Preprint. Reference for final version:
Computers and Operations Research, vol. 37, no. 9, pp. 1544-1556

Faster MIP Solutions via New Node Selection Rules

Daniel T. Wojtaszek dtwojtas@sce.carleton.ca
John W. Chinneck chinneck@sce.carleton.ca

Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6
Canada

November 12, 2009

Abstract

When a branch and bound method is used to solve a linear mixed integer program (MIP), the order in which the nodes of the branch and bound tree are explored significantly affects how quickly the MIP is solved. In this paper, new methods are presented that exploit correlation and distribution characteristics of branch and bound trees to trigger backtracking and to choose the next node to solve when backtracking. A new method is also presented that determines when the cost of using a node selection method outweighs its benefit, in which case it is abandoned in favor of a simpler method. Empirical experiments show that these proposed methods outperform the current state of the art.

1 Introduction

Linear *Mixed Integer Programs* (*MIP*) can be formulated as follows.

The objective function is defined as:

$$\text{minimize } z = \sum_{j \in I} o_j x_j + \sum_{j \in C} o_j x_j$$

The constraints are defined as:

$$\sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \; \{\leq, =, \geq\} \; v_i, \quad i = 1, \ldots, m$$

$$\begin{array}{ll} l_j \leq x_j \leq u_j & j \in I \cup C \\ x_j \in \mathbb{Z} & j \in I \\ x_j \in \mathbb{R} & j \in C \end{array}$$

I is the set of variables constrained to be integer valued (including binary-valued).

C is the set of continuous, real valued variables.

Many methods for solving MIPs employ a branch and bound solution method. The general procedure of branch and bound is summarized in Algorithm 1. A difficulty with branch and bound is that it can take an impractical amount of time to solve some MIPs, even when using the most sophisticated computers. For this reason, the focus of this work is on developing faster methods for solving MIPs.

The performance of the branch and bound method is greatly affected by the choice of the *branching variable selection* method (Step 4), and the *node selection* method (Step 2). After solving the LP relaxation of the current node (Step 3), there will be a list of *candidate variables*, i.e. variables that are required to take integer values and yet are not integer-valued in the current LP-relaxation optimal solution. The branching variable selection method chooses a candidate variable from the list for branching. This produces two child nodes: in one child node the lower bound of x_j is set to the nearest integer value that is greater than the LP-relaxation solution value of x_j, and in the other child node the upper bound is set to the nearest integer that is less than the value of x_j.

In Step 2, if there are two or more unexplored (*active*) nodes then the node selection method determines which active node to solve next. As far as possible, the goal is to choose nodes that are ancestors of a MIP optimal node so that the resulting tree is as small as possible, though there is no guaranteed method for making an accurate selection. There are numerous node selection heuristics that try to achieve this goal, or that use some other strategy of developing the branch and bound tree to achieve a different goal, such as reaching a first feasible solution quickly. The node selection heuristic can have a dramatic effect on the solution effort. For example, for the *bell4* model from MIPLIB 2.0, GLPK does not solve this model within 1 hour using its default node selection method, whereas this model is solved in 10.66 seconds when using one of the new node selection methods developed here.

This paper develops new heuristics for node selection that demonstrate significant improvement over state of the art methods in solving MIPs to optimality quickly. There are three aspects to the new methods: (i) determining when to backtrack, i.e. to select a node other than a child of the node most recently explored, (ii) determining which node to backtrack to, and (iii) determining when to abandon an advanced but costly backtrack selection method in favour of simple depth-first search.

1.1 Existing Node Selection Heuristics

Since the LP-relaxation solution is not yet available, the lower bound of an unexplored node is initially set equal to that of its parent node. The same is true for any other LP solution characteristics of a node such as the number of candidate variables. Using the available information, node selection heuristics try to avoid choosing *superfluous nodes*, i.e. nodes whose lower bound turns out to be greater than the (unknown) MIP optimal objective value, z^*.

The *best-first*, or *best-bound*, node selection method avoids the exploration of superfluous nodes by choosing the unexplored node having the smallest lower bound over all the unexplored nodes. Nodes that are closer to the root node are more likely to be chosen because their lower bounds are generally smaller than the lower bounds of nodes deeper in the tree. Since MIP feasible solutions usually occur at leaf nodes that are far from the root node, best-first search is not a good choice for quickly finding a MIP feasible solution. This is

Algorithm 1 Branch and Bound

Inputs: MIP instance.
Initialize: Incumbent solution, $I = \phi$. Objective value of incumbent solution, $z(I) = \infty$. List of unexplored nodes, $N = \phi$.

```
Procedure:

1. Add the initial LP-relaxation (the root node) to $N$.
2. Choose a node from $N$ for exploration and label it $currentNode$.
3. Solve the LP-relaxation for $currentNode$.
     • If LP-relaxation is infeasible or it is feasible with a lower bound that is greater
       than $z(I)$ then discard $currentNode$ and go to Step 7.
     • If LP-relaxation is MIP feasible then:
         – If LP-relaxation objective function value is less than $z(I)$ then replace $I$ with
           this solution, else discard $currentNode$.
         – Go to Step 7
4. Choose a candidate variable in $currentNode$ for branching.
5. Branch on the selected variable to create two child nodes of $currentNode$; add these
   nodes to $N$. Remove $currentNode$ from $N$.
6. Go to Step 2
7. If list of unexplored nodes is empty then:
     7.1 If $I = \phi$, then exit with infeasible outcome.
     7.2 Optimum is $I$: exit with optimal outcome.
8. Go to Step 2
```

also true of the *breadth-first* node selection method, which chooses the earliest created active node.

Depth-first node selection always chooses a child of the most recently explored node. Given the two child nodes created by branching on a variable, a heuristic is used to choose which one to explore next. Some common methods for making this *branch direction* decision include choosing the up branch, the down branch, the branch that has the nearest integer bound for the branching variable, or the branch that forces the value of the branching variable away from its value at the root node [22]. If the last solved node is either LP infeasible, MIP feasible, or worse than the incumbent solution, then the last created active node is explored. *Depth-first* node selection is a better choice if the goal is to find a MIP feasible solution as quickly as possible.

A major advantage of depth-first exploration of the tree is that the LP formulations for a parent and a child node differ by only a single variable bound, which means that the solution of the child node can be hot-started, and hence is very quick. Other node selection methods that do not move from parent to child node in succession cause the solution of dissimilar LPs in succession, and hence the average number of simplex iterations to solve each node is usually significantly higher as compared to depth-first node selection [5, 11, 20]. This assumes that the factorized simplex basis of only the most recently solved node is available which, due to computer memory limitations, is true for most MIP solvers. Depth-first search also tends to maintain fewer unexplored nodes in memory, and hence is less likely to exhaust the available memory. Some MIP solvers, such as SCIP [3], use a predefined threshold of memory usage to ensure that the size of the branch and bound tree does not exceed the available memory by switching to depth-first node selection if this threshold is exceeded. A number of node selection schemes try to combine the advantages of the best-first and depth-first methods. One such scheme uses depth-first search until a MIP feasible solution is found and then switches back and forth between best-first and depth-first strategies [5].

The *most-feasible* node selection method chooses the node with the smallest sum of fractional values over all candidate variables. This is useful in seeking integer-feasible solutions.

Knowing the value of z of the best possible MIP feasible solution attainable at a descendent of a given node would be useful in selecting which node to explore. Methods which estimate this value, called *estimate based methods*, include the best-projection method [16, 23] and the best-estimate method [6, 14].

The best-projection method uses the change in the value of z between the root node and the incumbent solution as well as the change in the sum of integer infeasibilities between the root node and the node for which the estimate is being computed. The best projection estimate at node i is computed as follows:

$$E_i = z_i + \left(\frac{z_{inc} - z_0}{s_0}\right) s_i$$

where z_i is the lower bound of node i, z_{inc} is the value of z for the incumbent solution, z_0 is the lower bound of the root node, s_0 is the sum of integer infeasibilities at the root node, and s_i is the sum of integer infeasibilities at node i.

The best-estimate method uses *pseudo-costs*. Each integer variable has two pseudo-cost values associated with it, one for the up branch, P_j^U, and the other for the down branch, P_j^D. When a variable is branched on, the change in the lower bound between the parent node and the down child node is $\Delta z_j^D = z_j^D - z_j^P$ where z_j^D is the lower bound for the down child node and z_j^P is the lower bound for the parent node. Likewise for the up child node $\Delta z_j^U = z_j^U - z_j^P$. The pseudo-cost value for the down branch of x_j is

$$P_j^D = \Delta z_j^D / f_j$$

and similarly for the up branch

$$P_j^U = \Delta z_j^U / (1 - f_j)$$

where f_j is the fractional value of x_j. Note that f_j is greater than 0 since only variables that are not integer feasible can be branched on. Updating the pseudo-costs for x_j is typically done by averaging the values from every instance that x_j was branched on. The estimate of the best integer feasible solution attainable from a node i is calculated as follows:

$$E_i = z_i + \sum_{j \in I} \min\{P_j^D f_j, P_j^U (1 - f_j)\}.$$

In these methods the active node with the smallest such estimate is explored next. Linderoth and Savelsbergh [20] considered the pseudo-cost estimates to be optimistic, so they proposed an adjusted pseudo-cost estimate. Achterberg [2] proposed interleaving best-estimate with best-first node selection by performing best-first node selection once for every b_{fq} best-estimate backtracking node selections. It is suggested that $b_{fq} = 9$ should be used. Forrest et al [14] proposed a method that uses best-estimate node selection until a MIP feasible solution is found and then chooses nodes according to the percentage error criterion.

Backtracking node selection uses depth-first search until the current node is either MIP feasible, LP infeasible, has a lower bound that is greater than the value of z of the incumbent solution, or is considered undesirable to explore, at which point backtracking occurs. A backtracking node selection method is then used to select the next node to explore instead of choosing a child of the current node, typically either the best-first, best-projection, or best-estimate criterion [20]. Many implementations of the branch and bound method use an estimate $\tilde{z}^*$ of the MIP optimal objective value, called an *aspiration value*, in an attempt to avoid the exploration of superfluous nodes by triggering backtracking node selection when the lower bound of a node exceeds $\tilde{z}^*$. The aspiration value is typically set by the user before the branch and bound process begins. Various methods have been proposed to compute $\tilde{z}^*$ including best-projection estimates [20] as well as some estimation methods based on pseudo-costs [15, 20].

Kostikas and Fragakis [17] have experimented with using genetic programming to create customized node selection rules for a given MIP instance during the branch and bound solution process of this instance.

1.2 Room for Improvement

We performed a proof-of-concept experiment to test the potential merit of using a good aspiration value in a branch and bound procedure. We solved a set of MIP instances and recorded the MIP optimum values. We then re-solved the instances using an aspiration value equal to the MIP optimal objective value. This reduced the total time to solve all of the test instances by 15% relative to the best existing aspiration method. It also reduced the mean ratio to the best time to solve each MIP instance by 48% relative to the best existing aspiration method. This suggests that it is worthwhile to find ways of generating good estimates of the MIP optimal objective value for use in triggering backtracking.

