1.74	2.4	2.82	3.13	3.1	2.81	2.81

Table 4A:15: Results for HPW15 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
Time	64.15	41.42	20.87	54.48	40.04	35.48	44.6	50.31
Nodes	366	245	154	470	247	253	338	347
Sol	2	2	2	3	3	3	3	3
Inf	0	0	0	0	0	0	0	0
Cut 1	231	110	90	298	123	148	204	193
Cut 2	17	54	14	33	43	22	32	37
Cut 3	0	0	0	0	0	1	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	116	79	48	136	78	79	99	114
Max Tptr	1	2	3	3	3	3	3	3
Av Tptr	1	1.51	1.73	1.59	1.72	1.63	1.58	1.63

Table 4A:16: Results for INGT274 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
Time	2322.7	1280.4	680.31	519.05	499.43	433.53	425.62	602.04
Nodes	680	779	580	567	700	720	770	1130
Sol	10	13	10	6	8	9	5	3
Inf	34	36	25	24	24	24	26	44
Cut 1	107	157	113	113	160	200	223	282
Cut 2	290	297	226	223	260	234	249	401
Cut 3	0	0	3	3	3	4	5	9
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	3	1	2	7	6	3	5
Attack	239	273	202	196	238	243	259	386
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.92	2.76	3.65	4.42	5.17	5.72	6.12

Table 4A:17: Results for GY using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
Time	52.51	411.5	224.7	172.02	409.2	95.79	1148.5	96.56
Nodes	70	628	393	343	564	195	1296	195
Sol	1	4	4	3	3	2	3	2
Inf	0	0	0	0	0	0	0	0
Cut 1	49	380	241	227	259	124	424	124
Cut 2	0	43	22	8	119	9	448	9
Cut 3	0	0	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	1	0
Attack	20	201	125	104	182	59	419	59
Max Tptr	1	2	3	4	4	3	5	3
Av Tptr	1	1.55	1.81	1.81	1.87	1.69	2.06	1.68

Table 4A:18: Results for INGT1345 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
Time	3.02	2.25	2.25	2.25	2.25	2.26	2.25	2.25
Nodes	3	3	3	3	3	3	3	3
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	1	0	0	0	0	0	0	0
Cut 3	0	1	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	1	1	1	1	1	1	1	1
Max Tptr	1	2	2	2	2	2	2	2
Av Tptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 4A:19: Results for G31 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
Time	9.44	6.09	4.5	3.18	2.96	2.97	3.02	2.97
Nodes	7	7	7	7	7	7	7	7
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	4	3	2	1	0	0	0	0
Cut 3	0	1	2	3	3	3	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	1	1	1
Attack	2	2	2	2	2	2	2	2
Max Tptr	1	2	3	4	5	5	5	5
Av Tptr	1	1.71	2.29	2.71	2.86	2.86	2.86	2.86

Table 4A:20: Results for G32 using Node Selection Strategy 3.

tptrs	1	2	3	4	5	6	7	8
Time	49785	23981	17021	12607	10523	8707.2	7653.4	6092.2
Nodes	2450	2259	2607	2440	2638	2661	2913	2431
Sol	13	11	15	11	13	13	15	16
Inf	0	0	0	0	0	0	0	0
Cut 1	452	363	550	437	532	562	731	486
Cut 2	1193	1151	1198	1200	1235	1219	1221	1131
Cut 3	0	2	1	0	3	7	3	8
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	4	5	3	5	5
Attack	792	731	841	788	850	857	938	785
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.99	2.98	3.98	4.98	5.97	6.96	7.95

Table 4A:21: Results for MO0788 using Node Selection Strategy 3.

<u>Test results using multi-way branching code with node selection 4</u>

tptrs	1	2	3	4	5	6	7	8
Time	3.96	3.18	3.29	2.91	3.57	2.52	2.8	2.59
Nodes	34	54	70	65	77	73	76	73
Sol	2	4	5	5	3	4	4	4
Inf	4	6	9	6	9	10	10	10
Cut 1	0	7	6	11	14	23	21	23
Cut 2	19	20	28	21	28	10	16	10
Cut 3	0	2	1	2	1	2	0	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	3	3	2	5	5	5
Attack	9	14	18	17	20	19	20	19
Max Tptr	1	2	3	4	5	5	6	5
Av Tptr	1	1.81	2.48	2.65	2.59	3.1	2.98	3.1

Table 4A:22: Results for HPW15 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
Time	30.82	71.57	20.65	44.88	21.15	21.26	25.92	20.98
Nodes	187	432	158	258	158	158	178	158
Sol	2	2	2	2	2	2	2	2
Inf	0	0	0	0	0	0	0	0
Cut 1	112	212	91	102	91	91	96	91
Cut 2	13	76	14	72	14	14	23	14
Cut 3	0	1	0	2	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	60	141	51	80	51	51	57	51
Max Tptr	1	2	3	4	3	4	3	3
Av Tptr	1	1.61	1.7	1.86	1.69	1.69	1.72	1.69

Table 4A:23: Results for INGT274 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
Time	349.21	894.79	153.52	194.33	446.77	458.9	432.54	722.44
Nodes	149	495	252	406	638	826	804	1374
Sol	6	7	6	10	6	6	7	11
Inf	4	23	3	6	23	24	27	50
Cut 1	56	72	131	225	171	282	234	367
Cut 2	32	216	26	24	211	227	253	476
Cut 3	0	0	0	2	1	8	6	6
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	2	2	5	6	4	5	3
Attack	51	175	84	134	220	275	272	461
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.91	2.82	3.56	4.39	5.16	5.7	6.07

Table 4A:24: Results for GY using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
Time	156.48	103.64	246.72	185.59	471.1	261.72	282.64	830.63
Nodes	192	154	419	349	708	453	485	986
Sol	2	2	3	3	3	2	3	3
Inf	0	0	0	0	0	0	0	0
Cut 1	114	82	236	218	366	261	277	361
Cut 2	14	20	45	18	112	44	48	303
Cut 3	0	2	1	0	2	1	2	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	0	0	0	0	1
Attack	62	48	133	110	225	145	155	317
Max Tptr	1	2	3	4	4	4	4	4
Av Tptr	1	1.67	1.92	1.81	2.05	1.9	1.91	2.07

Table 4A:25: Results for INGT1345 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
Time	3.02	2.25	2.26	2.25	2.25	2.26	2.25	2.25
Nodes	3	3	3	3	3	3	3	3
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	1	0	0	0	0	0	0	0
Cut 3	0	1	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	1	1	1	1	1	1	1	1
Max Tptr	1	2	2	2	2	2	2	2
Av Tptr	1	1.33	1.33	1.33	1.33	1.33	1.33	1.33

Table 4A:26: Results for G31 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
Time	9.5	6.1	4.56	3.18	2.97	3.02	2.97	2.97
Nodes	7	7	7	7	7	7	7	7
Sol	1	1	1	1	1	1	1	1
Inf	0	0	0	0	0	0	0	0
Cut 1	0	0	0	0	0	0	0	0
Cut 2	4	3	2	1	0	0	0	0
Cut 3	0	1	2	3	3	3	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	1	1	1
Attack	2	2	2	2	2	2	2	2
Max Tptr	1	2	3	4	5	5	5	5
Av Tptr	1	1.71	2.29	2.71	2.86	2.86	2.86	2.86

Table 4A:27: Results for G32 using Node Selection Strategy 4.

tptrs	1	2	3	4	5	6	7	8
Time	66231	33228	22558	17889	14804	12799	10246	8334.7
Nodes	3165	3218	3381	3632	3891	4009	3824	3180
Sol	20	20	18	18	15	16	19	13
Inf	0	0	0	0	0	0	0	0
Cut 1	483	523	604	687	825	829	851	487
Cut 2	1637	1631	1663	1750	1789	1860	1716	1644
Cut 3	0	2	4	4	2	8	6	7
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	2	3	3	4	3
Attack	1025	1041	1091	1171	1257	1293	1228	1026
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.99	2.99	3.98	4.98	5.97	6.96	7.95

Table 4A:28: Results for MO0788 using Node Selection Strategy 4.

