A two-stage parallel branch and bound algorithm for mixed integer programs

V. Nwana, K. Darby-Dowman and G. Mitra
Department of Mathematical Sciences
Brunel University, West London
UB8 3PH
United Kingdom

Abstract

Mixed integer programming (MIP) models are extensively used to aid both in strategic and tactical decision making in many business sectors. Solving MIP models is a computationally intensive process and there is a need to develop solution approaches that enable larger models to be solved within acceptable timeframes. In this paper, we describe the implementation of a two-stage parallel branch and bound (PB&B) algorithm for Mixed Integer Programming (MIP). In stage 1 of the algorithm, a multiple heuristic search (MHS) is implemented in which a number of alternative search trees are investigated using a forest search in the hope of finding a good solution quickly. In stage 2, the search is reorganised so that the branches of a chosen tree are investigated in parallel. A new heuristic is introduced, based on a best projection criterion, which evaluates alternative B&B trees in order to choose one for investigation in stage 2 of the algorithm. The heuristic also serves as a way of implementing a quality load balancing scheme for stage 2 of the algorithm. The results of experimental investigations are reported for a range of models taken from the MIPLIB library of benchmark problems.

1 Introduction

Mixed integer programming (MIP) models are extensively used to aid both strategic and tactical decision making in many business sectors. Successful recent applications reported in recent issues of the journal Interfaces, published by INFORMS include supply chain management in the motor industry [9], production scheduling in the brewing industry [13], aircraft and crew scheduling [5], asset liability management [20], energy management in the utilities sector [23] and network design in the telecommunication sector [2]. MIP models are classified as *NP-Hard* and problems in this class are often difficult to solve to proven optimality. Over the years, significant advances in computer technology have resulted in faster solvers capable of processing larger models. In addition, many improvements have resulted from refinements of existing algorithms [15],[22]. In spite of these advances, there is a need for even greater processing power. Parallel computers offer additional resources over serial computers, especially in terms of memory and processing speed and are an alternative platform for improved solver performance. The well established approach for solving MIPs is branch and bound (B&B) [12],[18] and several parallel implementations of B&B have been proposed [1], [4], [16],[19], [6].

In this paper, we describe extensions to the design of the two-stage parallel branch and bound (PB&B) algorithm of Mitra [19] for processing MIP. Central to the success of this implementation is a new heuristic based on a best projection criterion which evaluates alternative B&B trees in order to choose one for investigation. The heuristic also serves as a way of implementing a quality load balancing scheme for PB&B.

In section 2, we provide a mathematical definition of MIP as well as a description of a serial B&B algorithm for MIP. In section 3, we discuss the parallel rationale as well as some implementation issues. In addition, we present a new tree assessment heuristic that is used to find the "best" tree from a number of different B&B trees. In section 4, we provide experimental results for the entire two-stage algorithm before proceeding, in section 5, to draw conclusions from the results.

2 Problem definition and the serial B&B algorithm

The Mixed Integer Program (MIP) can be stated as follows:

$$Min \qquad Z_{MIP} = \sum_{j \in B} c_j x_j + \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \tag{1}$$

subject to

$$\sum_{j \in B} a_{ij} x_j + \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m \tag{2}$$

$$\begin{array}{ll} l_j \leq x_j \leq u_j & j \in N \\ x_j \in \{0,1\} & j \in B \\ x_j \geq 0 \text{ and integer} & j \in I \\ x_j \geq 0 & j \in C, \end{array}$$

where B is the index set of binary variables, I the index set of general integer variables, C the index set of continuous variables and $N = B \cup I \cup C$. l_j and u_j are the lower and upper bounds on the variables, x_j.

The B&B algorithm adopts a tree search in which the tree development process is characterised by two rules (operations) that perform branching and bounding of the solution space. The root node, P_0, of the tree represents the entire state space $S = S_0$ while subsequent nodes (sub-problems) P_j represent successively smaller partitions S_j of S. The set of all feasible solutions is represented by the set of feasible solutions of the sub-problems associated with the uninvestigated or dangling nodes of the tree. At each node of the tree, the linear programming relaxation (LPR) of the IP, in which the integrality constraints are dropped, is solved. If there is no feasible solution to the LPR at a node, then the node is terminated. Otherwise, if the solution to the LPR of a subproblem is integer feasible and its objective function value is less than the previous upper bound, then the objective function value of this subproblem is set as an upper bound for the problem. The subproblem or node for which the upper bound is obtained is called the incumbent node and the bound is known as the incumbent value. In addition, the optimal solution to the LPR provides a lower bound on the best integer feasible solution for

that subproblem. After each branching process, those subproblems with an objective function lower bound that exceeds the incumbent value are excluded from further branching. The branching continues until the best integer feasible solution is found and its optimality is proven by examining all the eligible nodes in the search tree.