2 Backtrack Triggering and Node Selection Heuristics Based on Frequent Patterns in MIP Solutions

The new methods developed in this paper are:

- the modified best-projection backtracking node selection and aspiration methods;
- the distribution backtracking node selection method;
- the feasibility depth extrapolation aspiration method;
- and the active node search threshold.

2.1 Modified Best Projection Node Selection and Aspiration Method

Our modification to the best-projection method estimates the objective value of the best MIP feasible solution attainable from any node using nodes that are not MIP feasible. This eliminates the need to find an incumbent solution before computing an estimate, as in the original best projection method. The modified version is used both to select a node during backtracking and to trigger backtracking. The modification is based on Observation 1.

A few definitions are needed. Let c_i be the number of candidate variables at node i. Let c_0 be the number of candidate variables at the root node. Let z_i be the objective value at node i. Let z_0 be the objective value at the root node. Let $z_{min}(c)$ be the smallest z_i over all nodes with c candidate variables:

$$z_{min}(c) = \min_i \{z_i : c_i = c\}.$$

Observation 1 ***(Node Infeasibility versus Optimality)***
There is an approximately linear correlation between $z_{min}(c)$ and c for many MIPs. ■

For many MIP instances, plots of $z_{min}(c)$ versus c show a trend that indicates the possibility of estimating the value of z^* early in the branch and bound process by simple linear extrapolation. Note that z^* occurs at $z_{min}(0)$. For the *dcmulti* MIP instance, the trend can be seen in Figure 1 for $10 \leq c \leq 30$.

Observation 1 underlies our improvement to the best-projection method. Recall that in the best-projection method an incumbent solution is required to compute an estimate of the degradation in objective value per unit MIP infeasibility. At any time during the branch and bound process, the incumbent solution objective value is $z_{min}(0)$ which is the minimum objective value found at a node with 0 candidate variables, i.e. a MIP feasible node. According to Observation 1, a useful estimate of the degradation in objective value per unit

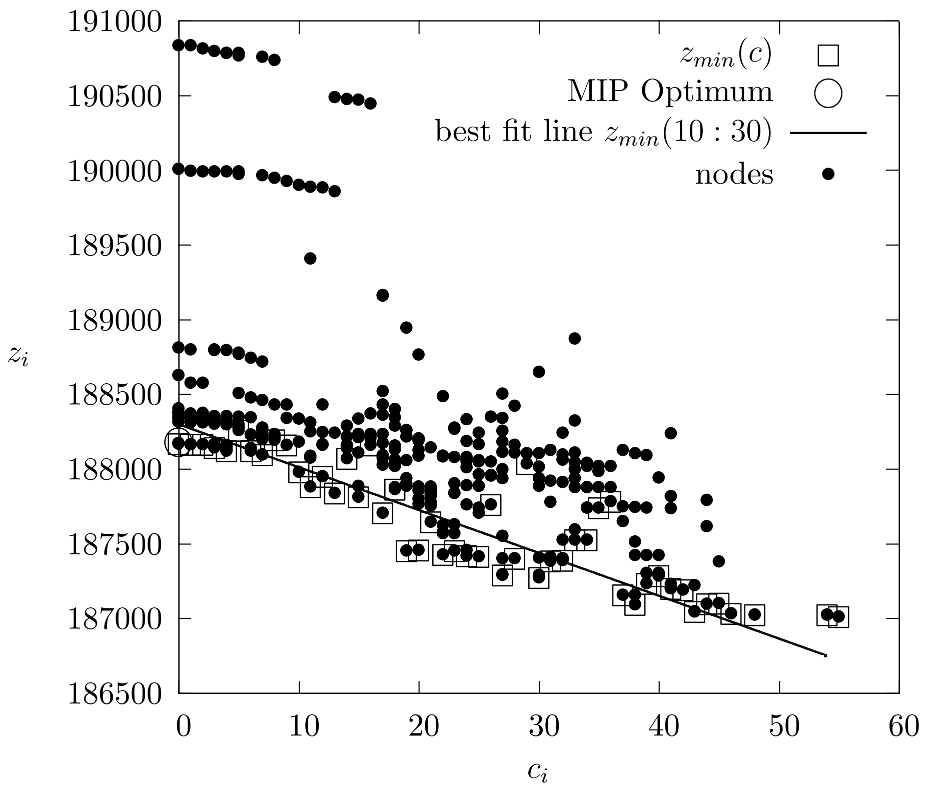

Figure 1: Plot of z_i and $z_{min}(c)$ versus c_i for all of the nodes in the tree at the completion of the branch and bound process for the *dcmulti* MIP instance. Also shown is the best fit line through $z_{min}(c)$ for $10 \leq c \leq 30$. The $BPr_0 - Lf - N$ node selection configuration is used (see Section 3.4 for the configuration definitions).

MIP infeasibility can be made using $z_{min}(c)$ for $c > 0$ thus allowing modified best-projection node selection to proceed without an incumbent solution.

The following definitions are needed for this algorithm. Let c_{min} be the minimum number of candidate variables over all nodes solved so far. This value is initially equal to the number of candidate variables at the root node and is updated after each node is solved. Let σ_z be the standard deviation of z_i over all nodes solved so far. Let σ_c be the standard deviation of c_i over all nodes solved so far.

The estimate of degradation of the objective value per unit change in MIP infeasibility, m, is computed as follows:

$$m = \frac{z_{min}(c_{min}) - z_0}{c_0 - c_{min}}. \tag{1}$$

The estimate of the objective value of the best MIP feasible solution attainable from a node i, $\tilde{z}_i^*$, is computed as follows:

$$\tilde{z}_i^* = c_i * m + z_i.$$

The value of $-m$ is the slope of the line through $(c_{min}, z_{min}(c_{min}))$ and (c_0, z_0). Figure 2 shows an example of this line at a time in the branch and bound process when $c_{min} = 23$, $c_0 = 55$ (the number of candidate variables at the root node), $z_{min}(c_{min}) = 187,851.53$, and $z_0 = 187,022.36$ (the objective value at the root node). The slope of the line through the points $(c_{min}, z_{min}(c_{min}))$ and (c_0, z_0) is $-m = -25.91$. At this time the value of $z_i = z_{min}(c_{min})$ for the most recently solved node i. Likewise, the value of $c_i = c_{min}$. An estimate of the objective value of the best MIP feasible solution attainable from node i, is $\tilde{z}_i^* = 188,447.5$. This estimate is the extrapolated value of $z_{min}(0)$ using the line through (c_i, z_i) with a slope equal to $-m$.

If at least one of the following conditions is satisfied then this method does not work well.

- $c_{min} = c_0$. In this case the denominator in (1) is 0.
- $\sigma_c < \sigma_c^{min}$, where σ_c^{min} is the minimum acceptable value of σ_c. In this case most nodes have very similar values of c_i. This causes the value of m to have little effect on node selection. We set $\sigma_c^{min} = 3$.
- $\sigma_z < \sigma_z^{min}$, where σ_z^{min} is the minimum acceptable value of σ_z. In this case there is very little change in z_i over all nodes. This causes the value of m to be almost 0 which then results in c_i having little effect on node selection. We set $\sigma_z^{min} = 0.001$.

The thresholds used for the last two conditions were determined empirically. None of the MIP instances where modified best-projection worked well satisfied any of these conditions. In most cases where it did not work well, at least one of these conditions was satisified.

2.2 Distribution Node Selection

This backtracking node selection method balances the goals of MIP feasibility and MIP optimality. It uses probability distributions of measures of MIP feasibility and MIP optimality that are based on data taken from all nodes solved so far in the solution process. This

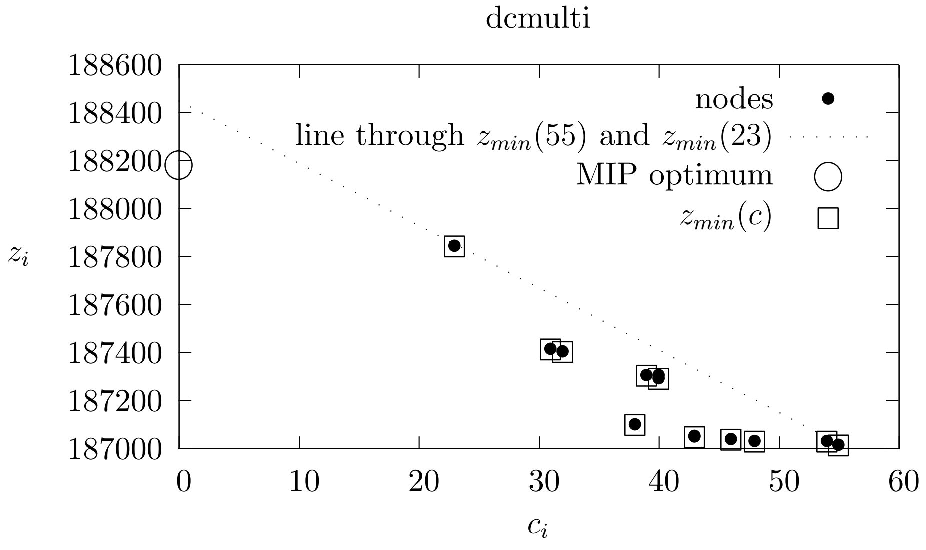

Figure 2: Plot of z_i and $z_{min}(c)$ versus c_i for all of the nodes in the tree at the time in the branch and bound process when $c_{min} = 23$ for the *dcmulti* MIP instance. Most of the points in the plot overlap because $z_i = z_{min}(c)$ for most of the nodes at this time. Also shown is the line through $z_{min}(55)$ (the root node) and $z_{min}(23)$.

approach weights the MIP feasibility and optimality goals dynamically so that neither can dominate the other for too long. It favours choosing nodes with lower objective values and less MIP infeasibility.

The most commonly used measure of a node's closeness to optimality is its value of z. The measure of MIP infeasibility used in this method is the number of candidate variables c. The number of candidate variables is used instead of the sum of the fractional components of the integer variables because a candidate variable will likely become integer valued if that variable is branched on regardless of what its fractional value is. For example, suppose one node has one candidate variable with a fractional value of 0.5 whereas another node has five candidate variables each with a fractional value of 0.09. If all other integer variables are fixed for both of these nodes, then choosing the first node will likely lead to a MIP feasible node more quickly than choosing the second node, which has a lower sum of integer infeasibilities.