Appendix 4B: Test Results for Multiway Branching on Special Ordered Sets

Tptrs	1	2	3	4	5	6	7	8
Time	0.54	0.33	0.27	0.33	0.33	0.28	0.33	0.28
Nodes	13	13	13	13	13	13	13	13
Sol nodes	1	1	1	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	0	0	0	0	0	0	0	0
Cut 2	2	2	1	1	1	1	1	1
Cut 3	0	0	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	4	4	4	4	4	4	4	4
Max tptr	1	2	3	3	3	3	3	3
Av tptr	1	1.54	2	2	2	2	2	2

Table 4B:1: Problem SETX;No branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.54	0.38	0.27	0.33	0.28	0.27	0.28	0.28
Nodes	13	13	13	13	13	13	13	13
Sol nodes	1	1	1	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	0	0	0	0	0	0	0	0
Cut 2	2	2	1	1	1	1	1	1
Cut 3	0	0	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	4	4	4	4	4	4	4	4
Max tptr	1	2	3	3	3	3	3	3
Av tptr	1	1.54	2	2	2	2	2	2

Table 4B:2 Problem SETX;No branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.55	0.33	0.27	0.28	0.33	0.27	0.28	0.32
Nodes	15	13	13	13	13	13	13	13
Sol nodes	2	1	1	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	1	0	0	0	0	0	0	0
Cut 2	1	1	1	1	1	1	1	1
Cut 3	0	1	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	5	4	4	4	4	4	4	4
Max tptr	1	2	3	3	3	3	3	3
Av tptr	1	1.62	2	2	2	2	2	2

Table 4B:3 Problem SETX;No branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.55	0.33	0.27	0.28	0.28	0.28	0.27	0.28
Nodes	15	13	13	13	13	13	13	13
Sol nodes	2	1	1	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	1	0	0	0	0	0	0	0
Cut 2	1	1	1	1	1	1	1	1
Cut 3	0	1	1	1	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	5	4	4	4	4	4	4	4
Max tptr	1	2	3	3	3	3	3	3
Av tptr	1	1.62	2	2	2	2	2	2

Table 4B:4 Problem SETX;No branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1.15	0.66	0.54	0.55	0.44	0.49	0.49	0.49
Nodes	26	26	26	30	30	30	30	30
Sol nodes	1	1	2	2	2	2	2	2
Inf nodes	12	12	12	12	12	12	12	12
Cut 1	0	0	0	4	4	4	4	4
Cut 2	6	6	5	3	3	3	3	3
Cut 3	0	0	0	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	1	1	1	1	1
Attach	7	7	7	8	8	8	8	8
Max tptr	1	2	3	4	4	4	4	4
Av tptr	1	1.73	2.38	2.62	2.62	2.62	2.62	2.62

Table 4B:5 Problem SETX;Branching priorities used;Fixed separation strategy.Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1.09	0.72	0.55	0.49	0.44	0.49	0.44	0.44
Nodes	26	26	26	30	30	30	30	30
Sol nodes	1	1	2	2	2	2	2	2
Inf nodes	12	12	12	12	12	12	12	12
Cut 1	0	0	0	4	4	4	4	4
Cut 2	6	6	5	3	3	3	3	3
Cut 3	0	0	0	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	1	1	1	1	1
Attach	7	7	7	8	8	8	8	8
Max tptr	1	2	3	4	4	4	4	4
Av tptr	1	1.62	2.38	2.69	2.69	2.69	2.69	2.69

Table 4B:6 Problem SETX;Branching priorities used.Fixed separation strategy.Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1.1	0.66	0.55	0.44	0.44	0.44	0.44	0.49
Nodes	26	34	34	38	38	38	38	38
Sol nodes	1	2	2	2	2	2	2	2
Inf nodes	12	12	12	12	12	12	12	12
Cut 1	0	8	8	12	12	12	12	12
Cut 2	6	3	2	2	2	2	2	2
Cut 3	0	0	1	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attach	7	9	9	10	10	10	10	10
Max tptr	1	2	3	4	5	5	5	5
Av tptr	1	1.62	2.23	2.69	2.85	2.85	2.85	2.85

Table 4B:7 Problem SETX;Branching priorities used;Fixed separation strategy.Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1.1	0.66	0.55	0.44	0.5	0.5	0.55	0.55
Nodes	26	26	26	34	34	34	34	34
Sol nodes	1	2	2	2	2	2	2	2
Inf nodes	12	12	12	12	15	15	15	15
Cut 1	0	0	0	8	4	4	4	4
Cut 2	6	5	5	2	4	4	4	4
Cut 3	0	0	0	1	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attach	7	7	7	9	9	9	9	9
Max tptr	1	2	3	4	5	5	5	5
Av tptr	1	1.58	2.23	2.73	2.63	2.63	2.63	2.63

Table 4B:8 Problem SETX;Branching priorities used;Fixed separation strategy.Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.99	0.6	0.55	0.49	0.5	0.49	0.5	0.55
Nodes	21	24	27	31	33	33	33	33
Sol nodes	1	1	2	2	2	2	2	2
Inf nodes	7	7	8	9	8	8	11	11
Cut 1	2	4	5	8	11	10	7	7
Cut 2	4	3	1	0	0	0	0	0
Cut 3	0	0	0	0	0	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	1	1	1	1	1
Attack	7	8	9	10	11	11	11	11
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.6	2	2.39	2.5	2.65	2.96	2.96

Table 4B:9 Problem SETX,No branching priorities used,Beale and Forrest separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.94	0.66	0.55	0.44	0.49	0.55	0.55	0.55
Nodes	21	24	29	32	32	30	35	35
Sol nodes	1	1	2	2	2	2	2	2
Inf nodes	7	7	8	8	12	12	12	12
Cut 1	2	4	8	12	4	3	8	8
Cut 2	4	3	1	0	2	1	0	0
Cut 3	0	0	0	0	0	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	0	1	1	0	0
Attack	7	8	10	11	10	10	12	12
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.5	2.1	2.57	2.42	2.56	2.93	2.93

Table 4B:10 Problem SETX;No branching priorities used;Beale and Forrest separation strategy.Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.99	0.66	0.71	0.6	0.6	0.66	0.71	0.66
Nodes	21	26	33	38	38	38	38	38
Sol nodes	1	3	1	3	2	3	3	3
Inf nodes	7	10	18	17	17	18	18	18
Cut 1	2	2	2	8	5	1	1	1
Cut 2	4	2	2	0	0	1	1	1
Cut 3	0	0	0	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	1	1	2	2	2
Attack	7	9	11	13	13	13	13	13
Max tptr	1	2	3	4	5	6	8	8
Av tptr	1	1.62	2.03	2.38	2.64	2.68	2.68	2.68