Notable enhancements to B&B include the incorporation of constraint classification, preprocessing, integer heuristics and cutting planes to the procedure [1],[14]. Constraint classification involves carrying out analyses on the constraint set with a view to capturing well known constraint classes that can be exploited by the B&B solution process. Preprocessing is applied prior to, and during the execution of B&B by making inferences that may lead to reduction or simplification of the search space. B&B also has the flexibility to accommodate problem-specific and general heuristics that exploit properties of the search space to generate integer feasible solutions and hence upper bounds. Bixby *et al.* [1] refers to these heuristics as primal heuristics. Cutting planes or valid inequalities that reduce the search space can be applied to the original MIP model as well as subproblems within the B&B tree. Details of these enhancements and their effects on the performance of B&B can be found in Kularajan [14] and Linderoth and Savelsbergh [18].

3 Two-stage B&B algorithm of Mitra *et al.*

3.1 Introduction and parallel rationale

The tree search of serial B&B has two distinct goals, namely

(i) to find a good and ultimately the best integer feasible solution, and

(ii) to prove optimality or infeasibility.

Hai [10] established that these two goals could be addressed in a two-stage parallel B&B algorithm. We refer to the two stages of the PB&B algorithm as *stage 1* and *stage 2*.

Serial B&B algorithms possess a high degree of non-determinism in their performance [19] such that different implementations of the same algorithm may result in vastly different trees and execution times, depending on the tree search heuristics (primarily node and variable choices). A study of the variability in performance of several different implementations of B&B is

discussed in Nwana [21]. In general, it is usually impossible to determine *a priori* the best B&B tree search strategy. In practice, dynamically estimated quantities such as *psuedocosts* for variable selection, and *best projection* for node choice are usually calculated in order to determine a search strategy for B&B.

Gendron and Crainic [8] identify three design approaches to PB&B algorithms. *Parallelism of Type 1* is akin to low level parallelism in which the innermost operations (such as node and variable choices) within B&B are parallelised whilst maintaining the overall framework of the serial algorithm. *Parallelism of type 2* employs parallelism at a higher level such that independent nodes within the B&B tree are investigated simultaneously. *Parallelism of type 3* is achieved when several distinct B&B trees are explored simultaneously.

In our two stage algorithm described below, we employ parallelism of type 3 in stage 1, and parallelism of type 2 in stage 2.

In serial B&B, once one or more integer feasible solutions are found, the algorithm proceeds to prove optimality by eliminating nodes which, by virtue of their bounds, cannot yield improved integer feasible solutions. Again, it is impossible to predict the amount of work required in order to investigate and potentially eliminate nodes of a serial B&B tree from being processed. Stage 2 of our PB&B implementation is akin to the more classical PB&B, whereby the nodes of the B&B tree are distributed between different processors in order to speed up the investigation of the unexplored nodes. This stage employs *high level parallelism* or parallelism of type 2 [8].

3.2 Parallel Branch and Bound – Stage 1

By exploiting *parallelism of type 3*, we can investigate a number of alternative trees, each following a unique search strategy. This approach is referred to as *Multiple Heuristic Search* (MHS) – effectively a *forest search*. By running several tree search strategies in parallel, we gain the advantages of the individual tree searches within a single parallel heuristic. In effect, by employing MHS, we increase the probability of finding better integer solutions early-on as well as finding the 'best' tree search strategy.

This stage of the PB&B algorithm involves the distribution of the nodes of the tree throughout a network of processors (distributed MIMD architecture). The underlying parallel software connecting the processors is PVM – software that permits a network of heterogeneous computers to be used as a single

large parallel computer. Each processor starts a unique sequential B&B search and improved bounds are shared between the processors in order to help them prune their search trees - thereby improving their sequential search.

A key advantage of carrying out MHS is the investigation into the difficulty of the underlying problem. If processing several alternative trees does not yield an integer feasible solution until a considerable number of nodes have been explored then the problem may be judged to be 'difficult'. Thus, we attempt to evaluate the trees in order to select the one that is assessed to be the best in terms of its ability to yield integer feasible solutions. The tree assessment heuristic, which aims to find the 'best tree' for investigation in stage 2, is discussed in section 3.3.

MHS has a key advantage in that it can fully exploit a *massively parallel computer*. Several permutations of node and variable choices in the B&B algorithm may result in a large number of trees to be investigated. Using a parallel architecture, these trees can be searched using an equally large number of processors. The more trees that are investigated, the greater the advantages of the MHS. Communication overheads are usually not a limiting factor since only a minimal amount of information (bounds) is shared. On the other hand, investigating a large number of trees may increase further the likelihood of obtaining good integer feasible solutions.

By using the MHS approach in stage 1, we also mimimise the set-up time that is usually incurred in PB&B algorithms. PB&B implementations usually proceed by starting B&B in serial mode, generating a minimum number of nodes, and finally kick-starting the parallel execution. Hai [10] refers to this elapsed time as the *rise time* of the PB&B algorithm. In our two stage implementation, the use of MHS, in which all processors investigate their trees independently, mimimises the parallel set-up or rise time.

The obvious drawback of stage 1 is duplication of work. In investigating different B&B trees, with minimum communication between the processors, we are likely at some stage to have different processors investigating a common portion of the search space. This is especially true if the MHS is allowed to run for long periods of time. However, during the initial search stages, duplication is minimised since there is a large number of nodes and different strategies are likely to choose different trees. MHS is therefore a good initialisation strategy for PB&B. However, it becomes wasteful [10] when pursued for too long. Usually when B&B or PB&B algorithms are executed, the user specifies the maximum time limit for the execution run. In our algorithm, we heuristically set (based on emperical evidence) the amount of time for stage

1 to be one third of the preset maximum time limit. Alternative criteria may also be used to terminate stage 1 such as:

(i) the threshold value for the number of waiting nodes on a processor is attained,

(ii) the maximum number of integer feasible solutions has been reached,

(iii) a good integer solution has been found; that is, one which has a value within a specified distance from the current lower bound.