Observation 2 ***(Balancing MIP Feasibility and Optimality)***
A characteristic that is typical of many branch and bound trees is that there is a negative correlation between the objective value and the number of candidate variables of a node; i.e. z_i tends to increase with depth while c_i tends to decrease with depth. ■

A consequence of Observation 2 is that when trying to balance MIP feasibility and MIP optimality during node selection, if greater weight is being given to z_i than to c_i, then the fraction of nodes chosen with small z_i will likely be large relative to the fraction of nodes chosen with small c_i. The opposite is likely true if greater weight is given to c_i than to z_i. Note that MIP feasibility is reached when $c_i = 0$.

Since it is desirable to choose nodes with both small z_i and small c_i, these values need to be combined into a single value in order to rank all of the active nodes. This combined value should be directly proportional to both z_i and c_i. Combining z_i and c_i so that neither dominates the other can be difficult due to the difference between the range of z_i and the range of c_i across all of the nodes. For example, Figure 1 shows that the range of z_i for *dcmulti* is over 3000, whereas the range of c_i is less than 60. This problem is solved by assembling the probability distributions of z_i and c_i, respectively, as data becomes available during the solution process. Note that the range of each distribution is $[0, 1]$ and that each distribution is a monotonically increasing function of its independent variable.

The distribution function used in this method is suggested by the Central Limit Theorem, which states that, under certain conditions, the distribution of a sum of random variables tends to a Normal distribution. Although the underlying process that determines the value of each variable at a node is not random, it is not usually easy to predict the value of every variable at a node before it is solved. Therefore the variables can be thought of as random variables whose values are measured each time an active node is solved. The value of z_i for a node is a weighted sum of these variables, therefore a Normal distribution is used to estimate the distribution of z_i in a branch and bound tree. The value of c_i can be thought of as the sum of a set of random variables where each integer variable adds either 0 or 1 to c_i depending on whether it is integer or fractional valued respectively. Therefore, a Normal distribution is used to estimate the distribution of c_i. In addition, the Normal distribution is well suited for balancing the pursuit of MIP feasibility with optimality since (i) it requires little more work to compute than the Uniform distribution, (ii) it estimates the values of z_i

and c_i that most frequently appear in the tree using their respective means m_z and m_c, and (iii) it uses information on the ranges of z_i and c_i via their respective standard deviations σ_z and σ_c.

Let $F_Z(z) = P(Z \leq z)$ and $F_C(c) = P(C \leq c)$ be estimates of the Normal cumulative distribution functions for z_i and c_i, respectively, over all nodes solved so far. The value of $F_Z(z_i)$ is the estimated fraction of nodes with a value of $z \leq z_i$. Similarly, the value of $F_C(c_i)$ is the estimated fraction of nodes with a value of $c \leq c_i$. The distribution backtracking node selection method chooses an active node with the smallest value of $F_{ZC}(z_i, c_i) = F_Z(z_i) \times F_C(c_i)$ to be explored next. This $F_{ZC}(z_i, c_i)$ is used as the comparison criterion because its value is directly proportional to both $F_Z(z_i)$ and $F_C(c_i)$, and $F_{ZC}(z_i, c_i) \leq \min(F_Z(z_i), F_C(c_i))$ (recall that $0 \leq F_Z(z_i) \leq 1$, and $0 \leq F_C(c_i) \leq 1$) which causes this node selection method to behave a little like a combination of the best-first and the most-feasible node selection methods.

The following example demonstrates how the pursuit of MIP feasibility and optimality can be balanced using Normal distributions. Suppose that during backtracking node selection there are 4 LP feasible nodes in the tree, two of which have 1 unsolved child each. The values of z_i and c_i for each LP feasible node are given below.

i	z_i	c_i	child
0	1	100	n1
1	100	1	
2	100	1	
3	100	1	n2

In this example, one of the candidate nodes is the unsolved child $n1$ of node 0, and the other is the unsolved child $n2$ of node 3. Recall that the value of z_i and c_i at an unsolved node is initially set to its parent's value of z_i and c_i respectively. The choice of which node to solve next is between node $n1$ which has a small value of z_i and a large value of c_i, and node $n2$ which has a large value of z_i and a small value of c_i. To emphasize the optimality goal, we would like to choose the node i that has the smallest fraction of nodes with $z \leq z_i$, but to emphasize the feasibility goal, we would like to choose the node i having the smallest fraction of nodes with $c \leq c_i$. Estimates of these fractions are computed for our example using $F_Z(z_i)$ and $F_C(c_i)$, and are given below along with the product of these fractions $F_{ZC}(z_i, c_i)$ for each candidate node.

i	$F_Z(z_i)$	$F_C(c_i)$	$F_{ZC}(z_i, c_i)$
$n1$	0.067	0.933	0.062
$n2$	0.691	0.309	0.213

Node $n1$ has the smallest value of $F_{ZC}(z_i, c_i)$ so it is chosen to be solved next. Node $n1$ is likely the better choice because three of the nodes have a large value of z_i and a small value of c_i whereas one of the nodes has a small value of z_i and a large value of c_i. This indicates that more emphasis has been placed on the pursuit of MIP feasibility than the pursuit of optimality so the emphasis should be shifted towards the pursuit of optimality by selecting node $n1$ to be solved next.

The distribution backtracking node selection method computes $F_{ZC}(z_i, c_i)$ for each active node i. The node n that will be explored next is chosen as follows:

$$n = \arg\min_i F_{ZC}(z_i, c_i).$$

If there is very little variation in the value of z_i, or if c_i does not change significantly in proportion to the depth of the nodes in the tree, then this method does not work well since one of the measures will be given too much weight in node selection. This is summarized in the following two conditions. Let d be the depth of the latest solved node before backtracking node selection.

- $\sigma_c/d < \sigma_{cd}^{min}$: In this case most nodes have very similar values of c_i over a large range of depths in the tree. We set $\sigma_{cd}^{min} = 0.1$.
- $\sigma_z < \sigma_z^{min}$: In this case there is very little change in z_i over all nodes. We set $\sigma_z^{min} = 0.001$.

The thresholds used for these conditions were determined empirically. No instances where distribution node selection worked well satisfied either condition. Most instances where distribution node selection did not work well satisfied at least one of these conditions.

2.3 Feasibility Depth Extrapolation Aspiration

This method estimates the MIP optimal objective value by first estimating the depth of the MIP optimal node via linear extrapolation of the number of candidate variables along the ancestors of each node in the branch and bound tree. The MIP optimal objective function value is then estimated based on the node optimum values at that depth. This method is based on Observations 3, 4 and 5.

Observation 3 ***(MIP Optimal Depth and Objective Value)***
The largest objective value over all nodes at the depth of a MIP optimal node in the branch and bound tree is greater than or equal to the MIP optimal objective value. ■

Observation 4 ***(Relative Depth of MIP Optimal Node)***
The MIP optimal node in the branch and bound tree tends to be closer to the root node than most other MIP feasible nodes (i.e. less deep than other MIP feasible nodes). ■

Observation 5 ***(Node Infeasibility versus Depth)*** *Extrapolating the rate of decrease of the number of candidate variables along a dive in the branch and bound tree provides a reasonable estimate of the depth of the first MIP feasible solution for the dive.* ■

Figure 3 demonstrates Observation 3 for the *bell5* instance. As shown, the largest objective value over all nodes at the MIP optimal depth is less than the objective values of a significant number of nodes in the tree. Therefore, if the largest objective value over all nodes at the MIP optimal depth is known and is used as an aspiration value, then the amount of effort required to solve this MIP instance will be reduced, without danger of eliminating the optimal solution. Figure 3 also shows that nodes that are close to a MIP optimal depth tend to have objective values that are relatively close to the MIP optimal objective value. This suggests a method of estimating a suitable aspiration value. Finally, Figure 3 shows that

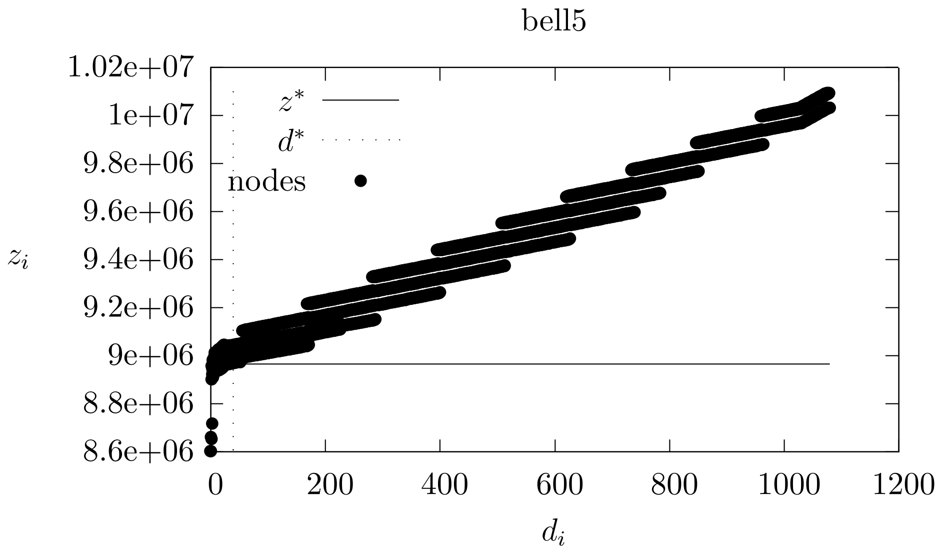

Figure 3: A plot of the objective value versus depth for all nodes for the *bell5* MIP instance. Also shown are lines representing the MIP optimal objective value and depth. The $Dist - Lf - N$ node selection configuration is used.

in some instances the estimate of the depth of the optimal node does not have to be very accurate in order to get a useful aspiration value.

To demonstrate Observation 4, all MIP instances were solved to optimality, using default GLPK with a 1 hour time limit, recording the depth of the optimal node as well as the depth of the closest MIP feasible node to the root node. Figure 4 shows that for a majority of MIP instances, the optimal node is close to the shallowest MIP feasible node in the branch and bound tree.

This pattern exists because fewer changes in variable bounds between the initial LP relaxation and a MIP feasible solution usually lead to less change in objective values between the two solutions. Hence a shallower MIP feasible solution tends to have a better objective function value, so the shallowest MIP feasible solution is often the optimum. Because of this, the rate of increase of MIP feasibility with depth tends to be greater for the ancestors of the optimal node than for other MIP feasible nodes.

If the depth d_i^* of the shallowest MIP feasible node attainable by exploring the descendants of node i is known, then the shallowest of these depths over all nodes in the tree is likely a MIP optimal depth. According to Observation 5, an estimate of the value of d_i^* (called $\tilde{d}_i^*$), may be computed using the correlation between the number of candidate variables c_i and the depth d_i.

Figure 5 shows an example in which changes in the number of candidate variables as one progresses along a dive deeper into the tree give some indication of the depth of a MIP feasible node at the end of this dive. Any node in the branch and bound tree is likely more MIP feasible than its parent because branching on a candidate variable in the parent node almost always results in that variable being integer valued in the resulting child node. Thus, it is expected that MIP infeasibility should decrease as one moves along a path deeper into the tree.