Table 4B:11 Problem SETX,No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.99	0.72	0.66	0.6	0.55	0.66	0.65	0.66
Nodes	21	26	33	38	36	38	38	38
Sol nodes	1	3	1	2	2	3	3	3
Inf nodes	7	10	18	15	13	18	18	18
Cut 1	2	2	2	7	8	2	2	2
Cut 2	4	2	2	0	0	1	1	1
Cut 3	0	0	0	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	1	1	1	1	1
Attack	7	9	11	11	12	13	13	13
Max tptr	1	2	3	4	5	6	6	6
Av tptr	1	1.62	2.03	2.32	2.71	2.78	2.78	2.78

Table 4B:12 Problem SETX;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.87	0.6	0.55	0.55	0.55	0.55	0.55	0.55
Nodes	21	26	27	29	29	29	29	29
Sol nodes	1	1	2	3	3	3	3	3
Inf nodes	7	8	9	9	9	9	9	9
Cut 1	2	4	5	4	4	4	4	4
Cut 2	4	3	1	1	1	1	1	1
Cut 3	0	0	0	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	2	2	2	2	2
Attack	7	8	9	10	10	10	10	10
Max tptr	1	2	3	4	4	4	4	4
Av tptr	1	1.77	1.95	2.4	2.4	2.4	2.4	2.4

Table 4B:13 Problem SETX;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 1 used

Tptrs	1	2	3	4	5	6	7	8
Time	0.88	0.61	0.55	0.62	0.55	0.55	0.55	0.55
Nodes	21	26	27	29	29	29	29	29
Sol nodes	1	1	2	3	3	3	3	3
Inf nodes	7	8	8	9	9	9	9	9
Cut 1	2	4	5	4	4	4	4	4
Cut 2	4	3	1	1	1	1	1	1
Cut 3	0	0	1	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	2	2	2	2	2
Attack	7	8	9	10	10	10	10	10
Max tptr	1	2	3	4	4	4	4	4
Av tptr	1	1.88	2	2.38	2.4	2.4	2.4	2.4

Table 4B:14 Problem SETX;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.94	0.6	0.55	0.6	0.55	0.54	0.61	0.6
Nodes	21	26	32	29	29	29	29	29
Sol nodes	1	2	2	2	2	2	2	2
Inf nodes	7	9	10	10	11	11	11	11
Cut 1	2	2	9	3	0	0	0	0
Cut 2	4	2	0	1	3	3	3	3
Cut 3	0	1	0	0	0	0	0	0
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	3	3	3	3	3
Attack	7	9	11	10	10	10	10	10
Max tptr	1	2	3	4	4	4	4	4
Av tptr	1	1.75	2.04	2.27	2.59	2.59	2.59	2.59

Table 4B:15 Problem SETX;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	0.87	0.66	0.55	0.55	0.49	0.55	0.55	0.55
Nodes	21	26	32	29	29	29	29	29
Sol nodes	1	2	2	2	2	2	2	2
Inf nodes	7	8	10	10	11	11	10	11
Cut 1	2	3	9	3	0	0	2	0
Cut 2	4	3	0	1	2	2	2	1
Cut 3	0	0	0	0	1	1	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	3	3	3	2	3
Attack	7	9	11	10	10	10	10	10
Max tptr	1	2	3	4	4	4	4	4
Av tptr	1	1.88	2.04	2.5	2.62	2.62	2.48	2.62

Table 4B:16 Problem SETX;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	77.83	41.52	28.84	29.5	26.26	25.6	21.2	21.04
Nodes	16	16	19	16	16	16	10	10
Sol nodes	3	2	2	2	2	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	6	6	9	6	5	3	0	0
Cut 2	2	2	0	0	1	1	0	0
Cut 3	0	0	1	2	2	4	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	1	1	1	2	2
Attack	5	5	6	5	5	5	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.8	2.5	3	3.55	4.15	3.7	3.7

Table 4B:17 Problem DOM1;No branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	77.83	41.52	28.83	31.75	26.3	25.59	21.2	21.04
Nodes	16	16	19	16	16	16	10	10
Sol nodes	3	2	2	2	2	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	6	6	9	5	5	3	0	0
Cut 2	2	2	0	2	1	1	0	0
Cut 3	0	0	1	1	2	4	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	1	1	1	2	2
Attack	5	5	6	5	5	5	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.8	2.5	3.09	3.55	4.15	3.7	3.7

Table 4B:18 Problem DOM1;No branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	77.95	56.25	56.3	38.33	29.39	24.99	21.2	21.09
Nodes	16	19	31	16	19	16	10	10
Sol nodes	3	3	3	3	3	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	6	7	12	4	6	3	0	0
Cut 2	2	1	2	1	1	0	0	0
Cut 3	0	2	0	3	3	5	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	4	0	0	1	2	2
Attack	5	6	10	5	6	5	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.83	2.74	3.17	3.77	4.15	3.7	3.7

Table 4B:19 Problem DOM1;No branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	119.96	62.41	57.51	38.34	29.27	25.16	21.04	21.2
Nodes	19	19	25	16	19	16	10	10
Sol nodes	7	7	4	3	3	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	2	2	7	4	6	3	0	0
Cut 2	4	1	3	1	1	0	0	0
Cut 3	0	1	0	3	3	5	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	2	3	0	0	1	2	2
Attack	6	6	8	5	6	5	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.88	2.67	3.17	3.77	4.15	3.7	3.7

Table 4B:20 Problem DOM1;No branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	86.45	44	42.85	34.2	33.07	33.06	22.08	22.14
Nodes	16	16	19	16	16	13	10	10
Sol nodes	3	3	3	2	2	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	6	6	8	6	5	0	0	0
Cut 2	2	1	1	0	1	2	0	0
Cut 3	0	1	1	2	2	4	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	1	1	1	2	2
Attack	5	5	6	5	5	4	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.8	2.45	3	3.55	3.92	3.7	3.7

Table 4B:21 Problem DOM1;Branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	86.55	43.98	42.84	37.35	33.12	33.02	22.14	22.13
Nodes	16	16	19	16	16	13	10	10
Sol nodes	3	3	3	2	2	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	6	6	8	5	5	0	0	0
Cut 2	2	1	1	2	1	2	0	0
Cut 3	0	1	1	1	2	4	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	1	1	1	2	2
Attack	5	5	6	5	5	4	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.8	2.45	3.09	3.55	3.92	3.7	3.7

Table 4B:22 Problem DOM1;Branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	86.4	56.79	53.45	37.8	35.7	33.05	22.14	22.14
Nodes	16	19	22	16	19	13	10	10
Sol nodes	3	3	5	3	3	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	6	8	6	4	5	0	0	0
Cut 2	2	2	1	2	3	2	0	0
Cut 3	0	0	2	2	2	4	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	0	0	1	2	2
Attack	5	6	7	5	6	4	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.82	2.62	3.08	3.79	3.93	3.7	3.7

Table 4B:23 Problem DOM1;Branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	129.73	70.36	55.58	37.8	33.13	33.06	22.08	22.13
Nodes	19	19	22	16	19	13	10	10
Sol nodes	7	5	4	3	3	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	2	3	3	4	5	0	0	0
Cut 2	4	4	3	2	3	2	0	0
Cut 3	0	1	2	2	2	4	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	3	0	0	1	2	2
Attack	6	6	7	5	6	4	3	3
Max tptr	1	2	3	4	5	6	7	7
Av tptr	1	1.87	2.68	3.08	3.79	3.92	3.7	3.7

