A Computational Study of Search Strategies for Mixed Integer Programming

J. T. Linderoth
M.W.P. Savelsbergh

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205, U.S.A.

Abstract

The branch and bound procedure for solving mixed integer programming (MIP) problems using linear programming relaxations has been used with great success for decades. Over the years, a variety of researchers have studied ways of making the basic algorithm more effective. Breakthroughs in the fields of computer hardware, computer software, and mathematics have led to increasing success at solving larger and larger MIP instances.

The goal of this paper is to survey many of the results regarding branch and bound search strategies and evaluate them again in light of the other advances that have taken place over the years. In addition, novel search strategies are presented and shown to often perform better than those currently used in practice.

October 1997

The effectiveness of the branch and bound procedure for solving mixed integer programming (MIP) problems using linear programming relaxations is well documented. After the introduction of this procedure in the 1960's [26] [10] [4], researchers in the 1970's examined strategies for searching the branch and bound tree in an efficient manner [5] [16] [28] [7] [14] [15]. With a few exceptions, (notably [30] and [19]) research in past decades has strayed from developing and examining effective search techniques and instead focused on improving the bound obtained by the linear programming relaxation [9] [33] [34]. The goal of this paper is to reexamine search techniques in light of the many advances made in the field over the years.

One major change over the past decades is in the hardware on which MIPs are solved. Computers of today are orders of magnitudes faster than their counterparts of yesteryear, allowing us to expend more computational effort in the solution of a MIP instance. Furthermore, computers of today are equipped with a great deal more memory than in the past, so a search strategy designed to limit memory size may not be of much use today.

A second change since the 1970's is in the theory behind solving MIPs. In the 1980's, the usefulness of combining cutting planes and branch and bound was demonstrated. All industrial strength MIP codes now contain some sort of cut generation in addition to branch and bound. Dramatic improvements in the simplex algorithm have also been made which allow for fast reoptimization of an LP, regardless of the change

in LP-relaxation from one node to the next. Also, most MIP codes now use a heuristic method to attempt to find feasible solutions. Each of these practical and theoretical improvements impacts the effectiveness of a particular strategy for searching the branch and bound tree.

What advances in the future will allow researchers to solve larger MIP instances? Undoubtedly, theoretical advancements will continue to have a major impact. Another major impact will come from implementing the current branch and bound algorithms on parallel computers. In a wide range of fields, the introduction of parallel computers consisting of many microprocessors has made possible the solutions of problems impossible to consider solving on a single processor. Many researchers have studied the effects, both good and bad, of dividing the search space for an optimization problem among many processors [24] [23] [37]. Key to achieving an effective algorithm on parallel computers is an effective means for exploring the search space. Further, in a parallel environment, information found on one processor may be useful in guiding the search on a different processor. Sharing information on a distributed memory architecture requires that a message be passed, for which some overhead is incurred. It therefore is useful to know what types of information are especially useful for guiding the search and how much of this information must be gathered and shared. This is yet another motivation for restudying search strategies for MIP.

The purpose of this paper is two-fold – to survey existing ideas for searching the branch and bound tree and to offer some new techniques for performing the search. Extensive computational experiments will be presented comparing the old and new methods. We begin by briefly reviewing the branch and bound algorithm for solving a mixed integer program using linear programming relaxations. In Section 2, we discuss and compare various "branching rules", or methods of subdividing the search space. In Section 3 different "node selection schemes", or search-ordering methods, are presented and compared. Finally, in Section 4 we draw some conclusions from our work and give some directions for future research.

1 LP-Based Branch and Bound.

A mixed integer program (MIP) can be stated mathematically as follows:

$$
\begin{aligned}
\text{Maximize} \quad & z_{MIP} = \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \\
\text{subject to} \quad & \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \ \leq \ b_i \quad i = 1, \ldots m \\
& l_j \leq x_j \leq u_j \qquad j \in N \\
& x_j \in \mathcal{Z} \qquad j \in I \\
& x_j \in \Re \qquad j \in C,
\end{aligned}
$$

where I is the set of integer variables, C is the set of continuous variables, and $N = I \cup C$. The lower and upper bounds l_j and u_j may take on the values of plus or minus infinity.

The term "branch and bound" was originally coined by Little *et al.* [27] in their study of such an algorithm to solve the traveling salesman problem. However, the idea of

using a branch and bound algorithm for integer programming using linear programming relaxations was proposed somewhat earlier by Land and Doig [26]. The process involves keeping a list of linear programming problems obtained by relaxing some or all of the integer requirements on the variables x_j, $j \in I$. To precisely define the algorithm, let us make a few definitions. We use the term *node* or *subproblem* to denote the problem associated with a certain portion of the feasible region of MIP. Define z_L to be a lower bound on the value of z_{MIP}. For a node N^i, let z_U^i be an upper bound on the value that z_{MIP} can have in N^i. The list $\mathcal{L}$ of problems that must still be solved is called the *active set*. Denote the optimal solution by x^*. Algorithm 1 is an LP-based branch and bound algorithm for solving MIP.

Algorithm 1 The Branch and Bound Algorithm

0. **Initialize.**
   $\mathcal{L} = \text{MIP}$. $z_L = -\infty$. $x^* = \emptyset$.
1. **Terminate?**
   Is $\mathcal{L} = \emptyset$? If so, the solution x^* is optimal.
2. **Select.**
   Choose and delete a problem N^i from $\mathcal{L}$.
3. **Evaluate.**
   Solve the LP relaxation of N^i. If the problem is infeasible, go to step 1, else let z_{LP}^i be its objective function value and x^i be its solution.
4. **Prune.**
   If $z_{LP}^i \leq z_L$, go to step 1. If x^i is fractional, go to step 5, else let $z_L = z_{LP}^i$, $x^* = x^i$, and delete from $\mathcal{L}$ all problems with $z_U^j \leq z_L$. Go to step 1.
5. **Divide.**
   Divide the feasible region of N^i into a number of smaller feasible regions $N^{i1}, N^{i2}, \ldots, N^{ik}$ such that $\cup_{j=1}^k N^{ij} = N^i$. For each $j = 1, 2, \ldots, k$, let $z_U^{ij} = z_{LP}^i$ and add the problem N^{ij} to $\mathcal{L}$. Go to 1.

The description makes it clear that there are various choices to be made during the course of the algorithm. Namely, how do we select which subproblem to evaluate, and how do we divide the feasible region. A partial answer to these two questions will be provided in the next two sections. When answering these questions, our main focus will be to build *robust* strategies that work well on a wide variety of problems.

2 Problem Division

Since our algorithm is based on linear programming relaxations, we need to enforce linear constraints to divide the current feasible region as required in step (5) of Algorithm 1. However, adding constraints to the linear programming formulation leads to the undesirable result of increasing the size of the linear program that must be solved as we progress. A more efficient way to divide the region is to change the bounds on variables. In the following sections, we present the two most common (and obvious) schemes for partitioning the feasible region by changing variables' bounds.

2.1 Variable Dichotomy

For MIP, the obvious way to divide the feasible region of N^i is to choose a variable j that has fractional value x_j^i in the solution to the LP relaxation of N^i and impose the new bounds of $x_j \leq \lfloor x_j^i \rfloor$ to define one subregion and $x_j \geq \lceil x_j^i \rceil$ to define another subregion. We call this branching on a *variable dichotomy*, or simply branching on a variable.

If there are many fractional variables in the solution to the LP relaxation of N^i, we must choose one variable to define the division. Because the effectiveness of the branch and bound method strongly depends on how quickly the upper and lower bounds converge, we would like to branch on a variable that will improve these bounds. It seems difficult to select a branching variable that will affect the lower bound. Doing this amounts to heuristically finding feasible solutions to MIP and is very problem specific. However, there are ways to attempt to predict which fractional variables will most improve the upper bound when required to be integral. These prediction methods fall into two general categories. The first category includes methods that attempt to estimate the change (or degradation) of the objective function value and the second category includes those that provide a lower bound on the degradation of the objective function value.

2.1.1 Estimation Methods

Estimation methods work as follows: with each integer variable x_j, we associate two quantities P_j^- and P_j^+ that attempt to measure the per unit decrease in objective function value if we fix x_j to its rounded down value and rounded up value, respectively. Suppose that $x_j^i = \lfloor x_j^i \rfloor + f_j^i$, with $f_j^i > 0$. Then by branching on x_j, we will estimate a decrease of $D_j^{i-} = P_j^- f_j^i$ on the down branch of node i and a decrease of $D_j^{i+} = P_j^+(1 - f_j^i)$ on the up branch of node i. Bénichou *et al.* [5] call the values P_j^- and P_j^+ *down* and *up pseudocosts*.

One way to obtain values for P_j^- and P_j^+ is to simply use the observed degradation in objective function value. Let N^{i-} and N^{i+} denote the nodes for the down and up branches of node N^i, then compute the pseudocosts as

$$P_j^- = \frac{z_{LP}^{i-} - z_{LP}^i}{f_j^i} \qquad \text{and} \qquad P_j^+ = \frac{z_{LP}^{i+} - z_{LP}^i}{1 - f_j^i}.$$

Two more questions need to be answered before we can implement a branching scheme based on pseudocosts:

- To what value should the pseudocosts be initialized?
- How should the pseudocosts be updated from one branch to the next?

The question of initialization is an important one. By the very nature of the branch and bound process, the branching decisions made at the top of the tree are the most crucial. As pointed out by Forrest *et al.* [14], if at the root node we branch on a variable that has little or no effect on the LP solution at subsequent nodes, we have essentially doubled the total amount of work required.

An obvious method for initialization is to simply let the pseudocosts for a variable be the value of its objective function coefficient, since if a variable were unrelated to the

other variables in the problem, its pseudocost would be precisely its objective function coefficient.

Bénichou *et al.* [5] and Gauthier and Ribière [15] experimented with explicitly computing the pseudocosts of each variable. They report that doing so can be effective in reducing the number of nodes evaluated, but that the time spent computing these values explicitly is significant. In fact, Gauthier and Ribière conclude that the computational effort is too significant compared to the benefit obtained.

Even though today's situation may be slightly different, due to faster computers and better LP solvers, it clearly indicates that care has to be taken when pseudocosts are explicitly computed for each variable. In doing our experiments, we noted that often only a small percentage of the integer variables are ever fractional in the solution to the linear programming relaxation and an even smaller percentage of integer variables are ever branched on. Therefore, if pseudocosts are going to be explicitly computed, then they should be computed only for the fractional variables as needed.

Yet another alternative, suggested by Eckstein [13], keeps track of the average value of the pseudocost on the up and down branches. For each variable that has yet to be arbitrated, the pseudocosts are set to these average values. This method has the disadvantage that variables not yet branched on are ranked in importance only by how fractional they are.

In the course of the solution process, the variable x_j may be branched on many times. How should the pseudocosts be updated from one branch to the next? Bénichou *et al.* [5] state that the pseudocosts vary little throughout the branch and bound tree, and suggest fixing P_j^- and P_j^+ to the values observed the first time when variable x_j is branched on. Alternatively, as suggested in Forrest *et al.* [14], one could also fix the pseudocosts to the values obtained from the last time when x_j was branched on. Forrest *et al.* [14] and Eckstein [13] suggest averaging the values from all x_j branches.

We performed an experiment to verify the observations of Bénichou *et al.* [5], i.e., that the pseudocosts are relatively constant throughout the branch and bound tree. Suppose the (either up or down) pseudocost P_j is some linear function of the number of times N_j variable x_j is branched on. We can express this relationship as

$$P_j = \beta_0 + \beta_1 N_j.$$

For our suite of 14 problems from MIPLIB, we explicitly computed the regression coefficients β_0 and β_1 for each variable and direction on which we chose to branch more than seven times. This gave us 693 variable-direction pairs. For these 693, zero was in the 95% confidence interval of the regression coefficient β_1 673 times, which would imply there was statistical reason to believe that the pseudocosts are constant throughout the branch and bound tree for these variable-direction pairs. However, we also observed that from node to node, pseudocosts can vary significantly. In Figure 2.1, we plot the observed pseudocosts as a function of the number of times we branch on a specific variable. Therefore, we believe that updating the pseudocosts by averaging the observations should be the most effective.

The issues of initialization of pseudocosts and updating of pseudocosts are unrelated. Generally once a variable is branched on the initial pseudocost value is discarded and

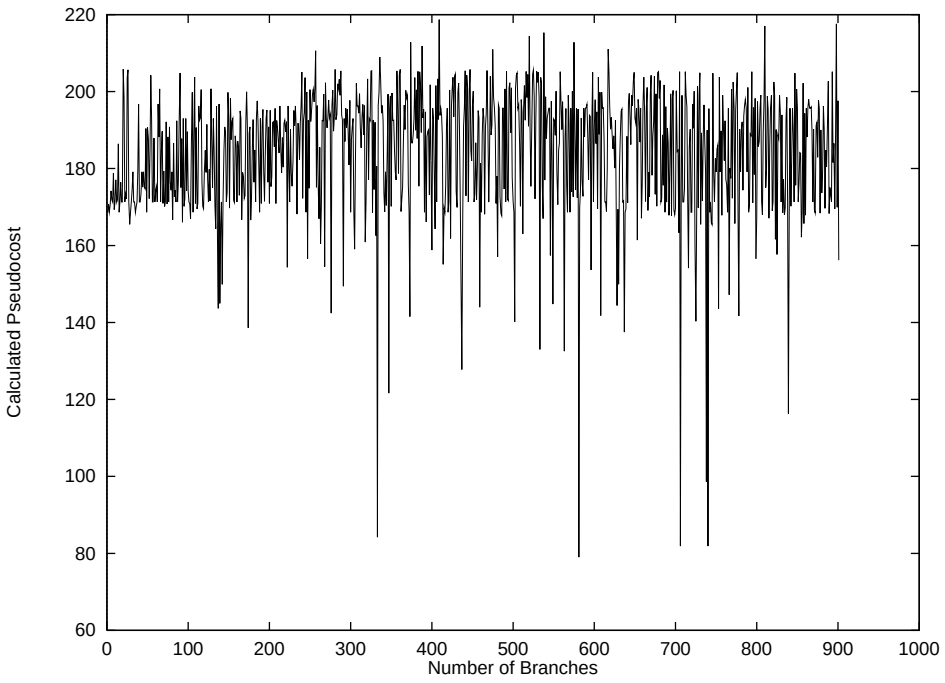

Figure 2.1: Observed Pseudocosts as a Function of Number of Branches. Problem **pp08a**. Variable 219.

replaced by the true (observed) pseudocost. Therefore, we will deal with the initialization and update issues separately.