Within the serial B&B code, a tree assessment heuristic is implemented such that each tree that is being investigated in stage 1 is evaluated in terms of its estimated potential for yielding integer feasible solutions at an early stage. We choose the 'best' tree for investigation in stage 2 based on this evaluation. Three alternative B&B tree assessment heuristics are presented below and, based on some empirical results, one is selected as our chosen tree assessment heuristic.

3.3 Assessing Alternative Branch and Bound Trees in Stage 1

The three tree assessment criteria considered in this investigation are based on three basic observations on the computational behaviour of the B&B algorithm:

(i) B&B trees which follow a depth-first search tend to find integer feasible solutions early-on in the search, since most integer solutions are found deep-down in the tree [17], [18]. Whilst the quality of the integer solutions cannot be assessed in advance, finding such solutions guides the search by providing upper bounds. Our first criterion therefore is called *Relative Depth* (RDPTH) and is given by

$$RDPTH = \frac{IDPTH}{BININT},$$

where $IDPTH$ represents the depth of the deepest node in the serial B&B tree, and $BININT$ represents the number of binary and general integer variables in the model . Since the rationale for this measure is finding good solutions, the tree with the highest value of RDPTH is chosen.

(ii) The trees resulting from breadth-first searches closely resemble a complete enumeration – especially in the initial stages of the search. Such trees therefore tend to take a longer time to reach integer solutions. Our measure of breadth of a tree is *Relative Breadth* (RBDTH) and is given by

$$RBDTH = \frac{UNODES}{IDPTH},$$

where $UNODES$ represents the total number of uninvestigated nodes in the B&B tree. Since the goal of finding integer solutions early in the search is of overriding importance, we choose the tree with the lowest value of *RBDTH*.

(iii) Best projection is often used as an intelligent node selection strategy which attempts to find an estimate of the best integer feasible solution attainable for a given node in a B&B tree. In best projection, an estimate E_i is calculated for node P_i in the following way. Let $s_i = \sum_{j \in B \cup I} min\{x_j^i - \lfloor x_j^i \rfloor, \lceil x_j^i \rceil - x_j^i\}$ where B and I are the index sets of binary and general integer variables respectively and x_j^i represents a binary or general integer variable whose current value is fractional in the optimal LP solution vector $\mathbf{x}^*{}_{LPR(i)}$. s_i therefore represents the sum of the integer infeasibilities at node P_i. Trivially, the optimal LP solution for the root node, P_0, is represented by $\mathbf{x}^*{}_{LPR_{(0)}}$ and has an objective function value, $\underline{z}^*_{LPR(0)}$. The best projection estimate, E_i for node P_i is given by,

$$E_i = \underline{z}^*_{LPR(i)} + \left(\frac{\overline{z}^*_{IP} - \underline{z}^*_{LPR(0)}}{s_0} \right) s_i. \tag{3}$$

The term $\frac{\overline{z}^*_{IP} - \underline{z}^*_{LPR(0)}}{s_0}$ is an estimate of the change in the objective function value per unit decrease in infeasibility. E_i is an estimate of the best integer solution attainable for node P_i. The best projection rule selects the node associated with the minimum value of E_i. By definition, best projection estimates can only be calculated if B&B has obtained an integer feasible solution. The sum of E_i over all uninvestigated or open nodes may be viewed as an estimate of the "goodness" of the B&B search tree. The summative best projection value $BPROJ_{(T_j)}$ for a

given B&B tree T_j is given by,

$$BPROJ_{(T_j)} = \sum_{over\ all\ open\ nodes} \left\{ \underline{z}^*_{LPR_{(i)}} + \left(\frac{\overline{z}^*_{IP} - \underline{z}^*_{LPR(0)}}{s_0} \right) s_i \right\}. \quad (4)$$

Given t B&B trees, the final tree assessment criterion that we consider is $BPROJ$, whose value is obtained by the following expression;

$$BPROJ = \underset{j}{Min}\{BPROJ_{(T_j)}\}$$

When many alternative B&B trees are being evaluated, we choose the tree whose summative best projection over all nodes ($BPROJ$) is a mimimum. The drawback of this criterion is the fact that $BPROJ$ depends on the existence of an existing integer feasible solution. If no such solution exists, then we use one of the other two criteria discussed above.

3.3.1 Results of Tree Assessment Criteria

In order to investigate the usefulness of the three tree assessment criteria described above, a computational experiment was carried out using five models from MIPLIB [3]. The model characteristics are described in Table 1. Seven different B&B trees were generated for each of the models. The trees were generated by considering various tree search heuristics, defined by the combination of node and variable choice techniques. Typical examples of node choice techniques used are: last in first out, first in first out, minimum fractionality and best projection. Variable choice criteria include: minimum or maximum fractionality of variables and a minimum or maximum cost. The solver used in this investigation is FortMP [7] and details of its variable and nodes choices may be found in the FortMP user manual [7]. Seven of the best tree search heuristics were chosen based on the results of previous studies from Hajian [11] and Kularajan [14]. In this preliminary investigation, a set of five models taken from the MIPLIB library [3] were considered. Summary statistics for these five models are shown in Table 1 below.