Table 4B:24 Problem DOM1;Branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	55.53	32.18	30.32	21.31	21.26	20.92	20.93	20.87
Nodes	19	22	22	16	16	16	16	16
Sol nodes	2	1	1	1	1	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	10	12	12	6	6	5	4	3
Cut 2	1	1	0	1	0	0	0	0
Cut 3	0	0	0	1	2	3	4	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	2	2	2	2	2
Attack	6	7	7	5	5	5	5	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.8	2.5	3	3.4	3.82	4.17	4.54

Table 4B:25 Problem DOM1;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	55.37	32.19	30.27	21.36	21.26	20.98	20.87	20.87
Nodes	19	22	22	16	16	16	16	16
Sol nodes	2	1	1	1	1	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	10	12	12	6	6	5	4	3
Cut 2	1	1	0	1	0	0	0	0
Cut 3	0	0	0	1	2	3	4	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	2	2	2	2	2
Attack	6	7	7	5	5	5	5	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.8	2.5	3	3.4	3.82	4.17	4.54

Table 4B:26 Problem DOM1;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	55.25	45.53	47.89	31.2	34.61	20.1	20	19.99
Nodes	19	25	31	31	31	16	16	16
Sol nodes	2	3	4	2	2	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	10	12	15	16	14	5	4	3
Cut 2	1	1	0	0	0	0	0	0
Cut 3	0	0	0	1	0	4	5	6
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	2	5	1	1	1
Attack	6	8	10	10	10	5	5	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.62	3.33	4	3.82	4.17	4.54

Table 4B:27 Problem DOM1;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	112.55	65.48	41.14	31.64	34.72	19.99	20.11	19.98
Nodes	25	28	25	25	28	16	16	16
Sol nodes	5	6	2	2	2	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	8	9	10	11	12	5	4	3
Cut 2	4	4	2	1	0	0	0	0
Cut 3	0	0	0	0	0	4	5	6
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	3	3	5	1	1	1
Attack	8	9	8	8	9	5	5	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.89	2.6	3.29	3.94	3.82	4.17	4.54

Table 4B:28 Problem DOM1;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	62.78	30.89	35.31	20.90	20.98	24.82	24.88	22.47
Nodes	19	22	22	16	16	19	19	16
Sol nodes	2	1	1	1	1	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	10	12	12	7	6	7	6	3
Cut 2	1	1	0	0	0	0	0	0
Cut 3	0	0	0	2	3	3	4	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	1	1	2	2	2
Attack	6	7	7	5	5	6	6	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.8	2.3	2.89	3.4	4	4.46	4.62

Table 4B:29 Problem DOM1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	62.78	30.94	35.31	21.09	20.98	24.89	24.88	22.3
Nodes	19	22	22	16	16	19	19	19
Sol nodes	2	1	1	1	1	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	10	12	12	7	6	7	6	6
Cut 2	1	1	0	0	0	0	0	0
Cut 3	0	0	0	2	3	3	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	1	1	2	2	2
Attack	6	7	7	5	5	6	6	6
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.8	2.3	2.89	3.4	4	4.46	4.62

Table 4B:30 Problem DOM1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	62.78	50.25	50.26	33.56	35.04	35.04	35.1	20.98
Nodes	19	25	31	31	34	34	37	16
Sol nodes	2	3	5	2	2	2	2	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	10	12	14	16	17	16	17	3
Cut 2	1	1	0	0	0	0	0	0
Cut 3	0	0	1	1	0	1	0	6
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	2	4	4	6	1
Attack	6	8	10	10	11	11	12	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.72	3.33	4.08	4.67	5.35	4.62

Table 4B:31 Problem DOM1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	121.76	72.49	44.27	36.52	35.05	35.04	35.05	20.92
Nodes	25	28	25	25	28	31	34	16
Sol nodes	5	6	6	2	2	2	3	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	8	8	8	11	13	14	15	3
Cut 2	4	4	0	1	0	0	0	0
Cut 3	0	0	1	0	0	0	0	6
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	2	3	4	5	6	1
Attack	8	9	8	8	9	10	11	5
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.86	2.71	3.29	3.93	4.59	5.26	4.62

Table 4B:32 Problem DOM1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	287.87	139.95	103.92	74.48	60.64	54.81	49.6	44.87
Nodes	127	123	127	123	119	121	119	117
Sol nodes	2	2	2	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	64	62	64	62	59	58	56	55
Cut 2	4	2	2	2	2	3	3	2
Cut 3	0	0	0	0	1	3	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	2	2	3	3	2	2	3
Attack	51	49	51	49	47	48	47	46
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.93	2.9	3.75	4.57	5.44	6.3	7.05

Table 4B:33 Problem MCA;No branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	289.95	145.45	100.18	74.65	60.58	54.76	49.38	45.15
Nodes	127	127	123	125	123	121	119	117
Sol nodes	3	3	1	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	62	62	62	63	62	58	57	55
Cut 2	5	2	3	2	2	3	2	2
Cut 3	0	0	0	0	0	3	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	3	2	3	3	2	2	3
Attack	51	51	49	50	49	48	47	46
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.97	2.89	3.77	4.56	5.46	6.32	6.98

Table 4B:34 Problem MCA;No branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	378.44	286.71	271.34	172.79	64.92	63.66	49.05	44.16
Nodes	105	161	182	138	82	101	94	87
Sol nodes	4	5	11	3	3	2	3	2
Inf nodes	8	15	12	8	6	7	6	6
Cut 1	22	52	56	44	14	24	20	19
Cut 2	31	23	23	26	23	28	21	18
Cut 3	0	0	2	2	4	3	5	4
Cut 4	0	1	3	2	0	0	0	0
Cut 5	0	0	0	0	0	0	3	3
Attack	40	65	75	53	32	37	36	35
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.94	2.92	3.79	4.68	5.6	6.46	7.13

Table 4B:35 Problem MCA;No branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	285.34	144.95	102.22	76.29	63.33	55.53	53.55	47.84
Nodes	63	67	71	69	74	74	84	89
Sol nodes	2	3	3	3	4	4	3	4
Inf nodes	6	6	7	6	6	6	6	6
Cut 1	0	1	3	3	7	6	12	15
Cut 2	32	31	30	29	28	29	26	26
Cut 3	0	0	0	1	1	1	2	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	1	0	0	3	2
Attack	23	25	27	26	28	28	32	34
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.97	2.88	3.8	4.66	5.49	6.37	7.22

Table 4B:36 Problem MCA;No branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	81.9	36.09	31.31	23.29	23.46	29.44	29.05	29.16
Nodes	41	39	47	49	49	43	49	49
Sol nodes	2	1	3	2	2	2	2	3
Inf nodes	2	1	3	2	4	5	6	4
Cut 1	15	19	15	17	13	4	8	8
Cut 2	6	2	7	6	9	11	11	10
Cut 3	0	0	0	0	0	2	1	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	2	1	1	1	1
Attack	16	15	19	20	20	18	20	20
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.7	2.75	3.62	4.33	5.33	5.98	6.54

Table 48:37 Problem MCA.Branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	81.95	36.19	29.77	23.38	23.35	29.77	28.11	28.45
Nodes	41	39	45	45	49	43	49	49
Sol nodes	2	1	3	2	2	2	2	2
Inf nodes	2	1	3	3	4	5	5	5
Cut 1	15	19	18	15	13	4	9	8
Cut 2	6	2	7	5	9	11	11	10
Cut 3	0	0	0	0	0	2	1	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	2	1	1	1	1
Attack	16	15	18	18	20	18	20	20
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.7	2.74	3.63	4.31	5.33	5.95	6.47