Throughout the course of the paper, we will be introducing experiments aimed at establishing the effectiveness of different search strategy techniques for mixed integer programming. The techniques under investigation were incorporated into the mixed integer optimizer MINTO (v2.0) [29]. Since one of the main focuses of the paper is to establish how search strategies interact with new, sophisticated, integer programming techniques, we have used the advanced features MINTO has to offer in performing the experiments. These features include preprocessing and probing, automatic cut generation, reduced cost fixing, and row management. We have chosen to perform a majority of the testing on a suite of 14 problems from the newest version of MIPLIB [6]. The instances were chosen more or less at random, but exhibit a wide range of problem characteristics. Unless otherwise noted, all experiments were run with the settings shown in Table 2.1. Other characteristics about the experiments will be mentioned as needed.

- Code compiled with the IBM XLC compiler, optimization level -O2.
- Code run on an RS/6000 Model 590.
- CPU time limited to one hour.
- Memory limited to 100mb.

Table 2.1: Characteristics of All Computational Experiments

We now describe an experiment that aims at establishing the best pseudocost ini-

tialization method. Since determining the best initialization method is our goal here, we have fixed the updating method in these runs to be the averaging suggestion. We branch on the variable x_j for which $D_j^{i-} + D_j^{i+}$ is the largest. This choice will be discussed in more detail in Section 2.1.4. As previously stated, the main focus of choosing a branching variable is to choose one that will most improve the upper bound of the child nodes from the parent. By setting z_L to the value of the optimal solution to the problem in our computational experiments, we minimize factors other than branching that determine the size of the branch and bound tree. For example, the node selection rule has no effect on the size of the branch and bound tree. Just for completeness, we mention that we use the "best bound" node selection rule, where at the **Select** portion of the branch and bound algorithm, we choose to evaluate the active node with the largest value of z_U^i. In our tables of computational results, the "Final Gap" is computed as

$$\text{Final Gap} \equiv \frac{\max_{i \in \mathcal{L}} z_U^i - z_L}{z_L},$$

and a "XXX" in the solution time column signifies that the memory limit was reached.

For many experiments, we will be including summary tables that rank the performance of the techniques under investigation. We rank techniques related to branching methods as follows:

- A method that proves the optimality of the solution is ranked higher than one that does not.
- If two methods prove the optimality of the solution, the one with shorter computation time is ranked higher.
- If two methods do not prove the optimality of the solution, the one with smaller final gap is ranked higher.
- Ties are allowed.

Table A.1 in Appendix A shows the results of solving various problems using four pseudocost initialization methods. A summary of the experiment is given in Table 2.2. The pseudocosts were initialized with objective function coefficients, by averaging observations, by explicitly computing them for all variables at the root node, and explicitly computing them for the fractional variables only as needed.

Initalization Method	Avg. Ranking
Obj. Coef.	2.93
Averaged	3.07
Computed All	2.50
Computed Fractional	1.50

Table 2.2: Summary of Pseudocost Initialization Experiment.

Examination of the results shows that explicitly computing initial pseudocosts for fractional variables as needed is clearly the best method. This result is different than

the conclusion reached by Gauthier and Ribière [15]. The faster simplex algorithm and computers of today now make it possible to invest more effort into the (often very important) initial branching decisions.

We conclude that a good pseudocost initialization strategy should allow for initially explicitly computing pseudocosts, take care not to expend too much computation time accomplishing this task, and allow for spending more time computing explicit pseudocosts at the top of the branch and bound tree where branching decisions are more crucial. After further experimentation, we adopted the following pseudocost initialization strategy. Let T be the maximum amount of time per node we would like to spend to initialize pseudocosts for variables on which we have yet to branch. In this time, we wish to gain useful branching information on all fractional variables. We therefore impose a limit L on the number of simplex iterations used in solving the linear program necessary to compute one particular pseudocost. Let γ be an estimate of the number of simplex iterations performed per unit time, obtained by

$$\gamma \equiv \frac{\text{Number of iterations needed to solve the initial LP}}{\text{Time to solve the initial LP}}.$$

Let η be the number of fractional variables in initial LP solution. Then we compute L as

$$L = \frac{T\gamma}{2\eta}.$$

As we develop the branch and bound tree, if there is a fractional variable x_j upon which we have never branched, we perform L simplex pivots after fixing the bounds of this variable to $\lfloor x_j \rfloor$ and $\lceil x_j \rceil$ in order to explicitly determine the pseudocosts.

Gauthier and Ribière [15] also proposed a pseudocost initialization strategy that uses a limited number of simplex iterations, but our approach is fundamentally different. They purposely limit the number of simplex iterations to a small number, while we set T to a large number hoping to be able to compute a "true" pseudocost. T is two minutes in the current implementation.

We now turn our attention to the question of how to update the pseudocosts from one branch to the next. As mentioned above, our initial experiments lead us to believe that updating the pseudocosts by averaging the observations would be the most computationally effective.

We empirically verified this conjecture by solving instances of MIPLIB where the pseudocosts were updated in each of the three ways suggested. For these runs we have initialized the pseudocosts by our strategy that explicitly computes them, with a limit on the number of iterations used. As in our previous experiment, we branch on the variable x_j for which $D_j^{i-} + D_j^{i+}$ is the largest, set z_L to be the known optimal solution to the problem, and we use the best bound node selection rule. Table A.2 shows the full results of this experiment.

For the pseudocost update experiment, Table 2.3 shows the average ranking over all the instances of each method.

From the results of the experiment we see that our intuition is correct. For the most part, it seems to be best to average the degradations when branching on a variable to determine its pseudocosts, and the overhead necessary to perform the averaging does not outweigh its benefits.

Update Method	Avg. Ranking
First	2.43
Last	1.64
Average	1.43

Table 2.3: Summary of Pseudocost Update Experiment.

In distributed memory parallel computer architectures, the overhead necessary to perform true averaging of pseudocosts increases dramatically, since different processors calculate different nodes (and hence different pseudocosts). The fact that the "Last" updating technique is not significantly outperformed by the "Average" updating technique leads us to believe that true averaging may not be necessary for computational effectiveness on parallel architectures.

For the remainder of this paper, when we refer to *pseudocost branching*, this will imply that we have used our strategy of explicitly computing the initial pseudocosts with a simplex iteration limit, and we update the pseudocosts by averaging the observations.

2.1.2 Lower Bounding Methods

Branching strategies that provide a lower bound on the objective function degradation work much the same way as estimation strategies. For each variable, we find quantities L_j^- and L_j^+ that provide lower bounds on the decrease in objective function value if we branch down and up on variable x_j. This idea originated with the work of Dakin [10], Healy [21], and Davis *et al.* [11]. Driebeek [12] shows how to compute values for L_j^- and L_j^+ by implicitly performing one dual simplex pivot. Breu and Burdet provide computational evidence that lower bounding methods can be beneficial [7], however branching methods based on simple lower bound calculations have fallen out of favor in recent years [2] [14]. As Beale states in [2], "the fact remains that practical problems tend to have several nonbasic variables with zero reduced costs, when these methods are largely useless."

Over the years since Beale made this statement, much work has been done in the area of generating strong valid inequalities for MIP and incorporating these inequalities into a branch and cut scheme [9] [17] [31] [33]. With the advent of these sophisticated cutting planes, one may suspect to have fewer nonbasic variables with zero reduced cost. The rationale for this statement is as follows. Non-basic variables with zero reduced cost correspond to alternative optimal solutions to the linear programming relaxation. Cutting planes meant to separate a fractional LP solution from the convex hull of integer points may also separate alternative LP-optimal solutions. Figure 2.2 graphically depicts this phenomenon in two dimensions. We also empirically verified this observation by determining the percentage of nodes where a non-zero lower bound on the degradation is found when cuts are added or not added to the formulation. Table A.3 shows the percentage of the first 250 nodes of the branch and bound tree that have useful dual estimates when both cuts are added and not added to the formulation. This experiment was run on all the problems of MIPLIB, but only the instances where the difference in percentage is greater than 4% are reported. For this experiment, we used the lower

bound obtained from implicitly performing one dual simplex pivot. The results indicate that somewhat more non-zero estimates are obtained with cutting planes, but there are no dramatic improvements.

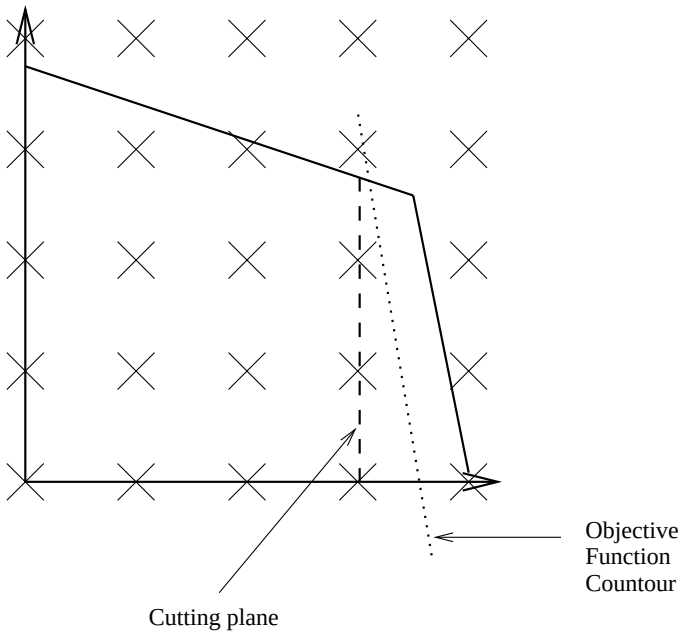

Figure 2.2: A Valid Inequality That Cuts Off Alternative Optimal Solutions

One may make the argument that since rows are being added to the linear programming formulation, it is likely that a greater number of dual simplex pivots are required for the child node to satisfy the bound we have imposed. Hence, the lower bound obtained by implicitly performing one dual simplex pivot bears less relation to the true degradation. However, we will show that the lower bounds obtained in this fashion can be useful in determining a branching variable.

Instead of implicitly performing one dual simplex pivot, a number of actual dual simplex pivots can be performed for each variable. The change in objective function again provides a lower bound on the true degradation. A strategy similar to this is what Applegate *et al.* [1] call *strong branching*. Strong branching selects a set of "good" variables on which one may choose to branch and performs a number of dual simplex pivots on each variable in this set. Strong branching has been shown to be an effective branching rule for large set partitioning problems, traveling salesman problems, and some general integer programs [1] [8]. The idea of performing multiple dual simplex pivots is closely related to the pseudocost initialization idea we propose. However, there are two main differences. For pseudocost initialization, the dual simplex pivots are performed only once for each variable, regardless of how many times the variable was branched on. Further, for pseudocost initialization, a large number of pivots are performed in the hope of completely resolving the problem.

Tomlin [36] has shown that Driebeek's lower bounding method, i.e., implicitly performing one dual simplex pivot, can be improved by taking into account the integrality

of the nonbasic variables. In that case, the quantities L_j^- and L_j^+ no longer provide lower bounds on the degradation of the objective function with respect to the optimal value of the LP relaxation at the child node, but on the degradation of the objective function with respect to the optimal value of the IP at the child node. Tomlin has also observed that these lower bounds are dominated by bounds derived from Gomory cuts.

Recently, Günlük [18] has proposed another extension of the Driebeek's lower bounding, which he calls *knapsack branching*. Knapsack branching is similar to the penalty improvement idea of Tomlin [36], since both methods take into account the integrality of the non-basic variables. However, knapsack branching takes the integrality into account in a more involved way. In order to determine the value L_j^- or L_j^+, a knapsack problem must be solved. Günlük also suggests a method for solving the knapsack problems efficiently.

We solved some MIPLIB instances using the various lower bounding methods in order to compare performance. The lower bounding branching methods we chose to compare were to perform one dual simplex pivot on all fractional variables, 10 dual simplex pivots on all fractional variables, 25 dual simplex pivots on a subset of the fractional variables (i.e. strong branching), and knapsack branching. To determine the subset $I' \subseteq I$ of variables on which to perform 25 dual simplex pivots, we used the following strategy. Given a fractional LP solution, let $L = \max\{f_j : f_j \leq 0.5, j \in I\}$ and $U = \min\{f_j : f_j \geq 0.5, j \in I\}$. We chose

$$I' \equiv \{j \in I : 0.8L \leq f_j \leq U + 0.2(1 - U)\}. \tag{2.1}$$

For each of the lower bounding methods, we branched on the variable x_j for which $L_j^+ + L_j^-$ was the largest. If $L_j^+ + L_j^- = 0 \ \forall x_j \in I$, then we chose to branch on the fractional variable $x_j \in I'$ with largest objective function coefficient. We used the best bound node selection rule. Table 2.4 shows a summary and Table A.4 shows the full results of our experiment.

Branching Rule	Avg. Ranking
1 pivot (all)	1.64
10 pivots (all)	2.79
Strong	3.14
Knapsack	2.36

Table 2.4: Summary of Computational Results Using Lower Bound Based Branching Rules.