The execution time limit for processing each of the seven trees generated per model was set at one hour. For each tree, numerical values for the three tree assessment criteria ($RDPTH$, $RBDTH$ and $BPROJ$) were recorded at the end of each run. We also recorded statistics for the number of integer

Model Name	No. of constraints	No. of binary variables	No. of continuous variables	Total no. of variables	No. of non zeros
DCMULTI	290	75	473	548	1833
NW04	36	87482	0	87482	724148
LSEU	28	89	0	89	394
DANOINT	664	56	465	521	3233
CAP6000	2176	6000	0	6000	54238

Table 1: Model statistics for five MIPLIB models

solutions ($NINT$), the number of processed nodes ($NODES$), the iteration count ($ITER$), the best integer solution ($IPBST$), the time taken to find the first integer solution ($TT1$) and the time taken to find the best integer solution (TTB), measured in seconds. $LP(OPT)$ and $IP(OPT)$ represent the LP and IP optimum solution values respectively. Except where stated, all problems were solved to proven optimality.

Table 2: Assessing alternative trees for model DCMULTI

Model: DCMULTI $LP(OPT) = 183975.5397$ $IP(OPT) = 188182$

Tree	RDPTH	RBDTH	BPROJ	NINT	NODES	ITER	IPBST	TT1	TTB
1	0.3567	4533.98	13200	1	15307	133045	189265	0.18	0.18
2	0.4000	0.5000	23.09	41	30337	86504	193276	0.28	120.03
3	0.1733	583.10	8756	16	15477	114528	196008	0.27	120.20
4	0.4267	0.0312	0.0304	16	10750	52794	188182*	0.24	119.34
5	0.3600	0.4815	21.09	39	30487	86901	193267	0.29	120.04
6	0.4000	0.4667	21.45	41	30444	86789	193276	0.29	120.03
7	0.4267	0.0200	0.0206	16	10750	52794	188182*	0.29	88.23

*: Optimal solution obtained but optimality not proved

Table 3: Assessing alternative trees for model NW04

Model: NW04 $LP(OPT) = 16310.70$ $IP(OPT) = 16862$									
Tree	**RDPTH**	**RBDTH**	**BPROJ**	**NINT**	**NODES**	**ITER**	**IPBST**	**TT1**	**TTB**
1	0.854	0.3230	4.138	2	35	1316	16922	245.54	720.18
2	0.754	0.4000	4.424	1	29	1316	16956	342.42	342.42
3	0.899	0.1733	6.322	1	35	1360	16956	445.27	445.27
4	0.712	0.4267	4.138	2	35	1316	16922	67 .78	459.48
5	0.812	0.3600	3.667	2	31	1387	16922	40.57	245.54
6	0.751	0.4000	3.615	2	31	1387	16922	57.61	340.63
7	0.719	0.4267	6.322	1	35	1360	16956	234.29	234.29

Table 4: Assessing alternative trees for model LSEU

Model: LSEU $LP(OPT) = 834.68$ $IP(OPT) = 1120$									
Tree	**RDPTH**	**RBDTH**	**BPROJ**	**NINT**	**NODES**	**ITER**	**IPBST**	**TT1**	**TTB**
1	0.545	60	1.638	16	81326	3357	1136	3.87	76.54
2	0.343	688	9200	7	55335	6523	1120	6.97	142.64
3	0.134	1140	9584	9	42607	4935	1128	4.65	341.47
4	0.000	0.00	2.865	16	96101	3563	1136	2.94	245.48
5	0.442	3.0	68.75	16	87269	4976	1120	16.51	145.88
6	0.471	2.0	78.22	14	87212	4312	1120	10.60	252.35
7	0.215	1128	9490	9	63653	5692	1128	23.99	133.25

Table 5: Assessing alternative trees for model DANOINT

Model: DANOINT $LP(OPT) = 62.637280418$ $IP(OPT) = 65.67$									
Tree	**RDPTH**	**RBDTH**	**BPROJ**	**NINT**	**NODES**	**ITER**	**IPBST**	**TT1**	**TTB**
1	0.2857	0.6875	3.154	3	6032	292469	66.2727	245.54	720.18
2	0.1607	227.90	1386	6	3330	229442	66.4545	342.42	342.42
3	0.2679	153.20	989.6	5	2894	225100	67.5000	445.27	445.27
4	0.2864	0.6250	2.318	3	5064	244311	66.2727	67 .78	459.48
5	0.2687	2.000	8.611	3	5506	244922	66.2727	40.57	245.54
6	0.2143	187.50	970.10	5	2834	220908	67.5000	57.61	340.63
7	0.3019	152.33	764	2	4353	221133	67.5000	234.29	234.29