For many MIP instances, plots of c_i versus d_i of all nodes on a given dive show some linearity. Figure 6 shows an example for a dive in the *vpm2* MIP instance. This plot also shows that extrapolating a linear best-fit line on c_i for $0 \leq d_i < d_i^*$ can result in a good value of $\tilde{d}_i^*$. This pattern does not hold for all MIP instances, but it holds frequently enough to be useful as a heuristic. Linear extrapolation is used to estimate the value of $\tilde{d}_i^*$ of a given node i, i.e. the depth at which the number of candidate variables reaches zero. Figure 6 shows that $d_i^* = 44$ and $\tilde{d}_i^* - d_i^* = -8$ for the node at $d_i = 20$. An estimate of the depth of the optimal solution $\tilde{d}^*$, according to Observation 4, is then taken as the smallest $\tilde{d}_i^*$ over all solved nodes in the tree.

The aspiration value $\tilde{z}^*$, according to Observation 3 is set to the maximum objective value over all solved nodes at $\tilde{d}^*$.

The following definitions are needed for this algorithm: m_i is the slope of the least squares best fit line for (c_i, d_i) over all ancestors of node i, and b_i is the d-intercept of the least squares best fit line $(d = m_i \cdot c + b_i)$ over all ancestors of node i. S is the set of solved nodes in the current branch and bound tree that are LP feasible but not MIP feasible.

The algorithm proceeds by computing the least squares best fit line of the ancestors for each solved node with at least 20 ancestors; this ensures that enough information is available to compute a reasonably accurate best fit line while still being able to compute an aspiration value early in the branch and bound process. An estimate of the depth of the first MIP

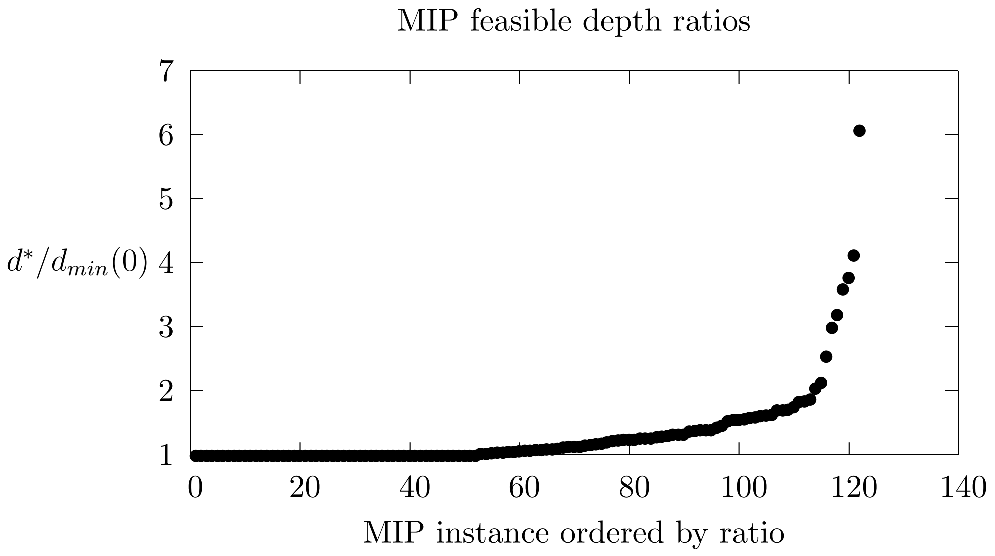

Figure 4: A plot of the ratio of depth d^* of a MIP optimal node to the minimum depth $d_{min}(0)$ over all MIP feasible nodes for a set of MIP instances. The $BPr_0 - Lf - N$ node selection configuration is used.

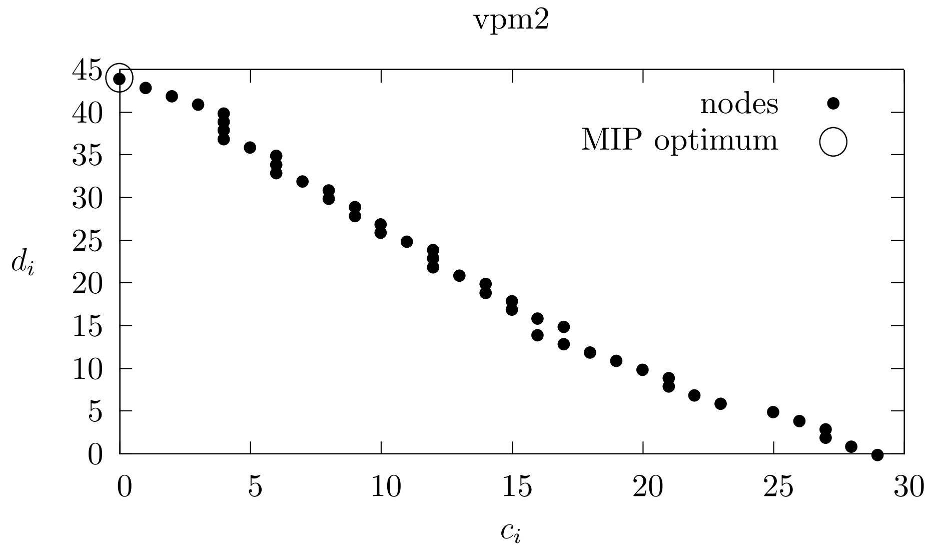

Figure 5: A plot of the number of candidate variables versus depth for all ancestors of a MIP optimal node for the *vpm2* MIP instance.

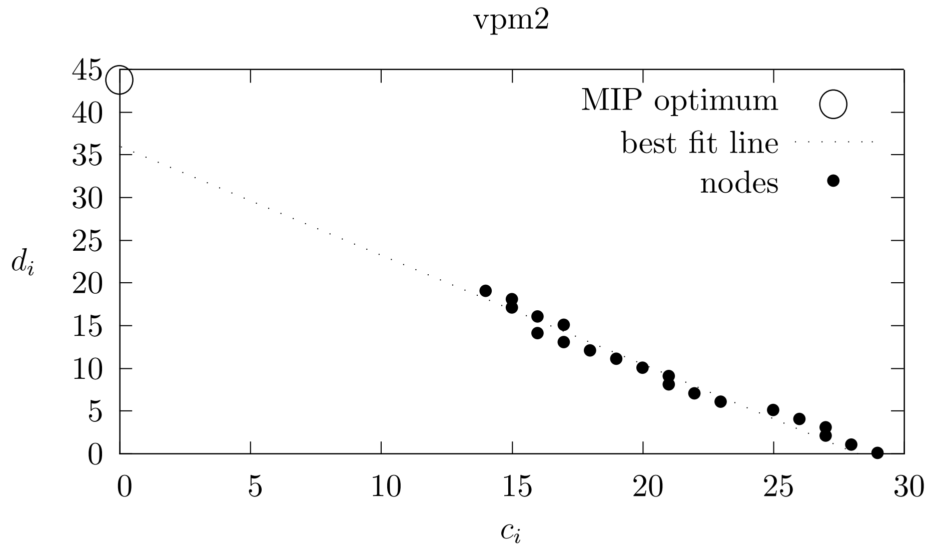

Figure 6: A plot of the number of candidate variables versus depth for the earliest 20 ancestors of a MIP optimal node for the *vpm2* MIP instance along with the best fit line.

feasible solution attainable from a node i is computed as

$$\tilde{d}_i^* = [b_i]$$

where $[b_i]$ represents the rounding of b_i to the nearest integer value.

The estimate of the optimal solution depth $\tilde{d}^*$ is computed as

$$\tilde{d}^* = \min_{\{i \in S : d_i \geq 20, \tilde{d}_i^* > 0\}} \tilde{d}_i^*. \tag{2}$$

The aspiration value $\tilde{z}^*$, according to Observation 3, is the maximum value of z_i over all solved nodes with $d_i = \tilde{d}^*$:

$$\tilde{z}^* = \max_{i \in S : d_i = \tilde{d}^*} z_i.$$

The condition that $\tilde{d}_i^* > 0$ is imposed in (2) because non-positive values would result in unreasonable values of $\tilde{d}^*$. A value of $\tilde{d}^* = 0$ is unreasonable since the root node is not MIP feasible. A value of $\tilde{d}^* < 0$ is unreasonable since there are no nodes with $d_i < 0$.

2.4 Active Node Search Threshold

Some backtracking node selection methods can be computationally expensive. The active node search threshold (ANST) estimates whether or not the cost of using an expensive backtracking node selection method outweighs its benefit. If the cost does outweigh the benefit, then a simpler node selection method is used instead. This idea is based on Observation 6.

Observation 6 ***(Cost of Node Selection Methods)*** *For some MIPs, the amount of time required to perform a given backtracking node selection method can become a significant fraction of the overall amount of time required to solve the problem. This occurs when the number of active nodes in the tree becomes very large.* ■

For example, the MIP instance *mas76* is solved in 20,609 seconds, requiring 3,186,117 simplex iterations and 1,177,063 nodes, using best-projection backtracking node selection. The same MIP instance is solved in 1,306 seconds, requiring 3,314,480 simplex iterations and 1,229,699 nodes using depth-first node selection. This MIP is solved in significantly less time but requiring more simplex iterations and nodes using depth-first versus best-projection.

To verify that the extra time needed to solve MIPs such as *mas76* using best-projection versus depth-first backtracking is due to the node selection method, the maximum ratio, R_t, of total time spent performing all of the best-projection backtracking node selections to the total time spent performing all of the other branch and bound operations was recorded for this MIP instance as well as for others. The amount of effort needed to solve each MIP instance in a small set is shown in Table 1 along with the corresponding value of R_t. The values of R_t for MIPs that are solved in less time but more simplex iterations using depth-first versus best-projection are much larger than the values of R_t for the remaining MIPs.

For the *mas76*, *pk1*, and *ran10x10c* MIP instances, the cost of using best-projection node selection outweighs the benefit because this method searches the entire set of active nodes and the number of active nodes in the tree becomes very large. The cost of using a *simple* backtracking node selection method does not increase with the number of active nodes in the tree. In GLPK [21], the depth-first backtracking node selection method is simple and requires very little computation because it chooses the active node that was last created and resides at the end of the list of active nodes. A backtracking node selection method that searches through the entire set of active nodes is not simple because the amount of effort required to search through the set increases as the number of nodes in the set increases. Examples of backtracking node selection methods that are not simple include the best-projection, best-first, and best-estimate methods.

An alternative to putting the last created active node at the end of the list is used in the SCIP solver [2, 3]. Each node is placed in a list according to its priority with respect to the node selection criteria. A drawback of this method is that the relative priority of nodes already in the list may change depending on the node selection criteria used. In the best-estimate node selection method, for example, the pseudo-cost estimates are updated after every node is solved. To ensure that the nodes in the list are correctly ordered, the best-estimate for each node in the list affected by the changed pseudo-cost estimates should be re-calculated and then all of the active nodes should be re-sorted.