From the tables we make the following observations:

- It is in general too costly to perform 10 dual simplex pivots on all fractional variables.
- Strong branching can be highly effective on some problems, but the effectiveness is impacted greatly by the ability to select a suitable subset of variables on which to perform a number of dual simplex pivots.

- Performing one dual simplex pivots seems to be the best, which is a somewhat surprising result.

2.1.3 Combining Estimates and Lower Bounds

Getting lower bounding estimates on the degradation by performing one or more dual simplex pivots can be an expensive operation, but it gives us insight as to how the objective function value will change when branching on a variable *given the current formulation at a node*. We consider this "local" branching information. In contrast, pseudocost information is more "global" since these values are averages of observations taken throughout the branch and bound tree.

We want to consider combining this local and global branching information in some way. Specifically, we wish to estimate the true (up or down) degradation when branching on a variable x_j as

$$\hat{D}_j = \beta_0 + \beta_1 D_j + \beta_2 L_j, \tag{2.2}$$

D_j is the degradation measure taken from pseudocosts, and L_j is a lower bound on the degradation.

Since we only use these estimates $\hat{D}_j$ to rank the variables, the parameter β_0 is of no importance to us. The weights β_1 and β_2 could be chosen *a priori*, but defining the estimate in this way gives us the opportunity to create a dynamic branching scheme by altering these parameters as the search progresses. Given that we have evaluated n nodes, we do a regression to find the parameters β_1 and β_2 that give the best linear fit of the degradation $\hat{D}$ as a function of the pseudocost degradation D_j and lower bound on the degradation L_j. Doing a full regression at every node of the branch and bound tree may be too computationally expensive, but updating a regression from one observation to the next is relatively easy. The parameters β_1 and β_2 can be obtained from the solution to the linear system

$$\begin{bmatrix} n & \sum D^i & \sum L^i \\ \sum D^i & \sum (D^i)^2 & \sum D^i L^i \\ \sum L^i & \sum D^i L^i & \sum (L^i)^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \sum \hat{D}^i \\ \sum P^i \hat{D}^i \\ \sum L^i \hat{D}^i \end{bmatrix}, \tag{2.3}$$

where the superscript ^{i} on D, L, or $\hat{D}$ is meant to denote the value of this quantity at node i, regardless of which variable was branched on. This system has a closed form solution involving only a few arithmetic operations, so computing the regression parameters is not too costly. If the lower bound on the degradation or the pseudocost is zero, we do not update the regression coefficients, because this would have the effect of "biasing" the coefficients so that the importance of the lower bound L_j in the calculation was weighted too heavily.

2.1.4 Using Degradation Estimates

Once we have computed estimates or bounds on the degradation of the objective function given that we branch on a specific variable, we still must decide how to use this information to make our branching choice. Our goal is to maximize the difference in LP value of the relaxation from a parent to its children, but since there are two children of

each parent, there are different measures of change. Gauthier *et al.* [15] suggest trying to maximize the sum of the degradation on both branches, i.e. branch on the variable x_{j^*} with

$$j^* = \arg\max_j \{D_j^+ + D_j^-\}$$

or

$$j^* = \arg\max_j \{L_j^+ + L_j^-\}.$$

Bénichou *et al.* [5] and Beale [2] suggest instead to branch on the variable for which the smaller of the two estimated degradations is as large as possible. That is,

$$j^* = \arg\max_j \{\min\{D_j^+, D_j^-\}\}$$

or

$$j^* = \arg\max_j \{\min\{L_j^+, L_j^-\}\}.$$

Eckstein [13] suggests combining these ideas by branching on the variable

$$j^* = \arg\max\{\alpha_1 \min\{D_j^+, D_j^-\} + \alpha_2 \max\{D_j^+, D_j^-\}\} \tag{2.4}$$

or

$$j^* = \arg\max\{\alpha_1 \min\{L_j^+, L_j^-\} + \alpha_2 \max\{L_j^+, L_j^-\}\}. \tag{2.5}$$

Note that we can maximize the sum of the degradation on both branches by letting $\alpha_1 = \alpha_2 = 1$ in equation (2.4) or (2.5), and we can maximize the minimum degradation on both branches by letting $\alpha_1 = 1$, $\alpha_2 = 0$.

Table 2.5 shows a summary of the effect of varying the parameters α_1 and α_2 for our MIPLIB test instances. The full results can be found in Table A.5. For these runs, we have computed an estimated degradation as suggested by equation (2.2) with $\beta_1 = \beta_2 = 1$. We have set z_L to be the known optimal solution to the problem and used the best bound node selection rule. From this experiment we draw the following conclusions about using the degradation estimates or lower bounds to choose a branching variable:

- Both the up and down degradations should be considered.
- The importance of a variable is more related to the smaller of the two degradations.
- Using $(\alpha_1, \alpha_2) = (2, 1)$ in equation (2.4) or (2.5) appears to be a good choice.

2.1.5 Non-estimate Based Branching Rules

One further branching rule of note is suggested by Padberg and Rinaldi [32] and modified slightly by Jünger, Reinelt, and Thienel [22]. We call this method *enhanced branching*. Recall from (2.1) the set I' of variables on which we chose to make dual simplex pivots for strong branching. Enhanced branching chooses to branch on the variable $x_j \in I'$ for which the objective function coefficient is the largest. A variation of this method is the default branching rule invoked by MINTO (v2.0) [29].

(α_1, α_2)	Avg. Ranking
(1,0)	4.71
(10,1)	2.86
(2,1)	1.86
(1,1)	2.86
(1,2)	3.64
(1,10)	4.29

Table 2.5: Summary of Computational Results On Using Degradations to Compute Branching Variables

2.1.6 Computational Results

This subsection describes a comprehensive experiment to compare the effectiveness of various branching rules. Table 2.6 describes the branching rules we studied. We solved each instance of MIPLIB using each of these branching rules. To minimize the effects of factors other than branching in proving optimality of the solution, we have fixed z_L to be the known optimal solution to the problem. We evaluate nodes in "best bound" order. When estimates or lower bounds of the degradation are used in the branching decision, we use them by taking $(\alpha_1, \alpha_2) = (2,1)$ in the formula (2.4) or (2.5).

Branching Method	Description
B1	Branch on variable closest to 1/2.
B2	Enhanced Branching.
B3	Determine penalties by implicitly performing one dual simplex pivot. Strengthen penalties for integer variables as suggested by Tomlin [36].
B4	Pseudocost based branching.
B5	Use the estimates of degradation as in equation (2.2), with $\beta_1 = 1$, $\beta_2 = 1$.
B6	Use the estimates of degradation as in equation (2.2), and dynamically update the coefficients β_1 and β_2 by solving the system (2.3).
B7	Knapsack branching.

Table 2.6: Branching Rules Investigated

The instance *nwo4* of MIPLIB was excluded from the runs, since it required too much memory for our machines. The instance *dano3mip* was also excluded from the runs since the linear program is extremely difficult to solve – only 3 or 4 nodes can be evaluated in an hour. This left us with a test suite of 57 problem. Table A.6 shows the results of all the runs. We call a problem instance hard if not all branching methods could prove the optimality of the solution in less than two minutes. In this experiment, there were 34 "hard" instances. A summary of the experiment for the hard instances is given in

Table 2.7.

Method	Ranking (Min, Avg, Max)	Problems Solved	Computation Time (sec.)
B1	(1, 5.77, 7)	8	97327
B2	(1, 5.32, 7)	11	91323
B3	(2, 4.29, 7)	14	78467
B4	(1, 2.26, 7)	19	65061
B5	(1, 2.41, 6)	19	65743
B6	(1, 2.32, 5)	19	65625
B7	(4, 4.97, 7)	14	79372

Table 2.7: Summary of Branching Results for all MIPLIB Instances.

Based on these experiments, we make the following observations:

- The use of pseudocosts in an intelligent manner is essential to solve many of the problems.
- Combining pseudocosts with lower bounding information seems to improve the robustness of the branching method at a relatively small computational price.
- There is no branching method which clearly dominates the others (note that almost all methods came in last at least once), so a sophisticated MIP solver should allow many different options for selecting the branching variable.

2.2 GUB Dichotomy

When the problem has generalized upper bound (GUB) constraints of the form $\sum_{j \in T} x_j = 1$ (or $\sum_{j \in T} x_j \leq 1$) for some $T \subseteq I$, another problem subdivision scheme used in practice is called branching on a *GUB Dichotomy*. Here, a subset $T' \subseteq T$ for which the solution of the LP relaxation x^i at node i satisfies $0 < \sum_{j \in T'} x^i_j < 1$ is chosen. The constraint $\sum_{j \in T'} x_j = 0$ is enforced in one subregion, and the constraint $\sum_{j \in T \setminus T'} x_j = 0$ is enforced in the other subregion. Note that these constraints can again be enforced by fixing variables' bounds. When there exists a logical ordering of the variables in the set T, this set is sometimes called a *special ordered set* (SOS) and hence this division method is sometimes called *SOS branching*.

One advantage of branching on a GUB constraint instead of a variable is that the branch and bound tree is more "balanced". Suppose we have some GUB $\sum_{j \in T} x_j = 1$, and we choose to branch on a single variable j^*. If x_{j^*} is not an "important" variable, then it is likely that the set of feasible solutions for the node with $x_{j^*} = 0$ is very nearly the same as the set of feasible solutions for the original node. In this case, we have made little progress in our search.

A second advantage of branching on a GUB constraint occurs when the GUB is actually a SOS. In this case, the fractional LP solution may suggest which variable will be one in the optimal solution, and thereby demonstrate a good set T' on which to base the branching dichotomy. An example will make this point clear. Suppose we are

modeling a facility location problem in which we must decide on the size of a warehouse to build. The choices of sizes and their associated cost are shown in Table 2.8.

Size	Cost
10	100
20	180
40	320
60	450
80	600

Table 2.8: Warehouse sizes and costs

Using binary decision variables $x_1, x_2, \ldots, x_5$, we can model the cost of building the warehouse as

$$\text{COST} \equiv 100x_1 + 180x_2 + 320x_3 + 450x_4 + 600x_5.$$

The warehouse will have size

$$\text{SIZE} \equiv 10x_1 + 20x_2 + 40x_3 + 60x_4 + 80x_5,$$

and we have the SOS constraint

$$x_1 + x_2 + x_3 + x_4 + x_5 = 1.$$

If a linear programming solution has $x_1 = 0.2$ and $x_5 = 0.8$, then it might be trying to suggest building a warehouse of size

$$\text{SIZE} = 0.15(10) + 0.85(80) = 69.5,$$

in which case a sensible set on which to base the branching dichotomy would be $T' = \{1, 2, 3, 4\}$. This means that our dichotomy is based on whether or not to build a warehouse of size 80, which the LP solution suggests might be a profitable warehouse size.

In our example, we assigned an order to the variables based on their coefficients in the "size" constraint. In general SOS branching, constraints like the "size" constraint are termed *reference rows*. If the coefficients $a_1, a_2, \ldots a_{|T|}$ in the reference row are ordered such that $a_1 \leq a_2 \leq \ldots \leq a_{|T|}$, then a sensible set on which to base the branching dichotomy is

$$T' = \{j : a_j \leq \sum_{j \in T} a_j x_j^*\}.$$

The index

$$j' \equiv \arg \min_{j \in T \setminus T'} \{a_j\}$$

is usually called the *branch point* of the SOS.

Generalizing pseudocosts to help determine on which GUB to branch is not entirely straightforward. One simple idea is to extend the definition of down and up pseudocosts

to apply to a GUB. Given we have chosen an appropriate subset $T' \subseteq T$ on which to base our dichotomy, we can define the down and up pseudocosts for this GUB to be

$$P_j^- = \frac{z_{LP}^{i-} - z_{LP}^i}{\sum_{j \in T'} f_j^i} \qquad P_j^+ = \frac{z_{LP}^{i+} - z_{LP}^i}{1 - \sum_{j \in T'} f_j^i}.$$

This pseudocost definition does not take into account how the dichotomy was formed. Gauthier *et al.* [15] suggest assigning pseudocosts for each branch point in a SOS. Beale [2] gives a "row-based" method for determining both lower bounds and an estimate on the degradation if a GUB is branched on. Tomlin [35] extends the idea of performing one implicit dual simplex pivot to a set of variables.

When a problem has GUB constraints, we are faced with the question of whether to branch on a GUB or on a variable. This question has received little attention in the literature. As suggested by Beale and Forrest [3], if one uses a "row-based" method for determining estimates or lower bounds on the degradation of objective function value, then comparing the usefulness of branching on a GUB or a variable is straightforward.

2.2.1 Computational Results

If there is no logical order to the variables in a GUB and an "important enough" variable is chosen to be branched on, then the value of branching on a GUB as opposed to a variable is lessened. In Table 2.9 we show a comparison of GUB branching versus plain variable branching on the instances of MIPLIB where there is a significant number of GUB constraints. In this experiment, we adopt the following GUB branching strategy. We choose to branch on the GUB containing the greatest number of fractional variables in the current LP solution x^*. If there is no GUB containing at least 3 fractional variables, then we branch instead on a variable, using the adaptive combined estimate and lower bound strategy (B6). If we choose to branch on a GUB, the set T' on which the dichotomy is based is chosen in such a way so as to make $\sum_{j \in T'} x_j^*$ close to 0.5.