Table 6: Assessing alternative trees for model CAP6000

Model: CAP6000 $LP(OPT) = -245154$ $IP(OPT) = -2451377$									
Tree	**RDPTH**	**RBDTH**	**BPROJ**	**NINT**	**NODES**	**ITER**	**IPBST**	**TT1**	**TTB**
1	0.5404	0.144	0.00	0	6014	8083	N/A	N/A	N/A
2	0.2625	0.2838	25.65	4	12244	15787	-2446110.2	361.13	486.48
3	0.0933	0.8929	1.839	14	35707	46907	-2450350	361.95	545.27
4	0.0173	0.9615	5.779	20	65245	92942	-2451129.8	365.09	600.76
5	0.0025	0.400	0.9552	26	122075	181540	-2451390	243.72	345.54
6	0.2254	0.3220	3.432	13	135334	150674	-2451129.8	421.48	740.63
7	0.4213	0.243	4.344	7	85439	107371	-2451129.8	234.29	764.29

Tables 2–6 show these statistics for each of the five models presented in Table 1. As expected, the larger the observed values of $RDPTH$ the smaller the values of $RBDTH$ tend to be. Thus, the use of either criterion would tend to lead to similar results.

The observed values of $BPROJ$ show interesting results. Smaller values of $BPROJ$ coincide with the "best trees" within the preset solution time. The assessment of best tree is based on the quality of the final integer solution, the number of nodes investigated, the number of integer solutions found and the number of iterations performed. The tree that is to be investigated in stage 2 of our PB&B implementation is the one with the minimum value for $BPROJ$. In cases where no integer solution is obtained during stage 1, that is $BPROJ = 0$ for the given time period, we choose the tree with the largest value of $RDPTH$.

3.4 Parallel Branch and Bound – Stage 2

In Stage 2 of the algorithm, we examine in parallel the tree chosen at the end of stage 1 for further investigation. This examination may result in an improved integer solution or prove that the incumbent solution is optimal.

The rationale for applying parallelism in stage 2 is to speed up the node investigation process. Parallelism of type 2 is employed whereby unexplored nodes of the chosen tree are farmed out to other processors. The order in which the nodes are investigated is not critical to the process. Consequently, we have the flexibility to use a different search heuristic from the one that generated the chosen tree. However, preliminary experiments showed that it was better to carry on with the same search heuristic for the chosen tree as in stage 1.

The nodes of the tree are distributed amongst all the processors. We ensure that all the processors have roughly the same number of B&B nodes at the start of Stage 2. This is achieved by distributing the waiting nodes, one at a time to successive processors. If, in the course of stage 2, a processor completes its search, more nodes are assigned to it from a loaded processor. However, in the course of stage 2, the tree assessment criteria are also computed. In effecting load balancing from a loaded to an idle slave we choose nodes from a loaded processor with the current 'best' tree defined by the tree assessment criteria presented earlier. This is an effective way of achieving quality load balancing.

The algorithm runs in stage 2 until one of the following termination criteria is encountered:

(i) the list of waiting nodes is empty,

(ii) the preset limit of total number of nodes opened has been reached or,

(iii) the preset limit on elapsed time is reached.

These criteria are the same as those for terminating serial B&B. When any of the termination criteria is encountered, the best integer solution found so far is retrieved and the solution is reported. If Stage 2 terminates as a result of condition (1) above, then the optimality of the current integer solution (if any) is guaranteed whereas, for conditions (2) and (3), optimality is not proven.

4 Experimental results of two-stage PB&B

4.1 Developing a Testing Strategy

We implemented the two-stage PB&B harness on a cluster of up to four Pentium III 500MHz machines with 128MB of RAM. The four machines were connected by a 100Mbit LAN (Ethernet) and the underlying communication (message passing) system is the PVM library. The serial B&B algorithm used as control is run using one of these processors. The main metric for testing the algorithm is *speed-up* defined by

$$\mu = \frac{t(1)}{t(n)}$$

– the ratio of the times taken to execute B&B on one and n processors, respectively.

The experiments are conducted on a set of MIPLIB [3] models for which summary statistics are presented in Table 7 below.

Model name	No. of constraints	No of variables	Total no. of general integer variables	No. of binary variables	No. of continuous variables	No. of non zeros
10TEAMS	230	2025	1800	1800	225	14175
AIR05	426	7195	7195	7195	0	59316
BELL3A	123	133	71	39	62	441
BLEND2	274	353	264	231	89	1497
CAP6000	2176	6000	6000	6000	0	54238
DANO3MIP	3202	13873	552	552	13321	79656
DANOINT	664	521	56	56	465	3233
DCMULTI	290	548	75	75	473	1833
DSBMIP	1182	1886	192	160	1694	9768
FIBER	363	1298	1254	1254	44	4298
FIXNET6	478	878	378	378	500	2550
GESA2	1392	1224	408	240	816	6000
GESA3	1368	1152	384	216	768	5736
L152LAV	97	1989	1989	1989	0	11911
LSEU	28	89	89	89	0	394
MARKSHARE1	6	62	50	50	12	324
MISC07	212	260	259	259	1	8620
MOD010	146	2655	2655	2655	0	13858
NW04	36	87482	87428	87482	0	724148
P0282	241	282	282	282	0	2248
P0548	176	548	548	548	0	2127
P2756	755	2756	2756	2756	0	11103
QIU	1192	840	48	48	792	3744
QNET1	503	1541	1417	1288	124	4746
ROUT	291	556	315	300	241	2432
SWATH	884	6805	6724	6724	81	34966
VPM1	234	378	168	168	210	917