In SCIP, the amount of computation required to place each node in the list according to its node selection priority increases with the number of nodes already in the list since a search of these nodes needs to be performed every time a new node is placed in the list. Therefore it is possible for the cost of node selection methods of this type to become very large. This issue is noted by Achterberg [2].

The cost of a backtracking node selection method is mainly affected by the number of active nodes in the tree, the amount of computation required per active node during the backtracking search, and by how frequently backtracking node selection is performed. If an aspiration backtrack triggering method is used then backtracking frequency is affected by the aspiration value. Therefore, if the cost of backtracking node selection is perceived to outweigh its benefit for a given MIP, and an aspiration method is being used, it may be better to first reduce the backtracking frequency by increasing or removing the aspiration value to see if backtracking continues to cost too much. If the backtracking node selection method continues to cost too much, then it should be changed to a simple method. This is one way in which an aspiration value that is too small may be detected.

The following definitions are needed for this algorithm. Let t_{NS} be the total amount of time spent performing all backtracking node selections. Let t_{BB} be the total amount of time spent performing all of the operations in the branch and bound method. Let q be the threshold that determines when to switch to a simple node selection method, set at $q = 0.1$. Let dq be an increment that is used to determine the effect of the current aspiration backtrack triggering method on the cost of backtracking node selection. We set $dq = 0.01$.

The active node search threshold algorithm computes the value of R_t after each backtracking node selection is performed:

$$R_t = \frac{t_{NS}}{t_{BB} - t_{NS}}.$$

If $R_t \geq q$ then either backtracking is happening too frequently because of an aspiration backtrack triggering method, or the cost of backtracking outweighs its benefit. If aspiration backtrack triggering is not being used, then the current backtracking node selection method is too costly, and depth-first backtracking node selection replaces the current node selection method. If an aspiration backtrack triggering method is in use, then it is discontinued, the current backtracking node selection method continues, and the value of q is set to the current value of $R_t + dq$ so that now $R_t < q$. If the value of R_t continues to increase and exceeds this new value of q then the current backtracking node selection method is too costly and the depth-first backtracking node selection method is substituted.

Depth-first is used as the simple backtracking node selection method because of its low cost and efficiency in exploring the branch and bound tree.

Table 1 shows that setting $q = 0.1$ allows a given node selection method to continue to be used when it is not too costly but quickly forces a change to the depth-first method when its cost becomes too great. Although the values of R_t for a small number of MIP instances were used to determine the value of q, the chosen value of $q = 0.1$ is over 4 times larger than the next smaller value of R_t. dq is set to a small value that, if an aspiration value is being used, is added to R_t to make a new threshold that allows the current backtracking node selection method to continue to be used without an aspiration value. dq is set to 10% of the value of q to allow the current backtracking node selection method to continue to be used a sufficient number of times without an aspiration backtrack triggering method so that the backtracking node selection method is not mistakenly determined to be too costly.

3 Experimental Setup

3.1 Hardware

The experiments were carried out on a PC equipped with an Intel Core 2 6600 CPU running at 2.4 GHz and equipped with 4 GB of RAM. The operating system was Linux 2.6.18. Although the CPU has two processing cores, each MIP instance was solved using only 1 core.

3.2 Implementation

The GLPK 4.9 [21] MIP solver was used in all experiments for a number of reasons. It is free open source software that has excellent documentation. It uses the depth-first with backtracking framework, and has most of the standard node selection methods. Most importantly, it is easy to modify to add new backtracking node selection methods, aspiration methods, and algorithms such as the active node search thresholds. Finally, it is capable of solving reasonably difficult MIP instances within a reasonable time limit [19].

The GLPK solver has many parameters that do not affect the branch and bound solution method. Unless otherwise stated, all of these parameters are set to their default values except for those mentioned here.

LPX_K_TMLIM, the time limit of the MIP solver in seconds, is set to different values for different experiments as described later. LPX_K_MSGLEV is set to 1 so that messages to the console are limited to error messages in order to reduce the corresponding overhead and thus reduce the solution time for all MIP instances.

Table 1: Solution time, number of nodes, and number of simplex iterations for best-projection ($BPr_0 - Lf - N$) vs. depth-first ($DeF - Lf - N$) node selection. Also shown is R_t. The best values for each metric in each row are shown in boldface.

	depth-first			best-projection			
MIP	time (s)	itns	nodes	time (s)	itns	nodes	Max R_t
bal8x12	4.51	5375	1483	**1.73**	**2592**	**376**	0.0185
10teams	1985.79	1250128	14258	**1076.87**	**711067**	**6043**	0.00194
fiber	1434.93	276720	37298	**617.71**	**129490**	**15673**	0.0060
l152lav	706.34	113407	8168	**515.35**	**82776**	**5333**	0.0025
blend2	679.12	1649444	316634	**40.27**	**49393**	**18186**	0.0202
pipex	2.32	14695	5701	**1.39**	**9936**	**3007**	0.0215
misc07	4171.5	1464603	131477	**2937.81**	**1058824**	**93303**	0.0142
pp08a	1932.69	515883	71946	**1527.03**	**417930**	**53975**	0.0146
bienst1	2673.93	4107520	34256	**2530.14**	**3971433**	**32621**	0.0020
mas76	**1306.61**	3314480	1229699	20609.65	**3186117**	**1177063**	0.8790
pk1	**4058.23**	9778734	1965503	12623.15	**4329434**	**820924**	0.7904
ran10x10c	**930.96**	1214630	566587	3095.58	**907230**	**428535**	0.6552

For all experiments, the default branching variable selection method in GLPK is used because an empirical study that we performed suggested that this is the best option. For a similar reason, the root node cuts available in GLPK are used for all experiments.

If the amount of memory used when solving a MIP instance exceeds 1.5 gigabytes then the depth-first backtracking node selection method without an aspiration backtrack triggering method is used until the memory usage drops below 1 gigabyte. This technique is commonly used to avoid using the extremely slow swap space of memory [3]. This threshold was never exceeded in our experiments.

3.3 Comparison Metrics

Solution time t_{MI} is the most important measure of the effort required to solve a MIP instance MI and is frequently used to evaluate the performance of a branch and bound method [1, 2, 7, 8, 20]. The number of nodes solved and the number of simplex iterations are also commonly used to measure effort [1, 2, 7, 13, 24]. Solution time is used for the experiments in this paper because the time to solve a MIP instance includes overhead computing time that is not reflected in the number of nodes and the number of simplex iterations. Section 2.4 shows an example of how the time spent performing node selection may become large relative to that spent performing the other aspects of branch and bound. Table 1 shows that although best-projection node selection solves the *mas76*, *pk1* and *ran10x10c* MIP instances in fewer nodes and simplex iterations than depth-first node selection, these MIP instances are solved in significantly less time with depth-first node selection. The number of nodes and simplex iterations do not always accurately reflect the amount of effort required to solve a MIP instance.

To evaluate the relative performance of a set of branch and bound methods, a set of many MIP instances should be solved using each method because the heuristic nature of these methods does not guarantee that the relative performance of each method will be consistent for every possible MIP instance. A way to combine the solution times for all of the MIP instances into a single performance measure is needed. A simple approach is to compute the total solution time over all of the instances:

$$TOTTM = \sum_{MI} t_{MI}.$$

$TOTTM$ can be dominated by a small number of very difficult MIP instances. This effect can be eliminated by using the best solution time t_{MIB} for MI over all of the contending methods to normalize t_{MI}:

$$r_{MI} = t_{MI}/t_{MIB}.$$

For a given method, $r_{MI} - 1$ is the fraction of extra time required to solve MI relative to the best method for solving MI. The geometric mean is used to combine the r_{MI} for every MI into a single measure $MRATE$. The geometric mean is preferable to the arithmetic mean because it is less sensitive to outlying values of r_{MI}. For example, the arithmetic means of each of the sets $A = \{1, 2, 3, 4, 5\}$ and $B = \{1, 1, 1, 1, 11\}$ are both equal to 3 whereas the geometric means are 2.61 and 1.62 respectively. The outlying value 11 in set B has less impact on the geometric mean than on the arithmetic mean.

If a time limit is imposed on solutions, then not all MIP instances will be solved in a given experiment. The number of MIP instances $FAIL$ that are not solved to optimality is a commonly used performance measure for a branch and bound method [8, 20].

The details of computing $FAIL$, $TOTTM$ and $MRATE$ for a set of MIP instances are given below.

$FAIL$**:** The number of MIP instances used in an experiment that are not solved to optimality within the time limit. If a MIP instance is not solved using any contending method, then this MIP instance is excluded from $FAIL$. Methods with a smaller value of $FAIL$ are better.

$TOTTM$**:** The total amount of time required to solve the set of MIP instances. Any MIP instance that is not solved using any contending method is excluded from this total. If a MIP instance is not solved using a given contending method but is solved using at least 1 other contender, then the time limit is added to $TOTTM$ for this instance. $TOTTM$ is, therefore, a lower bound on the total time required to solve the set of MIP instances if a method fails on at least one MIP instance (it is exact if the method solves all instances). Methods with a smaller value of $TOTTM$ are better.

$MRATE$**:** The geometric mean of t_{MI}/t_{MIB} over all solved MIP instances for a given contending method. Any MIP instance that is not solved using any contending method is excluded from this mean. If a MIP instance is not solved using a given contending method but is solved using at least 1 other contender, then t_{MI} = the time limit. $MRATE$ is, therefore, a lower bound if a method fails on at least one MIP instance (it is exact if the method solves all instances). If t_{MI} is smaller than 10 seconds for every contending method then MI is excluded from $MRATE$ to reduce the effects of timing error on this measure. Methods with a smaller value of $MRATE$ are better.

Another way to combine the solution times of MIP instances in order to evaluate the relative performance of contending methods is to use a *performance profile* [12]. A performance profile for a contending method is a plot that displays a point representing each MI that is solved. For a given MIP instance $MI1$ and contending method, the x-axis represents the value of r_{MI1} and the y-axis represents the fraction f_{MI1} of MIP instances in the set that have a value of $r_{MI} \leq r_{MI1}$ for a particular method.

Commonly used performance measures for prematurely halted MIP instances are the best incumbent solution found and proven optimality gap at termination. To determine how the competitor methods performed on the more difficult MIP instances, the following scheme is used to determine the relative ranking of two methods for a given MIP instance. Ties are allowed.

- If both methods find a MIP feasible solution then the better method is the one that finds the solution with the best value of the objective function.

- If both methods find equally good MIP feasible solutions then the method with the smaller provable optimality gap (see Definition 1) is better.

- A method is better if it finds a MIP feasible solution whereas the other method does not.