Table 2.9: GUB Versus Variable Branching

Problem	Branching Type	Nodes	Final Gap (%)	Sol. Time(sec.)
10teams	B4	641	0%	409
10teams	B6	113	0%	299
10teams	GUB	12813	0.758%	3600
air03	B4	3	0%	61
air03	B6	3	0%	62
air03	GUB	15	0%	66
air04	B4	155	0%	2453
air04	B6	203	0%	2674
air04	GUB	920	0.672%	3600
air05	B4	721	0%	1444
air05	B6	631	0%	1780
air05	GUB	1874	1.06%	3600
cap6000	B4	1862	7.01e-04%	3600

Problem	Branching Type	Nodes	Final Gap (%)	Sol. Time(sec.)
cap6000	B6	1703	6.21e-04%	3600
cap6000	GUB	1864	4.44e-04%	3600
harp2	B4	12058	7.55e-06%	3600
harp2	B6	4778	1.93e-02%	3600
harp2	GUB	4120	1.85e-01%	3600
l152lav	B4	469	0%	71
l152lav	B6	259	0%	67
l152lav	GUB	2955	0%	285
mitre	B4	23	0%	162
mitre	B6	23	0%	210
mitre	GUB	23	0%	131
mod010	B4	19	0%	11
mod010	B6	39	0%	14
mod010	GUB	41	0%	15
p0201	B4	77	0%	6
p0201	B6	63	0%	6
p0201	GUB	73	0%	7
p0282	B4	37	0%	3
p0282	B6	37	0%	3
p0282	GUB	39	0%	4
p0548	B4	9	0%	2
p0548	B6	9	0%	2
p0548	GUB	9	0%	2
p2756	B4	13	0%	7
p2756	B6	13	0%	7
p2756	GUB	13	0%	7

The results here seem to indicate that branching on a GUB may not be as important as choosing an important variable on which to base the branching dichotomy.

3 Node Selection

We now deal with the "select" portion of the branch and bound algorithm. When we make a decision to branch, we are solely concerned about maximizing the change in z^i_{LP} between a node N^i and its children. In selecting a node, our purpose is twofold: to find good integer feasible solutions or to prove that no solution better than our current one with value z_L exists. Therefore, the quality of the current solution value z_L is an important factor in determining which node to select for evaluation. Hence also the decision of whether or not a heuristic procedure is used in order to obtain good integer feasible solutions is a factor that must be considered when choosing a node selection rule. If a heuristic procedure is used, then node selection rules that emphasize proving that no better solution exist rather than finding improved integer feasible solutions may be preferred. Since many of the early branch and bound codes for solving MIP did not contain a heuristic as part of their solution procedure, existing ideas for node selection

deserve more exploration. Here, we provide a brief survey of node selection methods. We categorize the node selection methods as *static* methods, *estimate-based* methods, *two-phase* methods, and *backtracking* methods. In addition, we introduce a new calculation on which to base estimation methods, and we perform experiments to test the effectiveness of the various methods.

3.1 Static Methods

A popular way to choose which subproblem to explore is to choose the one with the largest value of z_U^i. There are theoretical reasons for making this choice, since for a fixed branching rule, selecting problems in this way minimizes the number evaluated nodes before completing the search. This node selection rule is usually called *best-first* or *best-bound* search.

At the other extreme is a selection rule called *depth-first* search. As the name suggests, the solution space is searched in a depth first manner.

Both these methods have inherent strengths and weaknesses. Best-first search will tend to minimize the number of nodes evaluated and at any point during the search is attempting to improve the global upper bound on the problem. Therefore best-first search concentrates on proving that no solution better than the current one exists. Memory requirements for searching the tree in a best-first manner may become prohibitive if good lower bounds are not found early, leading to relatively little pruning of the tree. Also, the search tree tends to be explored in a breadth-first fashion, so one linear program to solve has little relation to the next – leading to higher computation times.

Depth-first search overcomes both these shortcomings of best-first search. In fact, searching the tree in a depth first manner will tend to minimize the memory requirements, and the changes in the linear program from one node to the next are minimal – usually just changing one variable's bound. Depth-first search has another advantage over best-first search in finding feasible solutions since feasible solutions tend to be found deep in the search tree. Depth-first search was the strategy proposed by Dakin [10] and Little *et al.* [27], primarily due to the small memory capabilities of computers at that time. Despite its advantages, depth first search can lead to extremely large search trees. This stems from the fact that we may evaluate a good many nodes that would have been fathomed had a better value of z_L been known. For larger problems, depth first search has been shown to be impractical [14]. However, this conclusion was made in the days before primal heuristics were incorporated into most MIP codes, so depth first search deserves to be reexamined.

3.2 Estimate-based Methods

Neither best first search nor depth first search make any intelligent attempt to select nodes that may lead to improved integer feasible solutions. What would be useful is some estimate of the value of the best feasible integer solution obtainable from a given node of the branch and bound tree. The *best projection* criterion, introduced by Hirst [20] and Mitra [28] and the *best estimate criterion* found in Bénichou *et al.* [5] and Forrest *et al.* [14], are ways to incorporate this idea into a node selection scheme.

The best projection method and the best estimate method differ in how they determine an estimate of the best solution obtainable from a node. Given an estimate E^i, they both select the node in the active set for which this value is largest. For any node N^i, let $s^i \equiv \sum_{j \in I} \min(f_j, 1 - f_j)$ denote the sum total of its integer infeasibilities. Also, let the root node of the branch and bound tree be denoted by N^0. The best projection criterion for node selection is to choose the node with the highest value of

$$E_i = z_U^i + \left(\frac{z_L - z_U^0}{s_0} \right) s^i. \tag{3.6}$$

The value $\lambda = (z_L - z_U^0)/s_0$ can be thought of as the change in objective function value per unit decrease in infeasibility. Note that this method requires that there be a value of z_L.

The estimate obtained by the best projection method does not take into account which variables are fractional or the individual costs for satisfying each variable. A natural extension of the best projection idea would be to use pseudocosts in obtaining an estimate of the value of the best solution obtainable from a node. This extension is what is known as the best estimate criterion. Here, the estimate of the best solution obtainable from a node is

$$E_i = z_U^i + \sum_{j \in I} \min(P_j^- f_j, P_j^+ (1 - f_j)). \tag{3.7}$$

This estimate has the advantage that it does not require a value of z_L.

The estimate of the best solution obtainable from a node given by (3.7) assumes that we will always be able to round a fractional variable to the closest integer and obtain a feasible integer solution, which is somewhat optimistic. A more realistic estimate would be the following:

$$\begin{aligned} E_i = z_U^i \quad &+ \sum_{j \in I : f_j \leq 0.5} (f_j P_j^- q_j + (1 - f_j) P_j^+ (1 - q_j)) \\ &+ \sum_{j \in I : f_j > 0.5} (f_j P_j^- (1 - q_j) + (1 - f_j) P_j^+ q_j), \end{aligned} \tag{3.8}$$

where q_j is the "probability" that we will be able to round a fractional solution to the closest integer and obtain a feasible integer solution. Note that if $q_j = 1 \ \forall j \in I$, then (3.8) reduces to (3.7).

An obvious question is how to calculate q_j for a variable. Since we are using a primal heuristic to generate feasible solutions, we can get an estimate of the percentage of variables that are able to be rounded to the nearest integer. We do this by comparing a feasible integer solution $\hat{x}$ obtained by a primal heuristic to the fractional solution x at that node. Define the *flip-percentage* ζ as

$$\zeta \equiv \frac{|\{j \in I : |\hat{x}_j - x_j| > 0.5\}|}{|I|}. \tag{3.9}$$

Regression studies showed that ζ calculated as in (3.9) is approximately constant for a given problem instance over a wide range of feasible solutions. To obtain a less optimistic

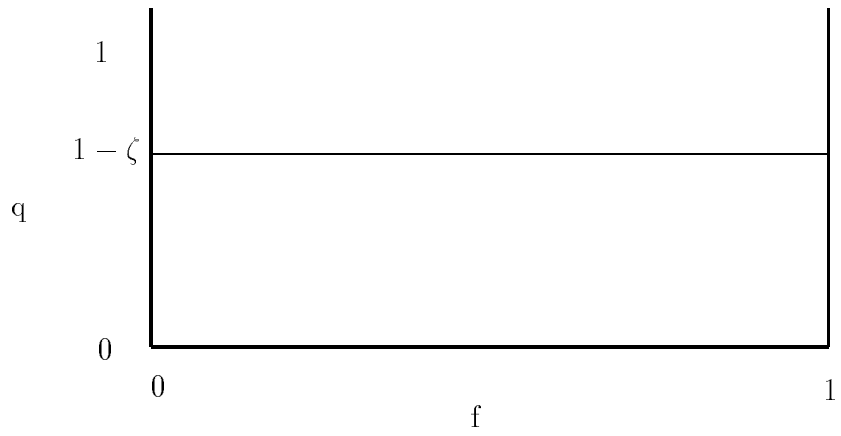

Figure 3.3: A graph of q_j.

estimate E_i, we could use equation (3.8), where $q_j = 1 - \zeta$. A graph of q_j is shown in Figure 3.3.

We make two modifications to the graph of q_j in order to more accurately reflect the probability. Intuitively, fractional variables close to integer values are more likely to be able to be rounded to this value in a feasible solution. To incorporate this idea into our calculation of q_j, we bias the graph of q_j to look as in Figure 3.4.

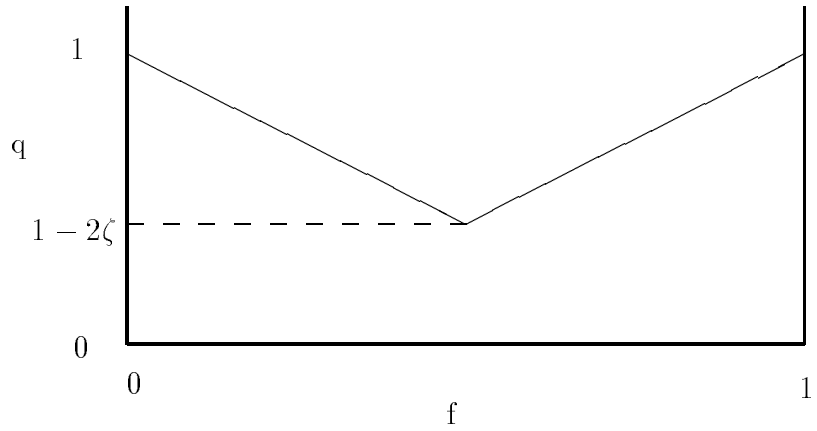

Figure 3.4: A graph of q_j taking into account x_j

Suppose we have found a number of feasible solutions, and in each of these solutions $x_j = 1$. We might conjecture that there is something inherent in the problem structure which will force $x_j = 1$ in a feasible solution. Our second adjustment of the graph of q_j is to try to capture this simple notion. For each variable, we keep track of the average feasible solution value $\bar{x}_j$ and construct a graph shown in Figure 3.5. For example, if $\bar{x}_j = 1$, and $f_j = 0.2$, Figure 3.5 states that the probability that $x_j = 0$ in a feasible integer solution is zero. The final graph of q_j is a weighted combination of the graphs

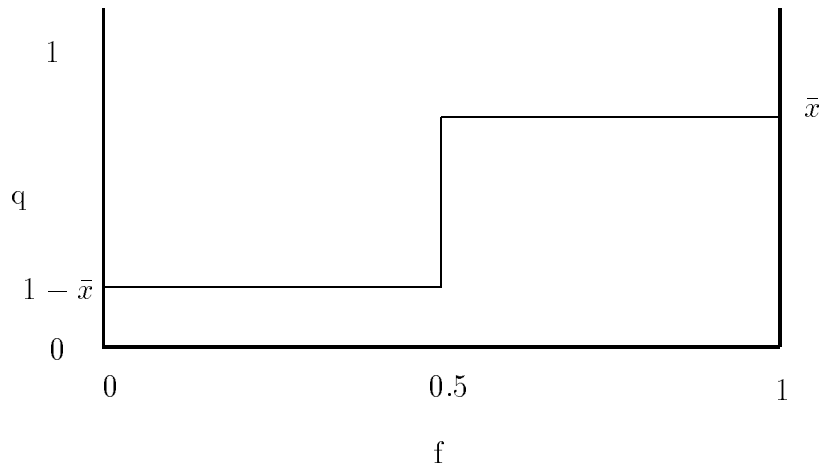

Figure 3.5: A graph of q_j taking into account x_j

shown in Figures 3.4 and 3.5. The weight given to q_j in the graph in Figure 3.5 gets larger as more feasible solutions are found.

We performed a study to compare the various estimation methods. At certain nodes of the branch and bound tree, we computed estimates of the best solution obtainable from that node using the estimate from the Best Projection method (3.6), the pseudocost estimate (3.7), and the "adjusted pseudocost" estimate (3.8). We then solved the problem with the formulation at that node to optimality for comparison purposes. Table B.1 shows the results. The experiments show that the best projection estimate often underestimates the true solution, and the pseudocost estimate usually overestimates the true solution. The adjusted pseudocost estimate also usually overestimates the solution, but not by as much as the regular pseudocost estimate. These characteristics are not that important by themselves, but have an impact on the performance of the backtracking methods that use them.

3.3 Two-Phase Methods

Since we have two goals in node selection: finding good feasible solutions and proving that no better feasible solutions exist, it is natural to develop node selection strategies that switch from one goal to the other in the course of the algorithm. In the first phase, we are interested in determining good feasible solutions, while in the second phase, we are interested in proving that the solutions we obtained in the first phase are good or optimal. Perhaps the simplest "two-phase" algorithm is to perform depth first search until a feasible solution is found, then switch to best first search. A slight variation of this strategy is used by Eckstein [13].