Table 7: Model statistics for twenty seven MIPLIB models

The different trees investigated in stage 1 of the algorithm are based on the harness for constructing alternative B&B trees, developed by Hajian [11] and implemented in the solver FortMP [7]. The three parameters which, when assigned different values, generate the alternative trees are VARCHO, FNODCH and SNODCH. VARCHO determines the variable choice strategy; FNODCH, the node choice strategy until the first integer solution is found and SNODCH, the subsequent node choice strategy. In addition, preprocessing and cuts were used where appropriate, following the investigations by Kularajan [14].

Any observed speed-up in the PB&B algorithm may be attributed to one or a combination of the following three points:

(i) the attainment of different good feasible solutions in stage 1 which impacts on the tree development of all strategies (see section 3.2),

(ii) the choice of the 'best' tree for investigation in stage 2 (see section 3.3), and

(iii) the rate of investigation of B&B nodes in the chosen tree (see section 3.4).

Points (i) and (ii) are implicitly investigated and presented in section 3.4. The tree best projection criterion, $BPROJ$, is adopted for choosing the best tree to be investigated in stage 2. In the serial B&B executions reported in section 3.3, the times taken to find the first integer solution vary from tree to tree. However, in the parallel execution, the first integer solution to be found over all trees is communicated to every other tree. Thus, ignoring the time taken to communicate this information, all trees obtain a first integer solution in the same time. In addition, the subsequent processing of a tree is affected by the receipt of the initial bound with an expected beneficial effect.

The effectiveness of parallelism in stage 2 is investigated by comparing the times taken by serial and parallel executions to investigate a given number of pre-generated nodes farmed out to the processors in the same sequence for each execution. Whilst this comparison is not strictly accurate (because of anomalies due to inter-processor communication), it provides a measure of *speed-up in proving optimality*. At the end of stage 1 of PB&B, we implemented special cases of the code which distributes the nodes to be investigated between two, three and four processors respectively. In the control experiment, all the nodes were processed by a single processor. This experimental procedure was carried out for all twenty seven MIPLIB models

presented in Table 7. In Table 8, we present the results of the experiments by illustrating the speed-up in proving optimality for each model, given by, $\mu = \frac{t(1)}{t(n)}$, where n is the number of processors and $t(n)$ the time taken to investigate all the nodes using n processors. The results show that, in most

	Number of Processors		
	2	3	4
Problem Name			
10TEAMS	1.67	2.74	3.97
AIR05	1.82	2.78	3.10
BELL3A	2.02	3.15	4.21
BLEND2	1.23	1.01	0.73
CAP6000	1.36	1.57	1.88
DANO3MIP	1.79	2.84	3.98
DANOINT	2.00	3.18	4.12
DCMULTI	0.93	0.71	0.70
DSBMIP	1.34	2.15	2.98
FIBER	2.00	3.19	4.24
FIXNET6	1.99	2.85	3.87
GESA2	1.27	2.35	3.65
GESA3	2.31	3.14	4.00
L152LAV	1.93	4.18	4.19
LSEU	0.98	0.77	0.61
MARKSHARE1	1.73	2.88	4.01
MISC07	1.47	2.87	3.60
MOD010	1.21	3.00	3.88
NW04	1.44	2.53	3.72
P0282	1.33	2.09	3.54
P0548	1.68	3.00	4.35
P2756	1.35	2.66	3.73
QIU	1.87	3.11	4.23
QNET1	1.78	2.57	3.70
ROUT	1.81	2.94	4.00
SWATH	1.71	2.98	3.33
VPM1	2.00	2.81	3.92

Table 8: Analysis of the speed-up (μ) in investigating a fixed number of nodes on 2, 3 and 4 processors respectively.

cases, there is at least a near-linear speed-up in the time to prove optimality or alternatively, in the time to obtain the best integer solution to the models. However, some models do not benefit from parallelisation (BLEND2,

DCMULTI, LSEU). These models were easily solved in serial mode and the overheads associated with a parallel execution dominated the small savings gained by having additional processors. However, the results confirm the overall benefits of parallelism in stage 2 of our PB&B algorithm.