Table 48:38 Problem MCA.Branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	102.68	64.94	35.15	40.81	28.51	30.7	31.62	36.58
Nodes	71	52	57	62	52	52	58	70
Sol nodes	4	2	3	2	3	3	3	4
Inf nodes	5	3	3	5	3	4	6	7
Cut 1	24	17	21	15	13	11	13	21
Cut 2	9	8	7	8	10	10	10	6
Cut 3	0	1	1	1	2	3	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	1	0	0	0	0
Attack	29	20	22	20	21	21	23	29
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.89	2.75	3.54	4.15	5.17	6.04	6.89

Table 48:39 Problem MCA.Branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	81.4	41.2	28.56	31.74	26.42	29.71	28.91	29.11
Nodes	26	26	34	43	51	61	57	54
Sol nodes	2	2	3	3	3	3	3	3
Inf nodes	2	2	3	3	4	4	5	6
Cut 1	0	0	7	9	14	11	14	12
Cut 2	11	10	9	9	8	10	10	8
Cut 3	0	0	6	1	1	2	2	3
Cut 4	0	0	6	0	0	0	0	0
Cut 5	0	1	1	0	0	0	0	0
Attack	11	11	14	18	21	21	23	22
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.92	2.74	3.32	4.08	5.05	5.7	6.48

Table 48:40 Problem MCA.Branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	288.41	142.52	104.31	76.12	61.46	54.98	48.94	45.31
Nodes	125	123	125	125	119	119	115	117
Sol nodes	2	2	2	1	1	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	62	61	62	62	58	56	54	54
Cut 2	4	2	2	2	2	3	2	2
Cut 3	0	0	0	0	1	3	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	2	2	3	3	2	2	3
Attack	51	50	51	51	48	48	46	47
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.94	2.87	3.79	4.52	5.49	6.28	7.14

Table 4B:41 Problem MCA;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	290.98	146.54	104.14	76.13	63.27	54.1	49.8	45.27
Nodes	125	125	125	125	125	115	117	117
Sol nodes	3	4	2	1	3	1	1	1
Inf nodes	6	6	6	6	6	6	6	6
Cut 1	60	59	60	62	60	54	55	54
Cut 2	5	3	3	2	2	3	2	2
Cut 3	0	0	0	0	0	3	4	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	2	3	3	3	2	2	3
Attack	51	51	51	51	51	46	47	47
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.91	2.91	3.73	4.58	5.44	6.31	7.19

Table 4B:42 Problem MCA;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	382.88	234.48	209.43	83.48	74.42	59.92	51.36	51.79
Nodes	109	124	157	86	96	98	96	117
Sol nodes	4	5	6	2	2	2	3	2
Inf nodes	9	9	12	7	7	7	6	8
Cut 1	23	31	49	12	18	24	20	36
Cut 2	31	29	24	30	30	24	24	22
Cut 3	0	0	1	3	3	3	5	4
Cut 4	0	1	2	0	0	0	0	0
Cut 5	0	0	0	0	0	0	1	0
Attack	42	49	63	32	36	38	37	45
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.97	2.94	3.82	4.74	5.58	6.43	7.36

Table 4B:43 Problem MCA;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	285.18	145.22	102.38	75.85	63.99	60.14	49.66	49.54
Nodes	63	67	71	69	69	84	74	79
Sol nodes	2	3	3	3	3	3	4	4
Inf nodes	6	6	7	6	6	6	6	6
Cut 1	0	1	3	3	3	12	6	8
Cut 2	32	31	30	30	30	26	29	30
Cut 3	0	0	0	1	1	2	1	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	0	0	3	0	0
Attack	23	25	27	26	26	32	28	30
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.95	2.87	3.79	4.65	5.54	6.38	7.28

Table 4B:44 Problem MCA;No branching priorities used;Beale and Forrest separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	82.17	36.2	31.26	25.49	23.34	29.83	29.22	28.94
Nodes	41	39	47	49	49	43	46	46
Sol nodes	2	1	3	2	2	2	2	2
Inf nodes	2	1	3	3	4	3	6	6
Cut 1	15	19	15	15	13	4	5	5
Cut 2	6	2	7	7	9	11	11	10
Cut 3	0	0	0	0	0	3	2	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	2	1	1	1	1
Attach	16	15	18	20	20	18	19	19
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.7	2.75	3.68	4.25	5.33	6	6.56

Table 4B:45 Problem MCA;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	82.16	36.36	29.28	23.44	29.59	28.93	29.11	29.11
Nodes	41	39	45	45	49	43	46	46
Sol nodes	2	1	3	2	2	2	2	2
Inf nodes	2	1	2	3	3	5	6	6
Cut 1	15	19	15	14	13	4	5	5
Cut 2	6	2	7	6	10	11	11	10
Cut 3	0	0	0	0	0	2	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	2	1	1	1	1
Attach	16	15	18	18	20	18	19	19
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.7	2.7	3.65	4.62	5.33	6	6.56

Table 4B:46 Problem MCA;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	182.35	64.76	34.33	40.86	29.22	31.69	31.64	32.13
Nodes	71	51	57	51	52	50	55	53
Sol nodes	4	2	3	2	3	3	3	2
Inf nodes	5	3	3	5	4	4	5	6
Cut 1	26	16	21	14	13	11	10	9
Cut 2	9	9	7	8	9	9	16	9
Cut 3	0	1	1	1	2	3	5	6
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	1	0	0	0	0
Attach	29	22	22	20	21	20	22	22
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.89	2.81	3.54	4.28	5.15	6.13	6.76

Table 4B:47 Problem MCA;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	81.68	41.25	28.67	31.75	24.33	29.66	30.92	29.44
Nodes	24	26	34	43	51	51	57	54
Sol nodes	2	2	1	3	3	3	3	3
Inf nodes	2	2	2	3	3	4	6	6
Cut 1	0	2	7	9	16	11	13	10
Cut 2	11	10	9	9	6	11	12	9
Cut 3	0	0	0	1	2	1	2	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	0	0	1	0	0
Attach	11	11	14	18	21	22	23	22
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.92	2.74	3.32	4.16	5.17	5.93	6.52

Table 4B:48 Problem MCA;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	108.69	63.71	44.21	39.33	38.78	38.67	38.62	39.77
Nodes	121	149	155	189	211	211	231	237
Sol nodes	2	2	2	1	2	2	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	18	34	43	70	76	78	93	81
Cut 2	42	39	33	23	26	26	21	22
Cut 3	0	1	1	3	3	1	2	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	1	0
Attack	59	73	76	93	104	104	115	117
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.84	3.3	3.83	4.08	4.3	4.39

Table 48:49 Problem MINE1;No branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	105.29	52.62	44.11	38.65	37.74	39.42	34.55	37.13
Nodes	121	126	155	175	201	217	203	229
Sol nodes	2	1	2	1	1	1	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	21	33	44	64	78	83	76	90
Cut 2	38	29	31	27	23	21	20	21
Cut 3	0	1	2	3	8	5	5	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	59	62	76	88	89	107	101	113
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.49	2.63	3.22	3.73	4.06	4.43	4.42

Table 48:50 Problem MINE1;No branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	111.94	61.74	132.92	69.53	45.1	44.33	43.01	51.79
Nodes	121	141	527	327	251	265	249	333
Sol nodes	2	2	7	3	1	2	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	16	28	187	124	106	110	114	139
Cut 2	44	41	69	37	18	21	19	23
Cut 3	0	1	2	2	4	2	2	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	1
Attack	59	99	262	161	126	130	133	158
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.83	2.94	3.6	3.63	4.08	4.26	4.45