Forrest *et al.* [14] and Beale [2] propose a two-phase method that first chooses nodes according to the best-estimate criterion. Once a feasible solution is found, they state that it is better to select nodes that maximize a different criterion, known as the *percentage error*. The percentage error can be thought of as the amount by which the estimate of

the solution obtainable from a node must be in error for the current solution x^* to not be optimal. The percentage error of a node i is

$$PE_i = 100\frac{z_L - E_i}{z_U^i - z_L}.$$

3.4 Backtracking Methods

Define a *superfluous node* as a node N^i that has $z_{LP}^i < z^*$. Searching the tree in a best first manner will ensure that no superfluous nodes are evaluated. If, however, one can be assured that all (or most) of the superfluous nodes will be fathomed, (which is the case if $z_L = z^*$), the memory and speed advantages of depth first search make this method the most preferable. Various authors have proposed strategies that attempt to go depth first as much as possible while minimizing the number of superfluous nodes evaluated [5] [7] [15] [8]. Given some estimate E_0 of the optimal objective function value z^*, the tree is searched in a depth first fashion as long as $z_{LP}^i > E_0$. If $z_{LP}^i \leq E_0$, then a node is selected by a different criterion such as best-first or best-estimate. The methods differ in the manner in which they obtain E_0 and in which criterion they use when deciding to backtrack.

There is a tradeoff in how "aggressive" one wants to be when calculating an estimate. If the estimate is too large, then the tree will be searched in a best-first or best-estimate fashion and none of the advantages of going depth first is obtained. If the estimate is too small, then the tree is searched in a more depth first fashion and many superfluous nodes may be evaluated. This point must be kept in mind when selecting an estimation method on which to base a backtracking node selection rule.

3.5 Branch Selection

Typical branching is based on a dichotomy that creates two new nodes for evaluation. The node selection scheme must also answer the question of how to rank the order of evaluation for these nodes. Schemes that prioritize the nodes based on an estimate of the optimal solution obtainable from that node have a built-in answer to this question, since distinct estimates are assigned to the newly created nodes. For schemes that do not distinguish between the importance of the two newly created nodes, such as depth-first, researchers have made the following suggestion. Suppose that we have based the branching dichotomy on the variable x_{j^*}, then we select the down node first if $f_{j^*}^i > 1 - f_{j^*}^i$ and the up node first otherwise [25]. If estimates are not available, we will use this rule for selecting whether to evaluate the down or up child of a node.

3.6 Computational Results

We performed an experiment to compare many of the node selection rules we have discussed. Each of the problems in MIPLIB was solved using the node selection methods detailed in Table 3.10. For a branching method, we use the adaptive regression method B6 in Table 2.6. All advanced features of MINTO were used, which includes a diving heuristic that is invoked every ten nodes of the branch and bound tree.

Node Selection Method	Description
N1	Best Bound.
N2	Depth First.
N3	Depth First until a solution is obtained, then Best Bound.
N4	Best Estimate (normal pseudocost) until a solution is obtained, then Percentage Error.
N5	Best Projection.
N6	Best Estimate (normal pseudocost).
N7	Best Estimate (adjusted pseudocost).
N8	Backtrack. Best Projection Estimate. When backtracking, select node by best bound criterion.
N9	Backtrack. Best Estimate (normal pseudocost). When backtracking, select node by best bound criterion.
N10	Backtrack. Best Estimate (adjusted pseudocost). When backtracking, select node by best bound criterion.
N11	Backtrack. Best Projection Estimate. When backtracking, select node by best estimate criterion.
N12	Backtrack. Best Estimate (normal pseudocost). When backtracking, select node by best estimate criterion.
N13	Backtrack. Best Estimate (adjusted pseudocost). When backtracking, select node by best estimate criterion.

Table 3.10: Node Selection Rules Investigated

As in branching methods experiment the instances *nwo4* and *dano3mip* were excluded from this experiment. In addition, the instance *arki001* was also excluded, since during the heuristic phase, we encounter a linear program which takes more than one hour of CPU time to solve. This left us with 56 problems in the test suite. Table B.2 shows the full results of this experiment, and Table 3.11 shows a summary of the results. When ranking the performance of a node selection method on a given instance, we used the following criteria:

- Methods are ranked first by the value of the best solution obtained.
- If two methods find the same solution, the method with the lower provable optimality gap is ranked higher.
- If both methods find the same solution and optimality gap, they are ranked according to computation time.
- Ties are allowed.

In computing the rankings, instances where each node selection method was able to prove the optimality of the solution and the difference between best and worst methods' computation times was less than 30 seconds were excluded. This left us with 33 instances.

Method	Ranking (Min, Avg, Max)	Times No Sol'n Found	Times Optimal Sol'n Found	Computation Time (sec.)
N1	(1, 6.67, 13)	2	8	68189
N2	(1, 8.42, 13)	1	3	80005
N3	(1, 7.12, 13)	1	7	72207
N4	(1, 6.12, 13)	2	10	72324
N5	(1, 7.03, 13)	0	7	79082
N6	(1, 5.67, 12)	2	11	66147
N7	(1, 5.94, 13)	1	8	67823
N8	(1, 7.00, 13)	1	7	74375
N9	(1, 6.85, 12)	1	8	66475
N10	(1, 6.94, 13)	1	8	68106
N11	(1, 7.42, 13)	1	2	78568
N12	(1, 5.70, 13)	1	10	65861
N13	(1, 5.42, 11)	1	11	67944

Table 3.11: Summary of Node Selection Method Experiment.

From the tables it is difficult to determine a clear winner among the node selection methods, but we can make the following observations:

- Pseudocost-based node estimate methods or combining a pseudocost based estimate method in backtracking seems to be the best idea for node selection.

- Backtracking methods that select the node with the best estimate when backtracking generally outperform those methods that choose the best bound node when backtracking.
- There is no node selection method which clearly dominates the others (note that almost all methods came in last at least once), so a sophisticated MIP solver should allow many different options for selecting the next node to evaluate.
- Even in the presence of a primal heuristic, depth first search performs poorly in practice.

4 Conclusions

We have examined a wide variety of branching methods and node selection rules for mixed integer programming. The strategies were examined in conjunction with many of the advanced features found in today's sophisticated MIP solvers to see if strategies developed decades ago are still practical today. In general, we conclude that the early methods are indeed still practical today. Especially pseudocost-based-methods, when used intelligently, seem to be very beneficial.

Some important topics for further investigation are

- Can the "local" and "global" branching information from simplex pivots and pseudocosts be combined in a more intelligent way than by simple linear regression.
- Can we develop intelligent ways to choose a "good" subset of variables for strong branching?
- Can we get better estimates of the optimal solution obtainable from a node?
- Can we develop search strategies that adapt themselves based on the observed behavior for a given problem instance?

5 Acknowledgment

The authors would like to acknowledge Alper Atamtürk for many useful discussions.

References

[1] D. Applegate, R. Bixby, W. Cook, and V. Chvátal, 1996. Personal communication.

[2] E. M. L. Beale. Branch and bound methods for mathematical programming systems. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II*, pages 201–219. North Holland Publishing Co., 1979.

[3] E. M. L. Beale and J. J. H. Forrest. Global optimization using special ordered sets. *Mathematical Programming*, 10:52–69, 1976.

[4] E. M. L. Beale and R. E. Small. Mixed integer programming by a branch and bound method. In W. H. Kalenich, editor, *Proceedings IFIP Congress 65*, volume 2, pages 450–451, 1966.

[5] 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.

[6] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *SIAM News*, 1996. Submitted.

[7] R. Breu and C. A. Burdet. Branch and bound experiments in zero-one programming. *Mathematical Programming*, 2:1–50, 1974.

[8] CPLEX Optimization, Inc. *Using the CPLEX Callable Library*, 1995.

[9] H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[10] R. J. Dakin. A tree search algorithm for mixed programming problems. *Computer Journal*, 8(3):250–255, 1965.

[11] R. E. Davis, D. A. Kendrick, and M. Weitzman. A branch and bound algorithm for zero-one mized integer programming problems. Technical Report Development Economic Report 69, Center for International Affairs, Harvard University, 1967.

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

[13] J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. *SIAM Journal on Optimization*, 4(4):794–814, 1994.

[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(5):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] A. M. Geoffrion and R. E. Marsten. Integer programming algorithms: A framework and state-of-the-art survey. *Management Science*, 18(9):465–491, 1972.

[17] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Cover inequalities for 0-1 linear programs: Computation. *INFORMS Journal on Computing*, 1994. Submitted.

[18] O Günlük. A branch-and-Cut algorithm for capacitated network design problems. Submitted, 1996.

[19] M. T. Hajian and G. Mitra. Design and testing of an integrated branch and bound algorithm for piecewise linear and discrete programming problems. Technical Report TR/01/95, Brunel, the University of West London, London, 1995.

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

[21] W. C. Healy Jr. Multiple choice programming. *Operations Research*, 12:122–138, 1964.

[22] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. *Zeitschrift für Operations Research*, 40:183–217, 1994.

[23] T. H. Lai and A. Sprague. A note on anomolies in parallel branch and bound algorithms with one-to-one bounding functions. *Information Processing Letters*, 23:119–122, 1986.

[24] T.H. Lai and S. Sahni. Anomalies in parallel branch and bound algorithms. In *Proceedings of the 1983 International Conference on Parallel Processing*, pages 183–190, 1983.

[25] A. Land and S. Powell. Computer codes for problems of integer programming. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II*, pages 221–269. North Holland Publishing Co., 1979.

[26] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[27] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 21:972–989, 1963.

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

[29] G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

[30] B. Nygreen. Branch and bound with estimation based on pseudo shadow prices. *Mathematical Programming*, 52(1):59–69, 1991.

[31] M. Padberg and T. J. Van Roy amd L. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

[32] M. W. Padberg and G. Rinaldi. A branch and cut algorithm for the solution of large scale traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[33] T. J. Van Roy and L. A. Wolsey. Solving mixed integer 0-1 programs by automatic reformulation. *Operations Research*, 35:45–57, 1987.

[34] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[35] J. A. Tomlin. Branch and bound methods for integer and non-convex progrmming. In J. Abadie, editor, *Integer and Non-linear Programming*. North Holland, Amsterdam, 1970.

[36] J. A. Tomlin. An improved branch-and-bound method for integer programming. *Operations Research*, 19:1070–1075, 1971.

[37] J. M. Troya and M. Ortega. Study of parallel branch-and-bound algorithms with best-bound-first search. *Parallel Computing*, 11:121–126, 1989.

Appendix A – Branching Tables

Table A.1: The Effect of Pseudocost Initialization

Problem	Initialization Method	Nodes	Final Gap	Sol. Time(sec.)
air04	Obj. Coef.	2618	0.159%	3600
air04	Computed	0	XXX	3600
air04	Computed Fractional	195	0%	2351
air04	Averaged	1588	0.950%	3600
arki001	Obj. Coef.	31915	0.0155%	3600
arki001	Computed	30739	0.00350%	3600
arki001	Computed Fractional	38583	0.00454%	3600
arki001	Averaged	36086	0.0101%	3600
bell3a	Obj. Coef.	30926	0%	99
bell3a	Computed	28627	0%	96
bell3a	Computed Fractional	28319	0%	97
bell3a	Averaged	30864	0%	98
bell5	Obj. Coef.	120000	0.387%	XXX
bell5	Computed	21365	0%	66
bell5	Computed Fractional	14013	0%	45
bell5	Averaged	64361	0%	240
gesa2	Obj. Coef.	32815	0.0572%	3600
gesa2	Computed	34681	0.0233%	3600
gesa2	Computed Fractional	40885	0.00814%	3600
gesa2	Averaged	32951	0.0226%	3600
harp2	Obj. Coef.	4900	0.185%	3600
harp2	Computed	11145	7.75e-06%	3600
harp2	Computed Fractional	10695	9.02e-06%	3600
harp2	Averaged	6476	0.148%	3600
l152lav	Obj. Coef.	7481	0%	481
l152lav	Computed	1573	0%	241
l152lav	Computed Fractional	1511	0%	133
l152lav	Averaged	9039	0%	533
mod011	Obj. Coef.	414	5.15%	3600
mod011	Computed	372	6.09%	3600
mod011	Computed Fractional	418	5.17%	3600
mod011	Averaged	432	5.37%	3600

Problem	Initialization Method	Nodes	Final Gap	Sol. Time(sec.)
pp08a	Obj. Coef.	85000	5.47%	XXX
pp08a	Computed	85000	5.20%	XXX
pp08a	Computed Fractional	83000	5.10%	XXX
pp08a	Averaged	85000	5.73%	XXX
qiu	Obj. Coef.	3614	112%	3600
qiu	Computed	4116	131%	3600
qiu	Computed Fractional	4310	113%	3600
qiu	Averaged	3717	91.3%	3600
qnet1	Obj. Coef.	30000	3.60%	XXX
qnet1	Computed	73	0%	118
qnet1	Compute Fractional	59	0%	20
qnet1	Averaged	4441	0%	511
rgn	Obj. Coef.	1953	0%	20
rgn	Computed	3465	0%	40
rgn	Computed Fractional	2907	0%	30
rgn	Averaged	3201	0%	37
stein45	Obj. Coef.	54841	0%	2540
stein45	Computed	61283	0%	2553
stein45	Computed Fractional	60315	0%	2283
stein45	Averaged	61933	0%	3008
vpm2	Obj. Coef.	14453	0%	435
vpm2	Computed	9375	0%	215
vpm2	Computed Fractional	9431	0%	210
vpm2	Averaged	16793	0%	486