4.2 Overall Results of Two-Stage B&B Algorithm

The results of the executions of our two stage PB&B algorithm presented in this section are further divided into three cases. First, we investigate the effect of our PB&B implementation on small problems which are easily solved to optimality by our serial solver. These problems are classed as *ESS*('Easy – Solved in Serial'). Secondly, we test the PB&B code on problems for which serial B&B required a large (prohibitive) amount of time to solve to optimality. We refer to this class as *DSS* ('Difficult – Solved in Serial'). Finally, we test PB&B on larger problems that serial B&B could not solve to optimality. We class these models as *DNS* (Difficult – Not Solved in Serial'). The results are presented in Table 9. $TT1$ represents the time taken to the first solution, TTB is the time taken to obtain the best integer solution and TT is the total time taken to process the problem. $IPBST$ refers to the best integer solution and a '†' appended to an entry in this column indicates that optimality is proved by the algorithm. If optimality could not be proved, the optimal value obtained from MIPLIB [3] is shown in brackets. Finally, $Speed-up$ represents the speed-up of the PB&B code. The '*' symbol is entered in the '$Speed-up$' column in cases where the PB&B code solves a DNS class model to optimality. Finally, a '‡' is appended to the model name entry in the '$Model$' column to indicate that the PB&B algorithm terminated due to reaching a the specified limit on the number of nodes that can be investigated or that the maximum time limit was reached. In cases where the executions reached their time or node limit before obtaining the optimal solution, we take $t(n)$ to be the time taken by n processors to obtain the best integer solution found to date. Since, in all cases reported here, the best integer solution of PB&B is at least as good as that obtained by the serial execution, the computational speed-up measure never over-estimates the effectiveness of PB&B.

Model	Class	TT1	TTB	TT	IPBST	Speed-up
10TEAMS	DNS	142.69	634.08	2761.23	924†	3.91
AIR05	DSS	5.12	146.98	994.35	26374†	4.29
BELL3A‡	DNS	223.69	542.11	687.02	1449323.1 (878430.32)	3.08
BLEND2	ESS	7.14	12.67	17.13	7.598985†	1.02
CAP6000	ESS	23.60	51.41	67.02	-2451377†	1.32
DANO3MIP	DNS	762.70	4313.11	6721.98	728.111†	*
DANOINT	DNS	651.20	724.99	6600.50	65.67†	*
DCMULTI	ESS	1.12	13.47	43.45	188182†	0.98
DSBMIP	ESS	2.39	62.71	65.32	-305.20†	3.58
FIBER	DNS	634.10	2434.76	5989.90	405935.18†	*
FIXNET6‡	DNS	1087.4	1087.4	2142.67	5444.00 (3983)	4.01
GESA2	DNS	13.74	67.23	4926.63	25779856.37†	*
GESA3‡	DNS	0.75	687.34	2549.75	28230469 (27991042.65)	3.92
L152LAV	DSS	133.03	556.37	625.08	4722†	2.69
LSEU	ESS	4.47	51.76	61.90	1120†	1.08
MARKSHARE‡	DNS	11.70	98.23	136.63	17.0 (1.0)	2.61
MISC07	DSS	13.74	65.89	319.55	2810†	*
MOD010	DNS	653.61	2683.25	6016.35	6548†	*
NW04	ESS	21.57	27.46	141.63	16862†	2.14
P0282	DSS	2.75	27.23	46.63	258411†	1.00
P0548‡	DNS	7.12	342.78	1211.00	9886 (8691)	3.38
P2756‡	DNS	34.78	232.35	1354.00	3721(3124)	3.12
QIU	DSS	5.87	20.45	200.10	-132.87†	5.12
QNET1	DNS	1.43	71.43	5321	16029.69†	*
ROUT‡	DNS	78.34	623.95	3543.32	1129.55(1077.56)	2.95
SWATH‡	DNS	1084.31	1084.21	7200	560.948(497.60)	3.82
VPM1	DNS	12.32	45.87	354.89	20†	*

†: Proven optimal value obtained

*: Solved to proven optimality by the parallel code but not by the serial code

‡: PB&B code terminated due to reaching a preset node limit or time limit

Table 9: Results of two-stage PB&B algorithm executions on a cluster of 4 Pentium III 500MHz processors

4.3 Summary of Results

The results of the PB&B algorithm execution show that the proposed two-stage algorithm is effective, especially for models that were difficult to solve in serial mode (DNS models). In the ESS class of models, limited speed-ups, if any, are achieved. This is the case with all the *ESS* models reported here: $BLEND2$, $CAP6000$, $DCMULTI$, $LSEU$, $NW04$ and $P0282$. The use of several different tree search strategies in stage 1 resulted in all six models solving to proven optimality in stage 1.

By exploiting a parallel architecture to investigate different tree strategies, that is performing a forest search, we are guaranteed that the results obtained by employing stage 1 of our PB&B algorithm are at least as good as the results from the best tree. The ability to process more than one tree simultaneously will result in a solution being obtained in either the same time as would be the case if only one tree was processed, or in a quicker time.

At the other extreme, one or more processors may pursue a good tree search strategy which may not have been considered during a serial execution run of B&B. If such a scenario is encountered in stage 1 of the PB&B algorithm, then we are likely to experience superlinear speed-ups which is a desirable speed-up anomaly. This was the case with the models $AIR05$ and QIU.

Of the twenty seven problems investigated, nineteen runs terminated as a result of proven optimality and eight runs terminated on reaching the limit on the number of nodes investigated or the preset execution time limit.

In line with the theory of parallel algorithms, most of the models investigated with our two-stage PB&B algorithm show at least a near-linear speed-up or improved integer solution. In addition to these, some scale-up advantages were recorded as eight of the models that could not be solved using the serial algorithm were solved to optimality using the parallel algorithm (*DANO3MIP, DANOINT, FIBER, GESA2, MISC07, MOD010, QNET1 and VPM1*).