Definition 1 ***(Provable optimality gap)***
The provable optimality gap G_o measures the proximity of the upper and lower bounds on the MIP optimal objective value. The value of G_o is computed as follows:

$$G_o = \frac{|z_{inc} - z_g^{min}|}{|z_{inc}| + \epsilon}$$

where z_g^{min} is the minimum objective value over all active nodes, z_{inc} is the objective value of the incumbent solution, and $\epsilon = 10^{-9}$ (used to avoid dividing by 0) .
A MIP is solved if $G_o = 0$. ■

The following measures are based on the ranking scheme described above.

AVGRANK is the average rank of a given method over all MIP instances that are not solved using any contending method. MIP instances for which no incumbent solution is found by any contending method are also excluded. Methods with a smaller value of *AVGRANK* are better.

NFIRSTS is the number of MIP instances for which a given method is ranked first. MIP instances that are solved by at least 1 contending method are excluded. MIP instances for which no incumbent solution is found by any contending method are also excluded. Methods with a smaller value of *NFIRSTS* are better.

NINC is the number of MIP instances for which a given method did not find an incumbent solution. MIP instances that are solved by at least 1 contending method are excluded. MIP instances for which no incumbent solution is found by any contending method are also excluded. Methods with a smaller value of *NINC* are better.

3.4 Experiments

The purpose of these experiments is to test all configurations of backtracking node selection, backtrack triggering, and active node search threshold methods to determine which is the best to use for solving MIP instances. All possible configurations are tested to assess how well each option for each component of a configuration performs in conjunction with various options for the other components.

The options for backtracking node selection method, backtrack triggering method and active node search threshold are described below.

- Backtracking node selection methods:
  - State of the art methods that are available in GLPK.

    DeF: Depth-first

    BrF: Breadth-first

    BPr_0: Default best-projection (GLPK default)

  BeF: Best-first

  - State of the art methods that were added to GLPK for this experiment.

    BeE: Best-estimate

    BeF/BeE: BeF interleaved with BeE

  - New methods that are proposed in this paper.

    $Dist$: Distribution (see Section 2.2)

    BPr_1: Modified best-projection (see Section 2.1)

- Backtrack triggering methods:

  - State of the art methods that are available in GLPK.

    Lf: Non-aspiration backtracking: trigger backtracking only at leaf nodes (GLPK default)

  - State of the art methods that were added to GLPK for this experiment.

    E: Perform backtracking node selection after every node solution

    $BPrA_0$: Default best-projection aspiration

    PCA: Pseudo-cost (best-estimate) aspiration

  - New methods that are proposed in this paper

    $DExA$: Linear feasibility depth extrapolation aspiration (see Section 2.3)

    $BPrA_1$: Modified best-projection aspiration (see Section 2.1)

- Active node search threshold (see Section 2.4) options:

  N: Do not use ANST (GLPK default)

  Y: Use ANST

State of the art methods not listed above are excluded because (i) there are no documented empirical studies in which the method outperforms all of the above methods, (ii) the principal behavior of the method relies on user defined parameters, or (iii) the details required to properly implement the method are not documented.

An example of a node selection configuration is $BPr_0 - Lf - N$. In this configuration best-projection backtracking node selection is used whenever a leaf node is reached, and the ANST method is not used. This is the default configuration in GLPK.

There are 79 possible configurations. Due to the impractical amount of time required to evaluate the performance of every configuration on a sufficiently large set of MIP instances, this experiment is carried out in two stages. Experiment 1 is used to reduce the list to a small subset of the most promising configurations. The set of MIP instances used in Experiment 1 allows the performance of all of the competitor configurations to be evaluated within a reasonable amount of time, i.e. these are the models that tend to be solved within a shorter time frame. Experiment 2 then evaluates the performance of the most promising configurations identified in Experiment 1 on a larger and more difficult set of MIP instances.

3.5 Test Models

The set of MIP instances used in the experiments includes a wide variety of MIP formulation characteristics. All 116 MIP instances from MIPLIB 2003 (MIPLIB 2003 [4], MIPLIB 3.0 [10], and MIPLIB 2.0 [9]) and all 156 MIP instances from CORAL [18] are used, for a total of 272 MIP instances. These are problems from industry, theoretical problems, or problems submitted to the NEOS server. These MIP instances are commonly used in testing branch and bound methods [2, 19].

4 Empirical Results

4.1 Experiment 1

This experiment tests all configurations of backtracking node selection, backtrack triggering, and active node search threshold methods on a small set of MIP instances to identify the most promising configurations for further testing in Experiment 2.

A solution time limit of 30 minutes is imposed for all methods. The 79 MIP instances that were solved within 30 minutes using the default GLPK node selection configuration $BPr_0 - Lf - N$ are used in this experiment. This biases the results to favor $BPr_0 - Lf - N$ by removing any instances that are not solved within the time limit using $BPr_0 - Lf - N$ even though they may be solvable within the time limit using a different configuration.

To determine how each configuration ranks compared to the others, we combine the three performance measures, TOTTM, MRATE and FAIL, into a single measure. Towards this end, the following definitions are used to rank each configuration.

TR: Time rank is the rank of a configuration according to TOTTM.

RR: Ratio rank is the rank of a configuration according to MRATE.

SR: Sum rank is the measure used to determine the overall rank of each configuration. $SR = FAIL + TR + RR$. Configurations with smaller values of SR are better.

R: The rank of each configuration according to SR.

Configurations are ranked according SR because it is desirable for a configuration to have the smallest value of $FAIL$, TR, and RR. Addition is used to prevent one of these measures from dominating the others. Table 4 shows that the largest difference either between TR and R, or between RR and R, is 11 for $BeF/BeE - PCA - N$ which has $R = 43$, $TR = 52$, and $RR = 32$. The average of this difference over all configurations is 3.2. $FAIL$ is added to SR instead of the ranking according to $FAIL$ because (i) $FAIL$ is integer valued, and (ii) if the $FAIL$ rank is added to SR, the difference between the values of $FAIL$ for two configurations is lost if this difference is greater than the difference in their ranks. In this experiment, the ranking of each configuration according to $FAIL$ is equal to $FAIL + 1$ for all but the worst 3 configurations, therefore either method of computing SR gives similar results.

This experiment required nearly 3 weeks of computation time to complete. For brevity, we summarize the results for just 12 of the 79 possible configurations: those that were chosen for testing in Experiment 2. These 12 configurations consist of the top seven in the combined ranking described above, and five state-of-the-art methods for comparison. The complete set of results is available online [26]. Further relevant analyses are also available in [25]. Tables 2 and 3 show the ranking of each configuration according to MRATE and TOTTM respectively.

Table 4 shows the rank measures for the 12 configurations. Standard node selection methods such as depth-first ($DeF - Lf - N$), best-first ($BeF - Lf - N$), and breadth-first ($BrF - Lf - N$) ranked 76, 64, and 71 respectively. FAIL is 0 for $BPr_0 - Lf - N$ because the MIP instances were selected based on the success of this method. The top 21 ranked configurations all involve at least one method out of those developed in this paper. The top ranked configuration, $BPr_1 - BPrA_1 - Y$, consists solely of methods developed here. Each of the new methods proposed in Sections 2.1 through 2.4 is used at least once in the top 7 configurations. For this reason, the top 7 ranked configurations are tested in Experiment 2. The best state of the art configuration, $BPr_0 - BPrA_0 - N$, is ranked 22 whereas the default GLPK configuration, $BPr_0 - Lf - N$, is ranked 39. These two configurations are also tested in Experiment 2 to provide a comparison to the best state of the art and to the baseline GLPK.

The rankings for configurations with a given backtracking node selection method tend to be closer to each other than those for a given backtrack triggering method, which indicates that the backtracking node selection method has a much greater influence on the performance of a given configuration. For example, the difference in ranks between the best and worst configurations with BPr_1 is 14. For $BPrA_1$, this difference is 77.

To determine the overall effect of using ANST, the arithmetic means of each of the performance measures ($\overline{TOTTM}$, $\overline{MRATE}$ and $\overline{FAIL}$) are computed over all configurations with ANST and compared to the corresponding means taken over all configurations that do not use ANST. All configurations with BrF and DeF are excluded from the means since neither of these methods is combined with ANST in this experiment. Table 5 shows that on average, ANST improves performance for all three of these measures. With respect to $\overline{FAIL}$, ANST improves performance by about 29%. With respect to $\overline{TOTTM}$, ANST improves performance by about 5%. With respect to $\overline{MRATE}$, ANST improves performance by about 1.7%.

Table 6 shows $\overline{TOTTM}$, $\overline{MRATE}$ and $\overline{FAIL}$ over all of the configurations grouped by backtrack triggering method. The performance of any given backtrack triggering method varies greatly depending on the choice of backtracking node selection method. For example, although $BPr_1 - BPrA_1 - Y$ is the best ranked configuration, Table 6 shows that, on average, $BPrA_1$ is inferior to $BPrA_0$. Similarly, $BPr_1 - DExA - Y$ ranks better than the highest ranked configuration with $BPrA_0$ but $DExA$ is inferior to $BPrA_0$ on average. Table 7 shows that the $DExA$ backtrack triggering method outperforms the state of the art backtrack triggering methods when either the BrF or BeF/BeE backtracking node selection method is used. It also shows that the $BPrA_1$ backtrack triggering method outperforms the state of the art backtrack triggering methods when either the BrF, $Dist$ or BPr_1 node selection method is used.

Table 2: Performance of each competitor configuration with respect to MRATE. The configurations that are available in the original GLPK solver are in boldface.

RR	Configuration	MRATE	RR	Configuration	MRATE
1	$Dist - E - N$	1.27133	27	$BPr_0 - BPrA_0 - N$	1.62144
2	$BPr_1 - BPrA_1 - Y$	1.31598	44	$\mathbf{BPr_0 - Lf - N}$	1.82184
3	$BPr_1 - BPrA_1 - N$	1.32779	65	$\mathbf{BeF - Lf - N}$	1.98177
4	$Dist - BPrA_1 - Y$	1.34052	71	$\mathbf{BrF - Lf - N}$	2.02911
5	$BPr_1 - BPrA_0 - Y$	1.34057	76	$\mathbf{DeF - Lf - N}$	2.69916
6	$BPr_1 - DExA - Y$	1.34315			
7	$BPr_1 - PCA - Y$	1.35208			

Table 3: Performance of each competitor configuration with respect to TOTTM. The configurations that are available in the original GLPK solver are in boldface.