Table A.2: The Effect of Pseudocost Update Method on Various MIPLIB instances

Problem	Update Method	Nodes	Final Gap (%)	Sol. Time(sec.)
air04	Average	1986	0.104%	3600
air04	First	1523	0.458%	3600
air04	Last	1932	0.105%	3600
arki001	Average	37821	0.00456%	3600
arki001	First	49037	0.00366%	3600
arki001	Last	39512	0.00288%	3600
bell3a	Average	28319	0%	100
bell3a	First	29841	0%	108
bell3a	Last	28293	0%	99
bell5	Average	14013	0%	45
bell5	First	15941	0%	51
bell5	Last	119421	0%	403
gesa2	Average	39190	0.00882%	3600
gesa2	First	37147	0.0155%	3600
gesa2	Last	35682	0.0181%	3600
harp2	Average	10110	9.51e-06%	3600
harp2	First	9734	8.35e-03%	3600
harp2	Last	9910	8.85e-06%	3600
l152lav	Average	1511	0%	138
l152lav	First	1639	0%	141
l152lav	Last	1323	0%	125
mod011	Average	407	5.21%	3600
mod011	First	416	5.64%	3600
mod011	Last	403	5.11%	3600
pp08a	Average	101574	4.76%	3600
pp08a	First	103054	5.46%	3600
pp08a	Last	97446	4.9%	3600
qiu	Average	4172	114%	3600
qiu	First	4844	181%	3600
qiu	Last	4062	124%	3600
qnet1	Average	57	0%	20
qnet1	First	57	0%	20
qnet1	Last	57	0%	20
rgn	Average	2907	0%	32
rgn	First	3639	0%	41
rgn	Last	2823	0%	32
stein45	Average	60315	0%	2286
stein45	First	66552	5%	3600
stein45	Last	64079	0%	2463
vpm2	Average	9431	0%	216

Problem	Update Method	Nodes	Final Gap (%)	Sol. Time(sec.)
vpm2	First	12517	0%	270
vpm2	Last	10689	0%	258

Problem	Cuts?	Percent of Nodes With Useful Estimates
danoint	Y	98.40%
danoint	N	94.00%
enigma	Y	5.13%
enigma	N	23.00%
lseu	Y	100.00%
lseu	N	92.00%
misc03	Y	66.45%
misc03	N	90.69%
misc07	Y	78.05%
misc07	N	92.62%
mod010	Y	100.00%
mod010	N	65.79%
p0201	Y	81.20%
p0201	N	66.00%
p2756	Y	80.00%
p2756	N	66.25%
pk1	Y	38.40%
pk1	N	44.40%
rentacar	Y	100.00%
rentacar	N	0.00%
rgn	Y	24.56%
rgn	N	0.85%
rout	Y	88.71%
rout	N	71.08%
vpm1	Y	9.38%
vpm1	N	100.00%

Table A.3: The Effect of Valid Inequalities on Percentage of Useful Lower Bound Estimates

Table A.4: Computational Results Using Different Lower Bound Based Branching Methods

Problem	Branching Method	Nodes	Final Gap (%)	Sol. Time(sec.)
air04	10 pivots (all)	29	0.94%	3600
air04	1 pivot (all)	884	0.305%	3600
air04	knapsack	572	0.511%	3600
air04	strong	131	0%	1516
arki001	10 pivots (all)	1939	0.0073%	3600
arki001	1 pivot (all)	20048	0.00288%	3600
arki001	knapsack	15269	0.00591%	3600
arki001	strong	8459	0.0135%	3600
bell3a	10 pivots (all)	28185	0%	284
bell3a	1 pivot (all)	28309	0%	216
bell3a	knapsack	28465	0%	220
bell3a	strong	30001	0%	307
bell5	10 pivots (all)	76817	0.053%	3600
bell5	1 pivot (all)	90758	0.0418%	3600
bell5	knapsack	89473	0.0546%	3600
bell5	strong	97745	3.67%	3600
gesa2	10 pivots (all)	7755	0.0257%	3600
gesa2	1 pivot (all)	27152	0.013%	3600
gesa2	knapsack	27982	0.00515%	3600
gesa2	strong	15638	0.305%	3600
harp2	10 pivots (all)	465	0.105%	3600
harp2	1 pivot (all)	3942	0.0888%	3600
harp2	knapsack	2800	0.124%	3600
harp2	strong	3377	0.132%	3600
l152lav	10 pivots (all)	269	0%	254
l152lav	1 pivot (all)	477	0%	85
l152lav	knapsack	7327	0%	984
l152lav	strong	699	0%	197
mod011	10 pivots (all)	122	6.28%	3600
mod011	1 pivot (all)	403	5.81%	3600
mod011	knapsack	410	6.08%	3600
mod011	strong	334	4.24%	3600
pp08a	10 pivots (all)	10626	8.4%	3600
pp08a	1 pivot (all)	44729	8.81%	3600
pp08a	knapsack	43402	8.81%	3600
pp08a	strong	33938	10.3%	3600
qiu	10 pivots (all)	354	183%	3600
qiu	1 pivot (all)	3765	194%	3600
qiu	knapsack	3810	190%	3600

Problem	Branching Method	Nodes	Final Gap (%)	Sol. Time(sec.)
qiu	strong	560	191%	3600
qnet1	10 pivots (all)	53	0%	61
qnet1	1 pivot (all)	295	0%	44
qnet1	knapsack	157	0%	27
qnet1	strong	14644	3.14%	3600
rgn	10 pivots (all)	1885	0%	45
rgn	1 pivot (all)	2039	0%	24
rgn	knapsack	2127	0%	26
rgn	strong	2097	0%	40
stein45	10 pivots (all)	13644	5%	3600
stein45	1 pivot (all)	51641	0%	3357
stein45	knapsack	52365	0%	3400
stein45	strong	24553	0%	3508
vpm2	10 pivots (all)	4611	0%	298
vpm2	1 pivot (all)	13081	0%	395
vpm2	knapsack	15397	0%	490
vpm2	strong	9957	0%	471

Table A.5: The Effect of Combining Up and Down Pseudocost Information When Choosing a Branching Variable

Problem	(α_1, α_2)	Nodes	Final Gap (%)	Sol. Time(sec.)
air04	(10, 1)	147	0%	2722
air04	(2, 1)	167	0%	2604
air04	(1, 1)	189	0%	2598
air04	(1, 0)	203	0%	2857
air04	(1, 2)	221	0%	2691
air04	(1, 10)	257	0%	2727
arki001	(10, 1)	25385	0.00327%	3600
arki001	(2, 1)	21096	0.00238%	3600
arki001	(1, 1)	22158	0.00264%	3600
arki001	(1, 0)	15625	0.0049%	3600
arki001	(1, 2)	22002	0.00313%	3600
arki001	(1, 10)	20059	0.00501%	3600
bell3a	(10, 1)	28449	0%	110
bell3a	(2, 1)	28419	0%	109
bell3a	(1, 1)	28321	0%	109
bell3a	(1, 0)	28559	0%	111
bell3a	(1, 2)	28307	0%	109
bell3a	(1, 10)	28445	0%	109
bell5	(10, 1)	202893	0%	771
bell5	(2, 1)	33491	0%	121
bell5	(1, 1)	17527	0%	63
bell5	(1, 0)	281761	0%	1056
bell5	(1, 2)	16599	0%	60
bell5	(1, 10)	17153	0%	62
gesa2	(10, 1)	33999	0.00278%	3600
gesa2	(2, 1)	33646	0.00397%	3600
gesa2	(1, 1)	31788	0.00724%	3600
gesa2	(1, 0)	26629	0.189%	3600
gesa2	(1, 2)	29858	0.0116%	3600
gesa2	(1, 10)	29423	0.0116%	3600
harp2	(10, 1)	12020	7.09e-06%	3600
harp2	(2, 1)	9084	7.46e-06%	3600
harp2	(1, 1)	5704	3.04e-02%	3600
harp2	(1, 0)	3486	2.21e-02%	3600
harp2	(1, 2)	5729	2.89e-02%	3600
harp2	(1, 10)	5382	2.76e-02%	3600
l152lav	(10, 1)	209	0%	65
l152lav	(2, 1)	313	0%	74
l152lav	(1, 1)	475	0%	103

Problem	(α_1, α_2)	Nodes	Final Gap (%)	Sol. Time(sec.)
l152lav	(1, 0)	407	0%	93
l152lav	(1, 2)	1005	0%	167
l152lav	(1, 10)	1061	0%	172
mod011	(10, 1)	382	4.09%	3600
mod011	(2, 1)	394	4.39%	3600
mod011	(1, 1)	373	5.17%	3600
mod011	(1, 0)	375	4.39%	3600
mod011	(1, 2)	397	5.56%	3600
mod011	(1, 10)	396	5.65%	3600
pp08a	(10, 1)	65737	6.66%	3600
pp08a	(2, 1)	71116	5.54%	3600
pp08a	(1, 1)	69241	5.84%	3600
pp08a	(1, 0)	66081	7.81%	3600
pp08a	(1, 2)	69356	6.33%	3600
pp08a	(1, 10)	69515	6.92%	3600
qiu	(10, 1)	3707	105%	3600
qiu	(2, 1)	3775	110%	3600
qiu	(1, 1)	3786	114%	3600
qiu	(1, 0)	3534	117%	3600
qiu	(1, 2)	3770	121%	3600
qiu	(1, 10)	4000	127%	3600
qnet1	(10, 1)	147	0%	44
qnet1	(2, 1)	73	0%	25
qnet1	(1, 1)	57	0%	22
qnet1	(1, 0)	309	0%	71
qnet1	(1, 2)	57	0%	21
qnet1	(1, 10)	61	0%	21
rgn	(10, 1)	2101	0%	25
rgn	(2, 1)	2137	0%	25
rgn	(1, 1)	2305	0%	27
rgn	(1, 0)	2387	0%	29
rgn	(1, 2)	2327	0%	27
rgn	(1, 10)	2421	0%	28
stein45	(10, 1)	50877	0%	2743
stein45	(2, 1)	47241	0%	2386
stein45	(1, 1)	46749	0%	2404
stein45	(1, 0)	57411	0%	2991
stein45	(1, 2)	48655	0%	2515
stein45	(1, 10)	49269	0%	2547
vpm2	(10, 1)	7939	0%	206
vpm2	(2, 1)	7143	0%	183

Problem	(α_1, α_2)	Nodes	Final Gap (%)	Sol. Time(sec.)
vpm2	$(1, 1)$	8113	0%	206
vpm2	$(1, 0)$	10365	0%	301
vpm2	$(1, 2)$	9061	0%	233
vpm2	$(1, 10)$	9859	0%	266

Table A.6: Comparison of Branching Methods

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
10teams	B1	16908	0.216%	3600
10teams	B2	16849	0.758%	3600
10teams	B3	8838	0.758%	3600
10teams	B4	641	0%	409
10teams	B5	113	0%	299
10teams	B6	113	0%	299
10teams	B7	8952	0.433%	3600
air03	B1	3	0%	61
air03	B2	5	0%	61
air03	B3	3	0%	61
air03	B4	3	0%	61
air03	B5	3	0%	62
air03	B6	3	0%	62
air03	B7	3	0%	62
air04	B1	1173	0.48%	3600
air04	B2	2755	0%	3275
air04	B3	956	0.24%	3600
air04	B4	155	0%	2453
air04	B5	167	0%	2600
air04	B6	203	0%	2674
air04	B7	771	0.219%	3600
air05	B1	2116	0.642%	3600
air05	B2	3346	0.0997%	3600
air05	B3	1552	0.37%	3600
air05	B4	721	0%	1444
air05	B5	691	0%	1876
air05	B6	631	0%	1780
air05	B7	931	0.78%	3600
arki001	B1	26404	0.0142%	3600
arki001	B2	29088	0.0157%	3600
arki001	B3	20333	0.00326%	3600
arki001	B4	40189	0.00294%	3600
arki001	B5	21345	0.00238%	3600
arki001	B6	21989	0.00243%	3600
arki001	B7	17230	0.00569%	3600
bell3a	B1	81541	0%	1111
bell3a	B2	39281	0%	345
bell3a	B3	28319	0%	214
bell3a	B4	28383	0%	100
bell3a	B5	28419	0%	109

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
bell3a	B6	28391	0%	109
bell3a	B7	28443	0%	217
bell5	B1	78793	3.81%	3600
bell5	B2	82805	3.77%	3600
bell5	B3	92357	0.0402%	3600
bell5	B4	14341	0%	45
bell5	B5	33491	0%	121
bell5	B6	19187	0%	69
bell5	B7	90314	0.0429%	3600
blend2	B1	2735	0%	252
blend2	B2	1827	0%	163
blend2	B3	1908	0%	178
blend2	B4	1163	0%	58
blend2	B5	1263	0%	76
blend2	B6	1327	0%	74
blend2	B7	1607	0%	124
cap6000	B1	1708	9.56e-04%	3600
cap6000	B2	1301	5.04e-03%	3600
cap6000	B3	1750	5.09e-04%	3600
cap6000	B4	1862	7.01e-04%	3600
cap6000	B5	1807	4.52e-04%	3600
cap6000	B6	1703	6.21e-04%	3600
cap6000	B7	1882	7.00e-04%	3600
danoint	B1	552	4.19%	3600
danoint	B2	473	4.14%	3600
danoint	B3	631	4.16%	3600
danoint	B4	572	4.12%	3600
danoint	B5	566	4.12%	3600
danoint	B6	592	4.14%	3600
danoint	B7	611	4.17%	3600
dcmulti	B1	69380	0.00841%	3600
dcmulti	B2	29635	0%	1155
dcmulti	B3	3989	0%	132
dcmulti	B4	897	0%	28
dcmulti	B5	943	0%	32
dcmulti	B6	973	0%	33
dcmulti	B7	3877	0%	156
dsbmip	B1	1	0%	2
dsbmip	B2	1	0%	2
dsbmip	B3	1	0%	2
dsbmip	B4	1	0%	2