Table 48:51 Problem MINE1;No branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	245.63	123.47	50.31	43.61	42.51	41.28	42.34	44.82
Nodes	373	359	189	211	225	263	251	283
Sol nodes	12	6	1	1	2	1	2	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	123	123	69	78	87	114	109	125
Cut 2	53	47	28	28	22	16	14	17
Cut 3	0	0	1	2	8	2	3	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	1	0	0	0	0
Attack	185	176	92	103	112	129	124	139
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.62	3.33	3.77	4.1	4.15	4.34

Table 48:52 Problem MINE1;No branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	194.11	100.07	68.06	58.16	53.05	49.21	46.74	46.85
Nodes	201	190	217	238	246	261	268	278
Sol nodes	3	2	2	2	2	2	2	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	31	35	50	62	63	71	76	82
Cut 2	86	77	77	76	77	76	74	74
Cut 3	0	1	1	1	4	5	5	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	81	75	87	97	100	107	111	115
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.81	2.77	3.45	4.08	4.59	5.05	5.27

Table 4B:53 Problem MINE1;Branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	189.6	93.92	68.06	53.27	47.68	45.75	43.88	42.95
Nodes	201	193	215	209	228	254	247	255
Sol nodes	3	2	2	1	1	1	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	39	37	49	57	66	83	75	81
Cut 2	78	77	76	66	65	65	68	64
Cut 3	0	1	2	2	4	1	2	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	81	76	86	83	92	104	101	104
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.89	2.77	3.35	4.06	4.67	4.95	5.13

Table 4B:54 Problem MINE1;Branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	243.43	140.55	106.22	73.93	52.18	66.46	54.98	48
Nodes	251	294	379	305	258	370	343	293
Sol nodes	5	4	3	2	1	6	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	28	47	114	84	87	121	131	111
Cut 2	113	115	98	89	65	85	64	58
Cut 3	0	3	2	4	2	4	5	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	0	0	0	0	0
Attack	105	125	161	126	103	154	142	118
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.91	2.65	3.32	3.9	4.4	4.89	4.91

Table 4B:55 Problem MINE1;Branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	272.21	138.36	83.26	66.4	77.72	62.45	55.31	60.8
Nodes	363	371	310	276	406	354	323	379
Sol nodes	11	7	2	2	2	4	2	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	102	114	112	96	147	117	109	143
Cut 2	87	80	64	62	79	78	73	73
Cut 3	0	2	2	2	3	4	5	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	0	0	0	0	0
Attack	163	167	130	114	175	151	134	159
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.92	2.6	3.27	3.81	4.48	4.77	4.8

Table 4B:56 Problem MINE1;Branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	96.23	56.74	37.78	35.45	34.11	36.14	36.2	36.09
Nodes	110	139	149	160	189	214	222	226
Sol nodes	2	3	1	2	2	1	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	18	34	55	67	73	88	92	94
Cut 2	38	35	19	22	20	16	18	24
Cut 3	0	1	2	2	3	5	4	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	1	0
Attack	52	66	71	77	91	104	108	107
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.9	2.59	3.08	3.44	4	4.29	4.47

Table 48:57 Problem MINE1:No branching priorities used;Beale and Forrest separation strategy:Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	91.03	56.85	34.22	36.71	36.2	36.36	34.94	35.32
Nodes	110	139	132	162	198	219	216	228
Sol nodes	2	3	1	2	1	2	3	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	21	34	47	56	81	92	88	98
Cut 2	35	35	20	24	16	15	21	17
Cut 3	0	1	1	2	3	3	5	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	0	0	1
Attack	52	66	63	90	96	106	105	111
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.88	2.66	3.25	3.60	4.01	4.45	4.48

Table 48:58 Problem MINE1/No branching priorities used;Beale and Forrest separation strategy:Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	89.44	60.32	83.1	49.87	45.2	41.41	43.94	43.12
Nodes	110	168	334	269	256	255	284	273
Sol nodes	2	2	4	2	1	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	16	48	115	111	112	113	128	121
Cut 2	40	36	49	23	14	12	11	14
Cut 3	0	1	3	2	2	3	5	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	1	0	0
Attack	52	91	163	131	123	124	139	133
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.89	2.54	3.3	3.6	3.92	4.16	4.18

Table 48:59 Problem MINE1:No branching priorities used.Beale and Forrest separation strategy:Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	227.5	89.75	57.24	46.85	44.38	52.37	59.82	35.12
Nodes	350	260	252	250	263	329	320	235
Sol nodes	12	3	2	1	2	2	3	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	115	90	98	103	108	145	129	107
Cut 2	51	37	26	23	21	18	30	11
Cut 3	0	2	3	2	4	3	2	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	0	0	0	0	0
Attack	172	127	122	132	128	161	158	114
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.64	3.31	3.83	4.16	3.78	4.41

Table 48:60 Problem MINE1:No branching priorities used;Beale and Forrest separation strategy:Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	140.67	69.75	49.48	44.21	43.06	41.48	40.86	41.31
Nodes	185	185	171	201	205	214	214	213
Sol nodes	3	2	2	2	2	2	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	30	39	40	59	60	65	69	67
Cut 2	58	50	51	48	48	48	45	44
Cut 3	0	1	2	2	3	5	3	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	74	73	78	90	92	96	95	95
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.88	2.69	3.18	3.7	4.24	4.25	4.35

Table 48:61 Problem MINE1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	133.9	60.08	48.72	37.57	38.67	33.23	33.12	38.82
Nodes	165	181	171	171	180	189	197	208
Sol nodes	3	1	2	1	1	1	1	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	38	35	41	57	67	62	73	66
Cut 2	50	42	51	36	38	37	31	44
Cut 3	0	1	1	1	2	5	5	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	78	63	76	76	88	94	87	91
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.92	2.73	3.32	3.78	4.43	4.6	4.39

Table 48:62 Problem MINE1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	192.07	256.95	80.25	55.3	38.72	52.45	50.59	41.14
Nodes	212	899	319	243	204	250	283	285
Sol nodes	5	12	2	2	1	2	1	1
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	24	203	107	78	72	110	109	129
Cut 2	88	148	60	49	36	43	43	27
Cut 3	0	8	3	1	3	4	0	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	97	328	144	109	92	131	128	127
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.81	2.63	3.21	3.88	4.03	4.17	4.52

Table 48:63 Problem MINE1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	250.4	156.68	101.55	61.73	42.48	54.87	48.56	50.3
Nodes	379	525	611	334	218	321	291	301
Sol nodes	12	8	7	3	3	2	3	3
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	121	217	347	140	64	120	112	119
Cut 2	69	45	67	31	49	48	43	40
Cut 3	0	3	6	5	3	4	2	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	0	0	0	0	0
Attack	177	252	204	155	96	147	131	134
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.86	2.48	3.35	3.9	4.08	4.14	4.42

Table 48:64 Problem MINE1;Branching priorities used;Beale and Forrest separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	84.03	50.37	33.01	31.42	32.19	31.86	33.23	32.52
Nodes	101	129	131	159	185	201	219	213
Sol	2	2	2	2	2	2	1	1
Inf	0	0	0	0	0	0	0	0
Cut I	18	35	45	65	75	84	96	92
Cut II	31	27	18	12	14	15	10	9
Cut III	0	1	1	1	2	0	2	5
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	0	0	0	0	0	1	0
Attack	50	64	65	79	92	100	109	106
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.87	2.66	3.23	3.85	4.22	4.35	4.43