TR	Configuration	TOTTM	TR	Configuration	TOTTM
1	$BPr_1 - PCA - Y$	16812.65	21	$BPr_0 - BPrA_0 - N$	19042.51
2	$BPr_1 - PCA - N$	16861.29	36	$\mathbf{BPr_0 - Lf - N}$	20584.63
3	$BPr_1 - BPrA_1 - Y$	17115.24	62	$\mathbf{BeF - Lf - N}$	24472.94
4	$BPr_1 - BPrA_1 - N$	17393.7	71	$\mathbf{BrF - Lf - N}$	25816.53
6	$Dist - BPrA_1 - Y$	17695.79	76	$\mathbf{DeF - Lf - N}$	30570.64
7	$Dist - E - N$	17776.13			
8	$BPr_1 - DExA - Y$	17906.25			

Table 4: Performance of each competitor configuration and their rankings. The configurations that are available in the original GLPK solver are in boldface. The smallest values of $FAIL$, TR, and RR are also in boldface.

R	Configuration	FAIL	TR	RR	R	Configuration	FAIL	TR	RR
1	$BPr_1 - BPrA_1 - Y$	1	3	2	22	$BPr_0 - BPrA_0 - N$	**0**	21	27
2	$BPr_1 - BPrA_1 - N$	1	4	3	39	$\mathbf{BPr_0 - Lf - N}$	**0**	36	44
3	$BPr_1 - PCA - Y$	1	**1**	7	64	$\mathbf{BeF - Lf - N}$	6	62	65
4	$Dist - E - N$	2	7	**1**	71	$\mathbf{BrF - Lf - N}$	6	71	71
5	$Dist - BPrA_1 - Y$	1	6	4	76	$\mathbf{DeF - Lf - N}$	7	76	76
5	$BPr_1 - PCA - N$	1	2	8					
7	$BPr_1 - DExA - Y$	2	8	6					

Table 5: Average performance of configurations with ANST. Configurations with BrF and DeF are excluded. The best values of each metric are in boldface.

	$\overline{FAIL}$	$\overline{TOTTM}$	$\overline{MRATE}$
Not using ANST	3.33	21684.53	1.72
ANST	**2.58**	**20574.42**	**1.69**

Table 6: Average performance of each aspiration method. $DeF - Lf - N$ is excluded. The best values of each metric are in boldface.

Aspiration	$\overline{FAIL}$	$\overline{TOTTM}$	$\overline{MRATE}$
$BPrA_0$	2.92	**21221.98**	**1.7**
$DExA$	3.23	21252.01	1.73
Lf	**2.85**	21432.44	1.76
PCA	3.69	22010.78	1.8
$BPrA_1$	3.69	22221.9	1.76
E	4.62	23705.7	1.94

Table 7: Rank of configurations with the same backtracking node selection option.

R	Configuration	R	Configuration	R	Configuration
1	$BPr_1 - BPrA_1 - Y$	4	$Dist - E - N$	9	$BeF/BeE - BPrA_1 - Y$
2	$BPr_1 - BPrA_1 - N$	5	$Dist - BPrA_1 - Y$	17	$BeF/BeE - DExA - Y$
3	$BPr_1 - PCA - Y$	15	$Dist - BPrA_1 - N$	23	$BeF/BeE - E - Y$
5	$BPr_1 - PCA - N$	17	$Dist - BPrA_0 - N$	24	$BeF/BeE - PCA - Y$
7	$BPr_1 - DExA - Y$	17	$Dist - Lf - Y$	25	$BeF/BeE - BPrA_0 - Y$
8	$BPr_1 - BPrA_0 - Y$	20	$Dist - BPrA_0 - Y$	25	$BeF/BeE - Lf - Y$
10	$BPr_1 - BPrA_0 - N$	21	$Dist - DExA - N$	33	$BeF/BeE - DExA - N$
10	$BPr_1 - Lf - N$	28	$Dist - DExA - Y$	34	$BeF/BeE - BPrA_1 - N$
12	$BPr_1 - DExA - N$	30	$Dist - Lf - N$	38	$BeF/BeE - E - N$
13	$BPr_1 - E - Y$	30	$Dist - PCA - N$	40	$BeF/BeE - Lf - N$
14	$BPr_1 - Lf - Y$	32	$Dist - PCA - Y$	43	$BeF/BeE - PCA - N$
15	$BPr_1 - E - N$	34	$Dist - E - Y$	44	$BeF/BeE - BPrA_0 - N$
22	$BPr_0 - BPrA_0 - N$	45	$BeE - E - Y$	55	$BeF - PCA - Y$
27	$BPr_0 - BPrA_0 - Y$	45	$BeE - BPrA_1 - Y$	56	$BeF - DExA - N$
29	$BPr_0 - E - Y$	45	$BeE - PCA - Y$	57	$BeF - DExA - Y$
36	$BPr_0 - Lf - Y$	49	$BeE - BPrA_0 - Y$	60	$BeF - BPrA_0 - Y$
37	$BPr_0 - DExA - N$	50	$BeE - Lf - Y$	60	$BeF - Lf - Y$
39	$BPr_0 - Lf - N$	51	$BeE - DExA - Y$	62	$BeF - BPrA_0 - N$
41	$BPr_0 - PCA - Y$	58	$BeE - BPrA_0 - N$	64	$BeF - Lf - N$
42	$BPr_0 - DExA - Y$	59	$BeE - BPrA_1 - N$	64	$BeF - PCA - N$
45	$BPr_0 - PCA - N$	63	$BeE - DExA - N$	66	$BeF - BPrA_1 - Y$
52	$BPr_0 - BPrA_1 - Y$	67	$BeE - PCA - N$	69	$BeF - BPrA_1 - N$
53	$BPr_0 - E - N$	68	$BeE - E - N$	74	$BeF - E - Y$
54	$BPr_0 - BPrA_1 - N$	70	$BeE - Lf - N$	75	$BeF - E - N$
71	$BrF - DExA - N$				
71	$BrF - Lf - N$				
73	$BrF - BPrA_0 - N$				
77	$BrF - PCA - N$				
78	$BrF - BPrA_1 - N$				
79	$BrF - E - N$				

4.2 Experiment 2

This experiment is a more thorough test to determine which of the most promising configurations identified in Experiment 1 is best. Both the time limit and the number of MIP instances are larger than those used in Experiment 1.

The configurations tested in this experiment are the top 7 ranked configurations from Experiment 1 as well as the top ranked state of the art and GLPK baseline configurations. Each of the new methods developed in this paper appears in at least one of the seven top ranked configurations. These configurations are listed in Table 8.

The entire set of MIP instances is used in this experiment, including the MIP instances from Experiment 1. The time limit for each MIP solution is 1 hour. This experiment required over 9 weeks of computation time to complete. The complete set of results is available online [27].

- 272 MIP instances are used in this experiment.
- 109 MIP instances were solved using at least 1 of the tested configurations.
- 130 MIP instances were not solved to optimality using any configuration, but a MIP feasible solution was found by at least one configuration.
- 33 MIP instances were not solved to optimality and no MIP feasible solution was found using any configuration.

Table 9 shows the performance of each tested configuration. All but one of the new configurations have fewer unsolved MIPs than the state of the art configurations. The configurations with the fewest unsolved MIPs are $Dist-BPrA_1-Y$, $BPr_1-DExA-Y$, $BPr_1-PCA-Y$ and BPr_1-BPrA_1-Y. These all use ANST whereas all of the others do not, showing that using ANST increases the chances of solving difficult MIPs within a stated time limit.

The TOTTM and MRATE for each of the new configurations is significantly better than for the state of the art configurations. The TOTTM for the worst new configuration, $BPr_1-PCA-N$, is 8% less than the best state of the art configuration, BPr_0-BPrA_0-N. In the case of the best new configuration, BPr_1-BPrA_1-Y, TOTTM is 26% less than BPr_0-BPrA_0-N. With respect to MRATE, the worst new configuration, $BPr_1-PCA-N$, is 17% better than the best state of the art configuration, BPr_0-BPrA_0-N. The best new configuration, BPr_1-BPrA_1-Y, has a value of MRATE that is 32% better than BPr_0-BPrA_0-N. The TOTTM for the second best new method, $Dist-BPrA_1-Y$, is 0.001% worse than BPr_1-BPrA_1-Y and the MRATE for $Dist-BPrA_1-Y$ is 3.2% worse than BPr_1-BPrA_1-Y which indicates that the performance of $Dist-BPrA_1-Y$ is very similar to BPr_1-BPrA_1-Y.

Figure 7 shows the performance profiles of the state of the art configurations: BPr_0-Lf-N and BPr_0-BPrA_0-N, and the two best new configurations with respect to TOTTM: BPr_1-BPrA_1-Y and $Dist-BPrA_1-Y$. This plot shows that for any given value of t_{MI}/t_{MIB}, the fraction of MIP instances with a ratio less than this value for BPr_1-BPrA_1-Y and $Dist-BPrA_1-Y$ is greater than that for BPr_0-Lf-Y and BPr_0-BPrA_0-Y. For example, t_{MI}/t_{MIB} for BPr_1-BPrA_1-Y is less than 2 for about 85% of the MIP instances; for $Dist-BPrA_1-Y$, this ratio is less than 2 for about 83% of MIP

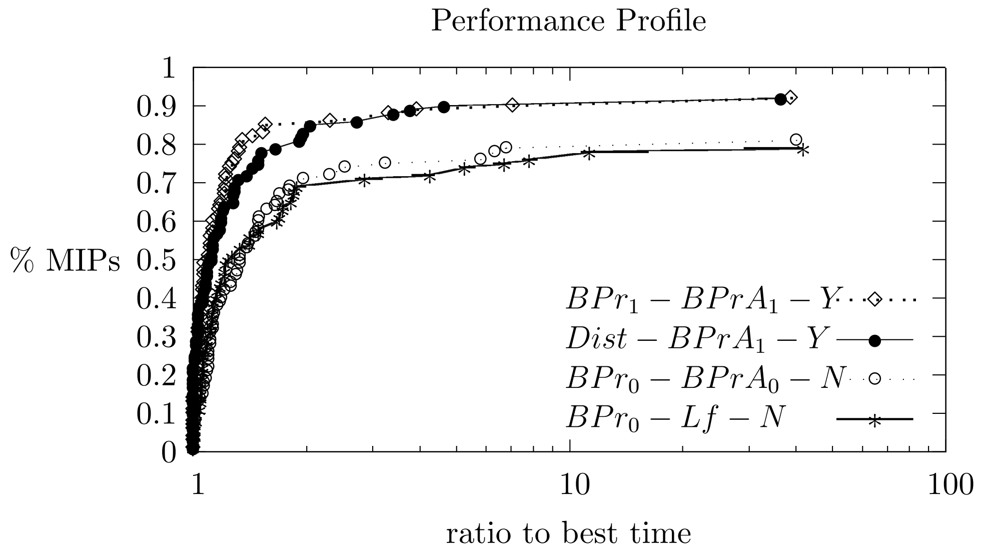

Figure 7: The performance profile comparing the two best new configurations with the two state of the art methods.

Table 8: Node selection configurations for Experiment 2. Rank R from Experiment 1 and reasons for inclusion are shown.