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
dsbmip	B5	1	0%	2
dsbmip	B6	1	0%	2
dsbmip	B7	1	0%	2
egout	B1	3	0%	0
egout	B2	3	0%	0
egout	B3	3	0%	0
egout	B4	3	0%	0
egout	B5	3	0%	0
egout	B6	3	0%	0
egout	B7	3	0%	0
enigma	B1	1	0%	0
enigma	B2	1	0%	0
enigma	B3	1	0%	0
enigma	B4	1	0%	0
enigma	B5	1	0%	0
enigma	B6	1	0%	0
enigma	B7	1	0%	0
fast0507	B1	173	1.01%	3600
fast0507	B2	167	0.991%	3600
fast0507	B3	57	0.916%	3600
fast0507	B4	72	0.935%	3600
fast0507	B5	37	0.917%	3600
fast0507	B6	37	0.917%	3600
fast0507	B7	49	0.926%	3600
fiber	B1	97	0%	10
fiber	B2	213	0%	17
fiber	B3	39	0%	7
fiber	B4	27	0%	7
fiber	B5	27	0%	8
fiber	B6	27	0%	8
fiber	B7	61	0%	10
fixnet6	B1	1005	0%	80
fixnet6	B2	411	0%	32
fixnet6	B3	79	0%	9
fixnet6	B4	65	0%	10
fixnet6	B5	63	0%	10
fixnet6	B6	63	0%	10
fixnet6	B7	77	0%	9
flugpl	B1	10829	0%	28
flugpl	B2	5331	0%	10
flugpl	B3	3745	0%	7

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
flugpl	B4	8587	0%	13
flugpl	B5	4685	0%	8
flugpl	B6	4485	0%	8
flugpl	B7	3033	0%	6
gen	B1	19	0%	2
gen	B2	3	0%	1
gen	B3	3	0%	1
gen	B4	3	0%	1
gen	B5	3	0%	1
gen	B6	3	0%	1
gen	B7	3	0%	1
gesa2	B1	28843	0.302%	3600
gesa2	B2	28908	0.326%	3600
gesa2	B3	27398	0.013%	3600
gesa2	B4	41160	0.00726%	3600
gesa2	B5	33960	0.00385%	3600
gesa2	B6	33024	0.00505%	3600
gesa2	B7	28324	0.00608%	3600
gesa2_o	B1	28470	0.378%	3600
gesa2_o	B2	28458	0.384%	3600
gesa2_o	B3	26309	0.0145%	3600
gesa2_o	B4	47870	0.00253%	3600
gesa2_o	B5	33398	0.00334%	3600
gesa2_o	B6	34126	0.00208%	3600
gesa2_o	B7	22062	0.0172%	3600
gesa3	B1	19974	0.0589%	3600
gesa3	B2	22285	0%	3460
gesa3	B3	893	0%	148
gesa3	B4	811	0%	86
gesa3	B5	723	0%	99
gesa3	B6	847	0%	113
gesa3	B7	1053	0%	186
gesa3_o	B1	22500	0.0798%	3600
gesa3_o	B2	21699	0.118%	3600
gesa3_o	B3	825	0%	115
gesa3_o	B4	987	0%	92
gesa3_o	B5	889	0%	127
gesa3_o	B6	981	0%	141
gesa3_o	B7	1121	0%	166
gt2	B1	85091	34%	3600
gt2	B2	86525	34%	3600

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
gt2	B3	132763	6.69%	3600
gt2	B4	57	0%	0
gt2	B5	53	0%	0
gt2	B6	53	0%	0
gt2	B7	74618	35.6%	3600
harp2	B1	7837	2.96e-01%	3600
harp2	B2	9814	3.60e-01%	3600
harp2	B3	3949	6.63e-02%	3600
harp2	B4	12058	7.55e-06%	3600
harp2	B5	9149	7.45e-06%	3600
harp2	B6	4778	1.93e-02%	3600
harp2	B7	2540	1.44e-01%	3600
khb05250	B1	23	0%	2
khb05250	B2	15	0%	1
khb05250	B3	17	0%	2
khb05250	B4	15	0%	2
khb05250	B5	15	0%	2
khb05250	B6	15	0%	2
khb05250	B7	17	0%	2
l152lav	B1	8337	0%	552
l152lav	B2	4131	0%	286
l152lav	B3	459	0%	84
l152lav	B4	469	0%	71
l152lav	B5	313	0%	72
l152lav	B6	259	0%	67
l152lav	B7	5203	0%	717
lseu	B1	59	0%	1
lseu	B2	99	0%	1
lseu	B3	85	0%	2
lseu	B4	91	0%	2
lseu	B5	83	0%	2
lseu	B6	89	0%	2
lseu	B7	109	0%	2
misc03	B1	439	0%	9
misc03	B2	363	0%	8
misc03	B3	547	0%	14
misc03	B4	483	0%	12
misc03	B5	405	0%	12
misc03	B6	395	0%	12
misc03	B7	491	0%	13
misc06	B1	423	0%	22

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
misc06	B2	423	0%	22
misc06	B3	165	0%	12
misc06	B4	59	0%	5
misc06	B5	59	0%	6
misc06	B6	59	0%	6
misc06	B7	165	0%	12
misc07	B1	13321	0%	980
misc07	B2	10249	0%	600
misc07	B3	18321	0%	1427
misc07	B4	51837	0%	3490
misc07	B5	43697	0%	3376
misc07	B6	38789	0%	3063
misc07	B7	17127	0%	1441
mitre	B1	23	0%	59
mitre	B2	23	0%	59
mitre	B3	23	0%	106
mitre	B4	23	0%	162
mitre	B5	23	0%	210
mitre	B6	23	0%	210
mitre	B7	23	0%	126
mod008	B1	437	0%	32
mod008	B2	579	0%	46
mod008	B3	431	0%	25
mod008	B4	91	0%	5
mod008	B5	79	0%	5
mod008	B6	79	0%	5
mod008	B7	501	0%	30
mod010	B1	31	0%	11
mod010	B2	15	0%	9
mod010	B3	131	0%	24
mod010	B4	19	0%	11
mod010	B5	39	0%	14
mod010	B6	39	0%	14
mod010	B7	1005	0%	97
mod011	B1	466	4.3%	3600
mod011	B2	448	5.19%	3600
mod011	B3	395	6.16%	3600
mod011	B4	414	4.73%	3600
mod011	B5	398	4.38%	3600
mod011	B6	413	4.24%	3601
mod011	B7	407	6.07%	3600

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
modglob	B1	275	0%	18
modglob	B2	823	0%	54
modglob	B3	1329	0%	162
modglob	B4	477	0%	39
modglob	B5	1015	0%	112
modglob	B6	429	0%	38
modglob	B7	1993	0%	135
noswot	B1	1	0%	0
noswot	B2	1	0%	0
noswot	B3	1	0%	0
noswot	B4	1	0%	0
noswot	B5	1	0%	0
noswot	B6	1	0%	0
noswot	B7	1	0%	0
p0033	B1	7	0%	0
p0033	B2	11	0%	0
p0033	B3	11	0%	0
p0033	B4	7	0%	0
p0033	B5	7	0%	0
p0033	B6	7	0%	0
p0033	B7	11	0%	0
p0201	B1	545	0%	17
p0201	B2	689	0%	18
p0201	B3	469	0%	23
p0201	B4	77	0%	6
p0201	B5	63	0%	6
p0201	B6	63	0%	6
p0201	B7	333	0%	19
p0282	B1	497	0%	32
p0282	B2	25	0%	3
p0282	B3	61	0%	5
p0282	B4	37	0%	3
p0282	B5	37	0%	3
p0282	B6	37	0%	3
p0282	B7	73	0%	5
p0548	B1	5	0%	2
p0548	B2	5	0%	2
p0548	B3	5	0%	2
p0548	B4	9	0%	2
p0548	B5	9	0%	2
p0548	B6	9	0%	2

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
p0548	B7	9	0%	2
p2756	B1	3883	0%	2035
p2756	B2	11	0%	5
p2756	B3	11	0%	5
p2756	B4	13	0%	7
p2756	B5	13	0%	7
p2756	B6	13	0%	7
p2756	B7	11	0%	6
pk1	B1	44838	50.3%	3600
pk1	B2	44186	51.1%	3600
pk1	B3	44021	49.3%	3600
pk1	B4	70055	37.4%	3600
pk1	B5	64587	34.4%	3600
pk1	B6	65074	34.3%	3600
pk1	B7	43309	49.2%	3600
pp08a	B1	50284	12.2%	3600
pp08a	B2	50882	10.1%	3600
pp08a	B3	44925	8.76%	3600
pp08a	B4	106531	4.51%	3600
pp08a	B5	71654	5.53%	3600
pp08a	B6	73753	5.07%	3600
pp08a	B7	43490	8.83%	3600
pp08aCUTS	B1	48143	12.3%	3600
pp08aCUTS	B2	48454	10.2%	3600
pp08aCUTS	B3	41498	8.82%	3600
pp08aCUTS	B4	89825	4.77%	3600
pp08aCUTS	B5	60842	5.74%	3600
pp08aCUTS	B6	62318	5.37%	3600
pp08aCUTS	B7	40205	8.79%	3600
qiu	B1	3904	177%	3600
qiu	B2	3550	139%	3600
qiu	B3	3874	191%	3600
qiu	B4	4102	109%	3600
qiu	B5	3785	110%	3600
qiu	B6	3706	94.4%	3600
qiu	B7	3909	190%	3600
qnet1	B1	28679	7.55%	3600
qnet1	B2	34584	8.68%	3600
qnet1	B3	287	0%	41
qnet1	B4	81	0%	25
qnet1	B5	83	0%	28

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
qnet1	B6	81	0%	28
qnet1	B7	219	0%	41
qnet1_o	B1	28299	0%	1768
qnet1_o	B2	37407	0%	2327
qnet1_o	B3	559	0%	59
qnet1_o	B4	249	0%	25
qnet1_o	B5	233	0%	24
qnet1_o	B6	237	0%	24
qnet1_o	B7	391	0%	40
rentacar	B1	17	0%	33
rentacar	B2	25	0%	39
rentacar	B3	16	0%	29
rentacar	B4	17	0%	38
rentacar	B5	17	0%	39
rentacar	B6	17	0%	39
rentacar	B7	16	0%	29
rgn	B1	3085	0%	41
rgn	B2	1855	0%	19
rgn	B3	2039	0%	24
rgn	B4	2745	0%	30
rgn	B5	2137	0%	25
rgn	B6	2107	0%	25
rgn	B7	2053	0%	24
rout	B1	4190	6.16%	3600
rout	B2	3096	4.22%	3600
rout	B3	3770	5.4%	3600
rout	B4	3704	3.58%	3600
rout	B5	2993	3.48%	3600
rout	B6	2838	3.64%	3600
rout	B7	3344	5.42%	3600
set1ch	B1	18934	24.6%	3600
set1ch	B2	19335	24.4%	3600
set1ch	B3	18629	16.1%	3600
set1ch	B4	65750	13.5%	3600
set1ch	B5	21844	15.7%	3600
set1ch	B6	20035	15%	3600
set1ch	B7	14958	16.3%	3600
seymour	B1	3023	3.96%	3600
seymour	B2	2675	4.12%	3600
seymour	B3	492	3.72%	3600
seymour	B4	111	3.45%	3600

Problem	Branch Method	Nodes	Final Gap	Sol. Time(sec.)
seymour	B5	71	3.53%	3600
seymour	B6	71	3.54%	3600
seymour	B7	524	3.78%	3600
stein27	B1	4541	0%	46
stein27	B2	4777	0%	40
stein27	B3	3317	0%	41
stein27	B4	4179	0%	34
stein27	B5	3599	0%	39
stein27	B6	3399	0%	40
stein27	B7	3371	0%	40
stein45	B1	57636	4.44%	3600
stein45	B2	64273	5%	3600
stein45	B3	52815	0%	3445
stein45	B4	60871	0%	2326
stein45	B5	47241	0%	2392
stein45	B6	47809	0%	2701
stein45	B7	53579	0%	3594
vpm1	B1	335	0%	4
vpm1	B2	541	0%	6
vpm1	B3	119	0%	2
vpm1	B4	41	0%	1
vpm1	B5	29	0%	1
vpm1	B6	29	0%	0
vpm1	B7	121	0%	2
vpm2	B1	16597	0%	552
vpm2	B2	12687	0%	394
vpm2	B3	11539	0%	351
vpm2	B4	8693	0%	201
vpm2	B5	7143	0%	183
vpm2	B6	7615	0%	194
vpm2	B7	13207	0%	423

Appendix B – Node Selection Tables

Table B.1: Comparison of Estimation Methods

Problem	Node	Best Projection	Pseudocost	Adjusted Pseudocost	True Optimum
bell3a	100	-878719	-875922	-875931	-880401
bell3a	200	-878440	-876149	-876203	-878430
bell3a	300	-878179	-875398	-875416	-879861
bell3a	400	-878225	-875407	-875466	-880637
bell3a	600	-880424	-877676	-877828	-881583
bell5	100	-9238936	-8964061	-8971500	-8988042
bell5	200	-9229115	-8960409	-8965001	-8994578
bell5	300	-9267804	-9012005	-9020515	-9068313
bell5	400	-9185050	-8955816	-8981035	-9004336
bell5	500	-9100616	-8957514	-8986731	-8998422
bell5	600	-9125318	-8959881	-8969655	-8999932
bell5	700	-9097684	-8956851	-8966321	-9000459
bell5	800	-9016859	-8958806	-8962628	-8968500
bell5	900	-9112098	-8979972	-8981002	-8997895
blend2	100	-7.325	-7.080	-7.089	-8.031
blend2	200	-7.927	-7.632	-7.670	-8.613
blend2	300	-7.885	-7.749	-7.777	-8.213
blend2	400	-7.583	-7.335	-7.378	-7.938
blend2	500	-8.062	-7.818	-7.945	-8.476
blend2	600	-7.492	-7.356	-7.434	-7.938
blend2	700	-8.240	-7.454	-7.565	-8.104
blend2	800	-8.023	-7.845	-7.904	-7.971
blend2	900	-7.869	-7.601	-7.681	-8.101
blend2	1000	-8.086	-7.709	-7.785	-8.431
dcmulti	100	-189562	-189376	-189378	-189376
dcmulti	200	-189495	-189380	-189381	-189439
dcmulti	300	-189784	-189355	-189362	-189356
dcmulti	400	-189380	-189342	-189343	-189460
dcmulti	500	-190134	-189315	-189356	-189306
dcmulti	600	-189315	-189315	-189315	-189566
dcmulti	700	-189527	-189371	-189373	-189368
dcmulti	800	-190214	-189488	-189504	-189500
dcmulti	900	-189987	-189439	-189447	-189434
dcmulti	1000	-190076	-189455	-189510	-189461
gesa2	100	-26017524	-25675289	-25800413	-25789927
gesa2	200	-26022342	-25730043	-25810203	-25802346
gesa2	300	-25925099	-25764748	-25889132	-25800290