5 Summary and conclusions

5.1 Summary

In this paper, we tested a two-stage PB&B algorithm for MIP based on the initial implementation of Mitra *et al.* [19]. The parallel algorithm exploits parallelism and B&B solution information in the following ways:

(i) stage 1 of the two-stage algorithm exploits the fact that different B&B tree implementations can yield different results. By using a parallel enviroment to investigate a number of different B&B trees concurrently, we stand a better chance of obtaining a better solution to the problem than if we had one dedicated tree search strategy.

(ii) we have developed a B&B tree assessment heuristic which gives insight into the likelihood of a B&B tree obtaining good integer feasible solutions. By using such a heuristic in stage 1 of our algorithm, we ensure that we are investigating the "best" tree while attempting to prove optimality in stage 2 of the algorithm.

(iii) in the same vein as (ii) above, the tree assessment heuristic, when used in stage 2, provides a way of implementing quality load balancing, whereby the best regions of a tree are transferred from a loaded slave to an idle slave. This further improves the efficiency of the overall algorithm

The computational experiments reported here demonstrate the effectiveness of the two-stage PBB algorithm. The effect is most pronounced when applied to problems that were difficult to solve in serial mode.

The two-stage procedure described in this paper is one of several possible hybrid approaches. A potentially productive future research area is to explore the use of different search strategies such as Simulated Annealing and Tabu Search in stage 1 of the algorithm.

References

[1] R. Bixby, W. Cook, A. Cox, and Lee E. Parallel mixed integer programming. Technical Report CRPC-TR95554, Rice University, Center for Parallel Computation, 1995.

[2] R. E. Bixby. Solving real-world linear programs: a decade and more of progress. *Operations Research*, 50:3–15, 2000.

[3] R. E. Bixby, S. Ceria, C. M. McZeal, and M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[4] Q. Chen and M. Ferris. A fault tolerant condor-pvm mixed integer program solver. Technical report, University of Wisconsin-Madison, Department of Computer Sciences, Madison, WI 53706, 1999.

[5] J. Desrosiers, A. Lasry, D. McInnis, M. M. Solomon, and F Soumis. Air transat uses ALTITUDE to magage its aircraft routing, crew pairing and work assignment. *Interfaces*, 30(2):41–53, 2000.

[6] J. Eckstein, C.A. Phillips, and W. Hart. PICO: An object-oriented framework for parallel branch and bound. Technical Report RRR 40-2000, Rutgers University, New Jersey, August 2000.

[7] E.F.D. Ellison, N. Hajian, I. Maros, G. Mitra, and D. Sayers. *A Fortran Based Mathematical Programming System: FORTMP User Manual Release 2, version 2.04*. Brunel University and NAG Ltd, 1997.

[8] B. Gendron and T. G. Crainic. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[9] C. K. Hahn, Duplaga E. A., and Hartley J. L. Supply-chain synchronization: lessons from hyundai motor company. *Interfaces*, 30(4):32–45, 2000.

[10] I. Hai. Integer programming on parallel computers. Master's thesis, Brunel University, 1994.

[11] M. T. Hajian. *Computational Methods for Discrete Programming Models*. PhD thesis, Brunel, The University of West London, 1992.

[12] E.L. Johnson, G.L. Nemhauser, and M.W.P. Savelsbergh. Progress in linear programming-based algorithms for integer programming. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[13] E. Katok and D. Ott. Using mixed-integer programming to reduce label changes in the coors aluminium can plant. *Interfaces*, 30(2):1–12, 2000.

[14] K Kularajan. *Analysis of Integer Programming Problems and Development of Solution Algorithms*. PhD thesis, Brunel University, Department of Mathematical Sciences, Uxbridge, Middlesex, UB8 3PH, March 2000.

[15] K. Kularajan, G. Mitra, F. Ellison, and B. Nygreen. Constraint classification, preprocessing and a branch and relax approach to solving mixed integer programming models. *International Journal of Mathematical Algorithms*, 2:1–45, 2000.

[16] R. S. Laundy. Implementation of branch and bound algorithms in XPRESS-MP. In T. Ciriani, S Gliozzi, E.L. Johnson, and R. Tadei, editors, *Operational Research in Industry*. Macmillan Press Ltd, 1999.

[17] J.T. Linderoth. *Topics in Parallel Integer Optimization*. PhD thesis, Georgia Institute of Technology, 1998.

[18] J.T. Linderoth and M.W.P Savelsbergh. A computational study of search strategies in mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[19] G. Mitra, I Hai, and M.T. Hajian. A distributed processing algorithm for solving integer programs using a cluster of workstations. *Parallel Computing*, 23:733–753, 1997.

[20] J. M. Mulvey, Gould G., and Morgan C. An asset and liability management system for Towers Perrin–Tillinghast. *Interfaces*, 30(1):96–114, 2000.

[21] V. L. Nwana. *Parallel Algorithms for Solving Mixed Integer Linear Programs*. PhD thesis, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom, May 2001.

[22] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming models. *ORSA Journal on Computing*, 6:445–454, 1994.

[23] R. F. Shortle, D. Dietz, P. Katz, C. Williamson, J. Koehler, and A. Elcan. Optimal design of a data-ofload network. *Interfaces*, 31(5):4–12, 2001.