R	Configuration	Description
39	$BPr_0 - Lf - N$	Default GLPK: included as a baseline for GLPK
22	$BPr_0 - BPrA_0 - N$	Top ranked state of the art configuration
1	$BPr_1 - BPrA_1 - Y$	Top ranked proposed configuration
2	$BPr_1 - BPrA_1 - N$	Second ranked proposed configuration
3	$BPr_1 - PCA - Y$	Third ranked proposed configuration
4	$Dist - E - N$	Fourth ranked proposed configuration
5	$Dist - BPrA_1 - Y$	Fifth ranked proposed configuration
5	$BPr_1 - PCA - N$	Fifth ranked proposed configuration
7	$BPr_1 - DExA - Y$	Highest ranked proposed configuration with $DExA$

Table 9: Performance Experiment 2 configurations. The smallest values of each metric are shown in boldface.

	Solved MIPs			Unsolved MIPs		
Configuration	FAIL	TOTTM	MRATE	AVGRANK	NFIRSTS	NINC
$BPr_1 - BPrA_1 - Y$	**6**	**86653.72**	**1.33**	4.54	8	75
$Dist - BPrA_1 - Y$	**6**	86654.38	1.37	2.89	34	21
$BPr_1 - PCA - Y$	**6**	88560.12	1.34	4.34	11	72
$BPr_1 - DExA - Y$	**6**	91509.11	1.4	4.18	17	57
$Dist - E - N$	15	97340.26	1.34	**2.48**	**67**	17
$BPr_1 - BPrA_1 - N$	13	99798.67	1.43	4.62	7	81
$BPr_1 - PCA - N$	12	100752.39	1.5	4.6	9	82
$BPr_0 - BPrA_0 - N$	14	109239.58	1.75	3.74	21	**15**
$BPr_0 - Lf - N$	15	109579.06	1.78	3.47	25	**15**

instances; for $BPr_0 - BPrA_0 - N$, this ratio is less than 2 for about 71% of MIP instances; for $BPr_0 - Lf - N$, this ratio is less than 2 for about 69% of MIP instances. Over all of the instances not solved within 10 seconds for every contending method $BPr_1 - BPrA_1 - Y$ performed best for solving 11% of these instances, $Dist - BPrA_1 - Y$ performed best for solving 22% of these instances, $BPr_0 - BPrA_0 - N$ performed best for solving 7% of these instances, and $BPr_0 - Lf - N$ performed best for solving 0% of these instances. Over all of the MIP instances not solved within 10 seconds for every contending method: $BPr_1 - BPrA_1 - Y$ and $Dist - BPrA_1 - Y$ solved 92% of these instances, $BPr_0 - BPrA_0 - N$ solved 81% of these instances, and $BPr_0 - Lf - N$ solved 79% of these instances.

130 MIP instances were not solved to optimality within 1 hour using any tested configuration, but a feasible solution was found by at least one configuration. With respect to AVGRANK and NFIRSTS, $Dist - E - N$ is the best configuration, the second best is $Dist - BPrA_1 - Y$. These results suggest that the $Dist$ backtracking node selection method performs better than the other methods for either finding good MIP feasible solutions to the more difficult MIP instances or reducing the optimality gap of these MIP instances.

The best configurations with respect to NINC are $BPr_0 - BPrA_0 - N$ and $BPr_0 - Lf - N$ because both use the most-feasible backtracking node selection method without an aspiration value until an incumbent solution is found. The goal of the most-feasible backtracking node selection method is to quickly find a MIP feasible solution.

The results of this experiment suggest that $Dist - BPrA_1 - Y$ is the best configuration for solving MIPs. It has the smallest number of unsolved MIPs (the same as 3 other configurations) and it outperforms the state of the art configurations with respect to all but one of the metrics FAIL, TOTTM, MRATE, AVGRANK, and NFIRSTS. $Dist - BPrA_1 - Y$ is only marginally worse than the only configuration, $BPr_1 - BPrA_1 - Y$, that performs better with respect to TOTTM. Similarly, $Dist - BPrA_1 - Y$ is marginally worse than the best two configurations, $BPr_1 - BPrA_1 - Y$ and $BPr_1 - PCA - Y$ with respect to MRATE. Finally, $Dist - BPrA_1 - Y$ performs significantly better than both $BPr_1 - BPrA_1 - Y$ and $BPr_1 - PCA - Y$ with respect to AVGRANK and NFIRSTS.

5 Conclusions

The backtracking node selection, backtrack triggering, and ANST methods developed in this paper reduce the amount of effort required to solve MIPs to optimality when compared to the current state of the art methods.

The best configuration for solving MIPs quickly is modified best-projection backtracking node selection with modified best-projection backtrack triggering, and with ANST. Distribution backtracking node selection with modified best-projection backtrack triggering and ANST solves MIPs nearly as quickly and performs better than the current state of the art with respect to finding good incumbent solutions and closing the optimality gap for the very difficult MIP instances that were not solved within the time limit.

We conclude that the methods developed in this paper show significant promise for solving MIPs quickly. Further research may bring even further improvements.

5.1 Future Research

There are a number of ideas that may improve the methods developed in this paper:

- The modified best-projection method may be improved by adaptively selecting the points (z_i, c_i) used to estimate the degradation in objective value per unit MIP infeasibility. The change in the gradient of $z^{min}(c)$ may be useful towards this end.

- The distribution method may be improved by using distribution functions that are better suited for modeling the objective values and the numbers of candidate variables of nodes in the tree. These distribution functions may account for the correlation of these values. Other node characteristics, such as depth, may also be used.

- The distribution method may be improved by combining correlation information on the paths from the root node to each active node with global information such as the averages and variances of the objective values, numbers of candidate variables, and depths of nodes.

- The parameters of the distribution functions used in the distribution method are computed using the assumption that the objective value of a node is independent of the objective value of any other node. This is not true for nodes that are ancestors or descendants of each other so it may be better to use only nodes that are not ancestors or descendants of each other in the computation of the parameters of the distribution functions.

- The ANST threshold value may be improved if it is computed for each MIP instance based on some features of that instance and its branch and bound tree. Features that may be useful are the number of variables, the number of constraints, the number of active nodes, the optimality gap, the number of times backtracking has been performed, the number of MIP feasible nodes, and the number of LP infeasible nodes. It may be possible to set the threshold value adaptively as the branch and bound method proceeds based on these features.

- The modified best-projection aspiration algorithm updates the aspiration value by searching through all of the active nodes after each node has been solved. If the number of active nodes becomes large then it may be better to reduce the frequency at which the aspiration value is updated.

- The feasibility depth extrapolation aspiration algorithm estimates the depth of an optimal node as the smallest extrapolated depth of a MIP feasible node over all solved nodes. This estimate may become much too small without any way of increasing it. It may be better to increase this estimate using information such as the depths of all active nodes, or to use a different method of choosing the MIP optimal depth from all of the extrapolated depths of MIP feasible nodes.

In Section 2.4 two methods of storing the active nodes in a branch and bound tree are discussed: the last active node created is placed at the end of the list; or each node is placed in the list by priority according to the node selection criteria. An empirical study comparing these methods has not been found in the literature. Such a comparative study should be conducted to determine which method is better.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

[2] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. http://opus.kobv.de/tuberlin/volltexte/2007/1611/.

[3] Tobias Achterberg. Scip 1.0.0. http://scip.zib.de/, 2007.

[4] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. See http://miplib.zib.de.

[5] A. Atamtürk and M. Savelsbergh. Integer-programming software systems. Research Report BCOL.03.01, IEOR, University of California at Berkeley, 2005.

[6] M. Bénichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[7] Timo Berthold. *Primal Heuristics for Mixed Integer Programs*. PhD thesis, Fachbereich Mathematik der Technische Universität Berlin, 2006.

[8] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mip: Theory and practice – closing the gap. In M. Powell and S. Scholtes, editors, *Systems Modelling and Optimization: Methods, Theory, and Applications*, pages 19–49. Kluwer Academic Publisher, 2000.

[9] Robert E. Bixby, Sebastian Ceria, Cassandra M. McZeal, and Martin W.P. Savelsbergh. Miplib 2.0 - mixed integer problem library. http://miplib.zib.de/miplib3/miplib_prev.html, 1996.

[10] Robert E. Bixby, Sebastian Ceria, Cassandra M. McZeal, and Martin W.P. Savelsbergh. Miplib 3.0 - mixed integer problem library. http://miplib.zib.de/miplib3/miplib.html, 1996.

[11] R. J. Dakin. A tree search algorithm for mixed programming problems. *Computer Journal*, 8:250–255, 1965.

[12] Elizabeth D. Dolan and Jorge J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201 – 213, 2002.

[13] N. J. Driebeek. An algorithm for the solution of mixed-integer programming problems. *Management Science*, 12:576–587, 1966.

[14] J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin. Practical solution of large scale mixed integer programming problems with umpire. *Management Science*, 20:736–773, 1974.

[15] J. M. Gauthier and G. Ribière. Experiments in mixed-integer linear programming using pseudocosts. *Mathematical Programming*, 12:26–47, 1977.

[16] J. P. H. Hirst. Features required in branch and bound algorithms for (0-1) mixed integer linear programming. Privately circulated manuscript, December 1969.

[17] Konstantinos Kostikas and Charalambos Fragakis. Genetic programming applied to mixed integer programming. In *Genetic Programming, 7th European Conference, EuroGP2004, Proceedings*, volume 3003 of *Lecture Notes in Computer Science*. Springer, 2004.

[18] J. Linderoth. Coral - mixed integer programming instances. http://coral.ie.lehigh.edu/mip-instances/, 2006.

[19] J. T. Linderoth and T. K. Ralphs. Noncommercial software for mixed-integer linear programming. Technical report, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, 2004. http://www.lehigh.edu/ tkr2/research/papers/MILP04.pdf.

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

[21] A. Makhorin. Glpk 4.9. http://www.gnu.org/software/glpk/glpk.html, 2006.

[22] A. Martin. Integer programs with block structure. Technical report, Habilitations-Schrift, Technische Universität Berlin, 1998. http://www.zib.de/Publications/abstracts/SC-99-03/.

[23] G. Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155–170, 1973.

[24] J. Patel and J. W. Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming Series A*, 110:445 – 474, 2007.

[25] D.T. Wojtaszek. *Faster MIP Solutions via New Node Selection Rules.* PhD thesis, Carleton University, 2008.

[26] D.T. Wojtaszek. Faster mip solutions via new node selection rules: Experiment 1 data. www.sce.carleton.ca/faculty/chinneck/students/Wojtaszek/data-exp-1.pdf, 2009.

[27] D.T. Wojtaszek. Faster mip solutions via new node selection rules: Experiment 2 data. www.sce.carleton.ca/faculty/chinneck/students/Wojtaszek/data-exp-2.pdf, 2009.