Table 4B:65 Problem MINE1;No branching priorities used;binary separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	81.4	50.81	34.11	31.58	33.45	33.01	32.74	33.73
Nodes	101	129	143	161	195	205	207	217
Sol	2	2	1	1	2	2	2	2
Inf	0	0	0	0	0	0	0	0
Cut I	21	34	55	65	82	86	89	95
Cut II	28	28	14	12	11	11	11	9
Cut III	0	1	2	3	3	4	2	3
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	0	0	0	0	0	0	0
Attack	50	64	71	80	97	102	103	108
Max Tptr	1	2	3	4	5	6	7	7
Av Tptr	1	1.86	2.64	3.31	3.87	4.22	4.28	4.36

Table 4B:66 Problem MINE1;No branching priorities used;binary separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	86.07	53.99	35.98	51.02	44.33	35.26	42.73	43.39
Nodes	101	147	145	223	271	223	279	273
Sol	2	2	1	3	1	1	2	2
Inf	0	0	0	0	0	0	0	0
Cut I	16	42	48	81	126	104	128	120
Cut II	33	29	21	25	7	5	10	11
Cut III	0	1	2	3	2	2	0	4
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	0	0	0	0	0	0	0
Attack	50	73	73	111	135	111	139	136
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.92	2.66	3	3.79	3.92	4.13	4.29

Table 4B:67 Problem MINE1;No branching priorities used;binary separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	215.42	114.46	62.84	51.68	41.75	43.11	35.86	44.21
Nodes	345	369	271	229	247	273	235	277
Sol	12	5	4	2	2	2	1	2
Inf	0	0	0	0	0	0	0	0
Cut I	120	150	99	81	103	124	107	123
Cut II	41	26	29	30	16	9	7	11
Cut III	0	2	3	2	3	2	3	3
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	2	1	0	0	0	0	0
Attack	172	184	135	114	123	136	117	138
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.84	2.68	3.09	3.82	4.04	4.26	4.23

Table 4B:68 Problem MINE1;No branching priorities used;binary separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	887.6	476.53	336.31	297.86	281.6	286.27	279.63	278.26
Nodes	1011	1013	1045	1095	1101	1157	1117	1147
Sol nodes	8	10	8	11	11	13	15	13
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	122	95	126	118	120	134	127	133
Cut 2	376	400	388	417	416	428	413	421
Cut 3	0	2	1	2	3	4	4	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	0	0	2
Attack	505	506	522	547	550	578	558	573
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.67	3.34	3.73	3.95	3.94	4.03

Table 4B:69 Problem MINE2;No branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	614.67	323.29	237.99	188.89	191.3	188.23	184.83	182.68
Nodes	737	727	755	715	765	779	805	786
Sol nodes	7	5	4	5	3	6	4	5
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	134	121	137	109	106	140	135	110
Cut 2	228	238	236	243	270	242	257	271
Cut 3	0	0	1	1	2	2	4	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	368	363	377	357	384	389	405	395
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.84	2.55	3.3	3.72	3.85	4.11	4.16

Table 4B:70 Problem MINE2;No branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	738.14	433.19	336.36	205.64	209.93	191.63	186.25	192.57
Nodes	837	967	1099	779	923	851	801	845
Sol nodes	6	10	11	4	5	3	3	3
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	112	164	202	118	186	149	122	168
Cut 2	301	308	329	263	261	268	274	248
Cut 3	0	2	5	3	7	3	1	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	2	3	1	1	1
Attack	418	483	551	389	461	427	400	422
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.83	2.63	3.35	3.75	4.05	4.14	4.04

Table 4B:71 Problem MINE2;No branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	976.63	358.55	426.5	244.25	304.23	196.24	281.27	369.54
Nodes	1059	763	1273	929	1199	901	1245	1645
Sol nodes	11	7	10	5	9	4	7	14
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	73	75	166	168	121	178	190	248
Cut 2	446	298	457	290	467	267	421	553
Cut 3	0	2	2	1	3	2	4	7
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	2	1	0	0	1	1
Attack	529	381	636	464	599	450	622	822
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.82	2.56	3.25	3.77	4.02	4.1	4.13

Table 4B:72 Problem MINE2;No branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	2212.6	1169.5	884.09	702.33	698.48	694.21	716.84	716.59
Nodes	2644	2656	2712	2667	2652	2648	2761	2707
Sol nodes	8	9	8	6	5	7	7	8
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	245	228	257	244	236	220	275	248
Cut 2	1103	1137	1125	1115	1117	1129	1136	1129
Cut 3	5	2	2	7	6	4	2	6
Cut 4	6	5	0	0	0	0	0	0
Cut 5	6	0	0	1	0	1	1	1
Attack	1288	1302	1320	1298	1288	1287	1340	1316
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.84	2.54	3.25	3.43	3.53	3.52	3.5

Table 4B:73 Problem MINE2;Branching priorities used;Fixed separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1905.5	1130.4	812.13	657.24	604.78	659.43	641.81	602.31
Nodes	2382	2401	2502	2386	2256	2472	2345	2406
Sol nodes	6	3	3	4	3	5	3	5
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	287	281	285	265	256	251	280	246
Cut 2	837	887	887	857	902	1015	1033	887
Cut 3	0	0	3	1	4	2	4	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	1	0	1	0
Attack	1162	1210	1214	1157	1080	1199	1145	1165
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.81	2.54	3.06	3.4	3.52	3.54	3.56

Table 4B:74 Problem MINE2;Branching priorities used;Fixed separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	2268.5	1260.1	784.61	721.31	602.17	572.05	648.83	596.76
Nodes	2845	3008	2437	2566	2411	2361	2584	2615
Sol nodes	8	18	7	2	5	3	5	5
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	407	427	238	258	314	304	307	405
Cut 2	1058	1086	1008	1061	838	906	1015	860
Cut 3	0	4	3	1	3	4	1	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	1	1	0	2	2	1
Attack	1376	1464	1180	1243	1151	1144	1054	1261
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.82	2.51	3.16	3.3	3.67	3.66	3.69

Table 4B:75 Problem MINE2;Branching priorities used;Fixed separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	2523.8	1365.2	770.34	819.16	614.71	568.59	595.66	575.51
Nodes	2962	2964	2442	3171	2408	2391	2479	2306
Sol nodes	11	10	8	11	4	5	7	4
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	138	154	217	147	252	304	283	287
Cut 2	1367	1351	1035	1258	984	926	980	926
Cut 3	0	0	4	4	6	2	3	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	1	2	1	0	0
Attack	1448	1448	1182	1550	1160	1155	1198	1125
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.82	2.61	3.25	3.5	3.71	3.72	3.67

Table 4B:76 Problem MINE2;Branching priorities used;Fixed separation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	484.57	478.73	345.65	286.27	277.38	273.48	271.72	270.48
Nodes	1011	1023	1059	1068	1117	1139	1165	1141
Sol nodes	8	9	8	10	12	11	11	10
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	122	94	115	115	108	127	115	128
Cut 2	376	405	405	406	436	424	450	428
Cut 3	0	1	2	2	2	5	3	3
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	0	2	1	1	2	2
Attack	505	513	529	534	558	569	582	570
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.86	2.63	3.16	3.63	4.03	4.16	4.04

Table 4B:77 Problem MINE2,No branching priorities used,Beale and Forrest separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	612.48	314.34	227.56	211.84	184	183.95	180.7	181.8
Nodes	737	715	735	804	779	797	799	805
Sol nodes	7	5	5	4	4	5	5	3
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	134	110	129	131	149	118	119	131
Cut 2	228	243	235	264	232	271	269	265
Cut 3	0	0	2	1	5	2	4	1
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	0	0
Attack	368	357	369	404	389	400	402	404
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.89	2.63	3.29	3.68	4.05	4.17	4.18

Table 4B:78 Problem MINE2,No branching priorities used,Beale and Forrest separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	729.96	507.24	282.65	252.82	186.26	208.28	188.84	191.75
Nodes	838	1259	955	985	770	1001	887	851
Sol nodes	6	14	8	8	4	5	3	2
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	112	292	188	165	109	245	170	212
Cut 2	303	324	275	317	275	246	264	245
Cut 3	0	0	8	1	2	4	3	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	1	2	0	1	1	2	1
Attack	418	628	477	494	384	500	445	467
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.85	2.63	3.34	3.78	4.03	4.25	4.22

Table 4B:79 Problem MINE2,No branching priorities used,Beale and Forrest separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	973.44	519.49	393.38	355.04	267.43	260.89	339.11	314.78
Nodes	1098	1129	1183	1349	1153	1233	1483	1384
Sol nodes	11	18	10	16	8	7	11	14
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	73	115	123	145	244	268	196	211
Cut 2	446	437	455	504	320	338	528	464
Cut 3	0	1	3	7	3	2	6	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	2	1	3	2	2	0	1
Attack	529	564	591	674	576	618	740	682
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.86	2.59	3.41	3.62	3.98	4.12	4.08

Table 4B:80 Problem MINE2,No branching priorities used,Beale and Forrest separation strategy,Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1259.8	463.33	402.03	427.1	392.38	391.87	388.58	351.52
Nodes	1559	1567	1574	1656	1617	1614	1624	1485
Sol nodes	8	7	6	8	9	7	5	4
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	166	147	154	168	164	170	197	135
Cut 2	628	654	650	663	656	649	632	625
Cut 3	0	0	0	2	4	4	1	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	1	0	1	1	1
Attack	757	759	763	804	784	783	787	718
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.98	2.65	3.28	3.6	3.66	3.7	3.81

Table 48:01 Problem MINE3;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	994.43	532.45	372.45	337.62	289.46	311.98	309.17	289.13
Nodes	1314	1320	1236	1308	1214	1168	1639	1220
Sol nodes	6	4	3	4	3	5	4	3
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	243	239	181	256	169	231	336	191
Cut 2	432	447	448	458	458	460	406	450
Cut 3	0	1	2	3	3	4	2	2
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	0	0	0	0	1	0
Attack	833	631	592	687	581	648	690	584
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.86	2.57	3.24	3.61	3.67	3.69	3.71

Table 48:02 Problem MINE3;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1249.5	490.49	479.06	364.98	364.29	314.56	279.85	308.38
Nodes	1626	1755	1600	1485	1606	1345	1158	1401
Sol nodes	8	13	14	4	8	3	2	3
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	264	314	289	286	310	156	117	284
Cut 2	570	571	557	478	513	543	404	435
Cut 3	0	4	1	1	4	1	1	4
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	3	1	1	3	0	0	3
Attack	784	848	808	712	772	642	554	672
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.92	2.58	3.16	3.53	3.83	3.81	3.72

Table 48:03 Problem MINE3;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	1476.1	769.29	497.85	422.49	306.4	318.89	306.3	341.37
Nodes	1800	1790	1677	1691	1308	1421	1359	1453
Sol nodes	12	9	12	5	7	4	3	5
Inf nodes	0	0	0	0	0	0	0	0
Cut 1	152	143	193	230	189	248	181	100
Cut 2	760	768	654	636	521	487	524	637
Cut 3	0	2	3	2	0	3	4	5
Cut 4	0	0	0	0	0	0	0	0
Cut 5	0	0	1	0	2	1	0	5
Attack	876	868	814	818	624	678	647	701
Max tptr	1	2	3	4	5	6	7	8
Av tptr	1	1.88	2.67	3.27	3.74	3.78	3.85	3.91

Table 48:04 Problem MINE3;Branching priorities used;Beale and Forrest seperation strategy;Node selection strategy 4 used.

Tptrs	1	2	3	4	5	6	7	8
Time	880.89	471.32	357.73	291.44	268.47	268.7	274.46	275.17
Nodes	1011	1029	1057	1107	1117	1143	1153	1143
Sol	8	7	9	10	11	13	11	13
Ini	0	0	0	0	0	0	0	0
Cut I	122	118	122	132	118	134	146	128
Cut II	376	389	396	406	423	424	416	426
Cut III	0	1	2	4	6	1	4	5
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	0	0	2	1	0	0	0
Attack	505	514	528	553	558	571	576	571
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.86	2.59	3.39	3.88	4.05	4.02	4.01

Table 4B:85 Problem MINE2;No branching priorities used;binary separation strategy;Node selection strategy 1 used.

Tptrs	1	2	3	4	5	6	7	8
Time	602.04	345.1	228.11	200.42	174.99	181.36	170.98	182.63
Nodes	737	764	749	765	775	801	807	825
Sol	7	4	5	4	5	4	4	5
Inf	0	0	0	0	0	0	0	0
Cut I	134	104	133	109	144	116	168	141
Cut II	228	272	236	265	236	275	229	265
Cut III	0	0	1	2	3	4	3	2
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	0	0	0	0	0	0	0
Attack	368	384	374	385	387	402	403	412
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.85	2.65	3.35	3.83	4.11	4.17	4.19

Table 4B:86 Problem MINE2;No branching priorities used;binary separation strategy;Node selection strategy 2 used.

Tptrs	1	2	3	4	5	6	7	8
Time	723.42	433.09	291.98	233.38	184.99	188.23	169.28	192.96
Nodes	837	1004	911	925	747	861	789	919
Sol	6	7	7	3	2	5	3	4
Inf	0	0	0	0	0	0	0	0
Cut I	112	190	132	180	97	142	135	201
Cut II	301	302	314	276	271	277	252	253
Cut III	0	2	0	2	2	3	4	0
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	0	1	0	0	2	1	0
Attack	418	503	457	464	375	432	394	461
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.81	2.58	3.3	3.77	4.19	4.33	4.22

Table 4B:87 Problem MINE2;No branching priorities used;binary separation strategy;Node selection strategy 3 used.

Tptrs	1	2	3	4	5	6	7	8
Time	970.64	519.21	421.78	334	331.86	176.25	221.68	200.81
Nodes	1059	1085	1293	1259	1307	777	1015	891
Sol	11	12	11	14	9	6	7	4
Inf	0	0	0	0	0	0	0	0
Cut I	73	66	134	164	170	108	171	135
Cut II	446	464	495	450	467	273	326	304
Cut III	0	0	5	2	5	2	2	2
Cut IV	0	0	0	0	0	0	0	0
Cut V	0	1	2	0	3	0	2	1
Attack	529	542	646	629	653	388	507	445
Max Tptr	1	2	3	4	5	6	7	8
Av Tptr	1	1.85	2.66	3.33	3.69	4.15	4.16	4.12

Table 4B:88 Problem MINE2;No branching priorities used;binary separation strategy;Node selection strategy 4 used.