Problem	Node	Best Projection	Pseudocost	Adjusted Pseudocost	True Optimum
gesa2	400	-25897281	-25765513	-25803062	-25822553
gesa2	500	-25824644	-25775169	-25783165	-25785232
gesa2	600	-25896747	-25775996	-25797620	-25797985
gesa2	700	-25917635	-25796416	-25840070	-25801872
gesa2	800	-25936207	-25789201	-25818264	-25811843
gesa2	900	-25887163	-25773484	-25797243	-25798712
gesa2	1000	-25882166	-25779565	-25796477	-25802292
gesa3_o	100	-28001002	-28021313	-28025779	-28009487
gesa3_o	200	-27996719	-28018387	-28026756	-28007721
gesa3_o	300	-27994832	-28029647	-28063520	-27996090
gesa3_o	400	-28001392	-28007818	-28014242	-28007895
gesa3_o	500	-28034124	-28070988	-28125782	-28018408
gesa3_o	600	-28022357	-28055442	-28101118	-28016848
gesa3_o	800	-28104094	-28196811	-28436973	-28097912
gesa3_o	900	-27998004	-28057821	-28294219	-27991042
l152lav	200	-4734.942	-4767.509	-4869.609	-4722.000
l152lav	300	-4754.593	-4753.697	-4845.671	-4768.000
l152lav	400	-4772.125	-4755.607	-4841.122	-4743.000
l152lav	900	-4772.506	-4741.417	-4804.221	-4751.000
misc07	100	-3953.125	-3140.982	-3821.228	-3085.000
misc07	200	-3032.143	-3032.143	-3032.143	-3250.000
misc07	300	-3903.125	-3037.010	-3539.726	-3245.000
misc07	400	-3884.643	-2692.007	-2918.734	-3475.000
misc07	500	-3062.708	-2079.030	-2475.589	-2865.000
mod008	10	-309.056	-304.843	-306.856	-307.000
mod008	20	-310.239	-297.691	-298.763	-307.000
mod008	30	-310.972	-304.893	-306.572	-307.000
mod008	40	-311.207	-308.091	-309.827	-307.000
mod008	50	-313.889	-300.209	-302.141	-307.000
mod008	60	-324.805	-307.317	-308.776	-307.000
mod008	70	-313.181	-306.573	-307.539	-323.000
mod008	80	-306.571	-306.571	-306.571	-346.000
mod008	90	-314.995	-311.049	-319.783	-342.000
mod008	100	-307.198	-306.568	-306.671	-332.000
p0201	10	-7665.000	-7665.000	-7665.000	-7735.000
p0201	20	-7747.000	-8239.938	-9213.731	-7735.000
p0201	30	-7649.091	-8018.188	-9761.640	-7615.000
p0201	50	-7891.957	-8049.682	-8847.175	-7805.000
p0201	60	-7696.364	-8068.026	-8788.989	-7615.000
p0201	70	-7733.788	-8256.728	-8760.803	-7675.000

Problem	Node	Best Projection	Pseudocost	Adjusted Pseudocost	True Optimum
p0201	80	-7712.977	-7835.500	-8119.564	-7795.000
p0201	90	-7734.773	-8244.848	-8665.251	-7665.000
pk1	100	-23.110	-0.193	-0.632	-17.000
pk1	200	-22.840	-0.704	-2.206	-15.000
pk1	300	-20.300	-2.626	-3.086	-21.000
pk1	400	-21.491	-0.426	-2.026	-12.000
pk1	500	-16.000	-0.133	-0.737	-20.000
pk1	600	-18.781	-0.835	-2.778	-18.000
pk1	700	-24.142	-1.329	-2.105	-17.000
pk1	800	-19.684	-1.372	-1.984	-17.000
pk1	900	-15.107	-0.752	-3.064	-17.000
pk1	1000	-21.869	-7.265	-7.393	-23.000
qiu	100	-29.775	279.861	-17.040	128.467
qiu	200	101.816	130.479	121.296	32.058
qiu	300	47.681	111.033	91.419	27.652
qiu	400	127.145	127.145	127.145	27.652
qiu	500	98.405	116.255	116.255	110.842
qiu	600	108.652	108.652	108.652	-0.000
qiu	700	45.176	121.674	63.336	40.870
qiu	800	-21.472	76.131	42.422	14.433
qiu	900	30.811	113.749	76.508	36.464
qiu	1000	77.237	93.212	70.041	-63.335
rgn	100	-82.451	-70.823	-71.730	-82.200
rgn	200	-81.128	-74.470	-75.284	-82.200
rgn	300	-82.236	-79.232	-79.523	-82.200
rgn	400	-79.027	-75.394	-76.602	-82.200
rgn	500	-81.736	-77.903	-80.605	-82.200
stein45	100	-32.257	-23.278	-24.364	-30.000
stein45	200	-31.714	-24.000	-24.000	-31.000
stein45	300	-31.790	-24.601	-25.357	-31.000
stein45	400	-32.819	-24.856	-25.408	-31.000
stein45	500	-31.790	-24.333	-24.333	-31.000
stein45	600	-31.790	-24.333	-24.333	-31.000
stein45	700	-31.867	-24.667	-24.667	-31.000
stein45	800	-36.800	-26.976	-27.502	-31.000
stein45	900	-35.486	-27.000	-27.000	-31.000
stein45	1000	-33.448	-25.621	-26.905	-31.000
vpm2	100	-14.082	-13.326	-13.364	-13.750
vpm2	200	-14.024	-13.494	-13.599	-14.500
vpm2	300	-14.081	-13.303	-13.412	-13.750

Problem	Node	Best Projection	Pseudocost	Adjusted Pseudocost	True Optimum
vpm2	400	-14.016	-13.538	-13.622	-14.000
vpm2	500	-13.990	-13.402	-13.569	-14.250
vpm2	600	-13.986	-13.554	-13.727	-14.500
vpm2	700	-14.085	-13.468	-13.677	-14.500
vpm2	800	-14.548	-14.062	-14.216	-14.250
vpm2	900	-14.528	-14.079	-14.173	-14.750
vpm2	1000	-14.161	-13.637	-13.791	-14.000

Table B.2: Comparison of Node Selection Methods

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
10teams	N1	450	1071	0%	0%
10teams	N2	3311	3600	(N/A)	(N/A)
10teams	N3	3315	3600	(N/A)	(N/A)
10teams	N4	240	612	0%	0%
10teams	N5	7254	3600	0%	0.758%
10teams	N6	240	611	0%	0%
10teams	N7	4570	3600	0.649%	1.398%
10teams	N8	4501	3600	(N/A)	(N/A)
10teams	N9	290	677	0%	0%
10teams	N10	490	1011	0%	0%
10teams	N11	6636	3600	(N/A)	(N/A)
10teams	N12	261	482	0%	0%
10teams	N13	652	622	0%	0%
air03	N1	3	62	0%	0%
air03	N2	3	62	0%	0%
air03	N3	3	62	0%	0%
air03	N4	3	62	0%	0%
air03	N5	3	62	0%	0%
air03	N6	3	62	0%	0%
air03	N7	3	62	0%	0%
air03	N8	3	63	0%	0%
air03	N9	3	62	0%	0%
air03	N10	3	62	0%	0%
air03	N11	3	62	0%	0%
air03	N12	3	62	0%	0%
air03	N13	3	62	0%	0%
air04	N1	701	3475	0%	0%
air04	N2	1505	3355	0%	0%
air04	N3	854	3600	0.0196%	0.116%
air04	N4	683	3600	0%	0.740%
air04	N5	845	2915	0%	0%
air04	N6	684	3600	0%	0.740%
air04	N7	891	3601	0.00178%	0.742%
air04	N8	1449	3601	0%	0.988%
air04	N9	862	3600	0.00178%	0.232%
air04	N10	499	3605	0.203%	0.554%
air04	N11	1361	3033	0%	0%
air04	N12	901	3600	0%	0.902%
air04	N13	825	3158	0%	0%
air05	N1	520	3600	3.58%	4.031%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
air05	N2	2193	2778	0%	0%
air05	N3	843	3600	0%	0.227%
air05	N4	1154	2728	0%	0%
air05	N5	2030	3491	0%	0%
air05	N6	1284	2689	0%	0%
air05	N7	1390	2962	0%	0%
air05	N8	2461	3002	0%	0%
air05	N9	1501	2730	0%	0%
air05	N10	2193	2780	0%	0%
air05	N11	2463	2966	0%	0%
air05	N12	2337	3040	0%	0%
air05	N13	2193	2772	0%	0%
bell3a	N1	38421	153	0%	0%
bell3a	N2	23755	102	0%	0%
bell3a	N3	38421	153	0%	0%
bell3a	N4	173626	2675	0%	0%
bell3a	N5	34677	142	0%	0%
bell3a	N6	173626	742	0%	0%
bell3a	N7	148294	641	0%	0%
bell3a	N8	20766	107	0%	0%
bell3a	N9	39292	360	0%	0%
bell3a	N10	38712	288	0%	0%
bell3a	N11	27673	126	0%	0%
bell3a	N12	42306	356	0%	0%
bell3a	N13	42297	336	0%	0%
bell5	N1	23683	90	0%	0%
bell5	N2	1098601	3600	0.846%	1.502%
bell5	N3	23683	90	0%	0%
bell5	N4	149494	3600	0.109%	4.097%
bell5	N5	981577	3600	0.738%	1.396%
bell5	N6	635000	XXX	0.109%	4.097%
bell5	N7	553000	XXX	0.189%	0.856%
bell5	N8	1092553	3600	0.283%	0.681%
bell5	N9	40723	339	0%	0%
bell5	N10	153075	3600	0%	0.019%
bell5	N11	1103324	3600	0.846%	1.502%
bell5	N12	27791	122	0%	0%
bell5	N13	125064	3600	0.022%	0.691%
blend2	N1	2365	292	0%	0%
blend2	N2	31424	1674	0%	0%
blend2	N3	2932	324	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
blend2	N4	9788	592	0%	0%
blend2	N5	4127	203	0%	0%
blend2	N6	9788	578	0%	0%
blend2	N7	3509	161	0%	0%
blend2	N8	1788	104	0%	0%
blend2	N9	4041	441	0%	0%
blend2	N10	3872	296	0%	0%
blend2	N11	2640	150	0%	0%
blend2	N12	5732	278	0%	0%
blend2	N13	3467	149	0%	0%
cap6000	N1	4099	1851	0%	0%
cap6000	N2	21171	3301	0%	0%
cap6000	N3	4099	1851	0%	0%
cap6000	N4	8884	2372	0%	0%
cap6000	N5	7423	3600	0.000245%	0.004%
cap6000	N6	8884	2328	0%	0%
cap6000	N7	8758	2294	0%	0%
cap6000	N8	16116	3600	0.00322%	0.004%
cap6000	N9	6860	3284	0%	0%
cap6000	N10	11403	3077	0%	0%
cap6000	N11	19485	3600	0.00216%	0.009%
cap6000	N12	15411	3600	0%	0.002%
cap6000	N13	14751	3067	0%	0%
danoint	N1	561	3600	1.264%	5.330%
danoint	N2	4521	3600	1.264%	5.784%
danoint	N3	561	3600	1.264%	5.330%
danoint	N4	477	3600	1.264%	5.729%
danoint	N5	845	3600	1.264%	5.729%
danoint	N6	477	3600	1.264%	5.729%
danoint	N7	1058	3600	1.264%	5.729%
danoint	N8	2223	3600	0%	4.323%
danoint	N9	1656	3600	0%	4.191%
danoint	N10	1798	3600	0.918%	5.123%
danoint	N11	4964	3600	1.264%	5.784%
danoint	N12	1923	3600	0%	4.407%
danoint	N13	3704	3600	0%	4.533%
dcmulti	N1	967	38	0%	0%
dcmulti	N2	5563	120	0%	0%
dcmulti	N3	967	38	0%	0%
dcmulti	N4	3906	97	0%	0%
dcmulti	N5	2081	51	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
dcmulti	N6	3906	96	0%	0%
dcmulti	N7	3659	85	0%	0%
dcmulti	N8	3259	77	0%	0%
dcmulti	N9	1176	36	0%	0%
dcmulti	N10	5283	117	0%	0%
dcmulti	N11	3365	79	0%	0%
dcmulti	N12	4364	102	0%	0%
dcmulti	N13	4397	103	0%	0%
dsbmip	N1	150	55	0%	0%
dsbmip	N2	1290	219	0%	0%
dsbmip	N3	579	156	0%	0%
dsbmip	N4	40	25	0%	0%
dsbmip	N5	151	50	0%	0%
dsbmip	N6	40	25	0%	0%
dsbmip	N7	129	55	0%	0%
dsbmip	N8	186	72	0%	0%
dsbmip	N9	150	69	0%	0%
dsbmip	N10	321	118	0%	0%
dsbmip	N11	220	76	0%	0%
dsbmip	N12	104	50	0%	0%
dsbmip	N13	293	95	0%	0%
egout	N1	3	0	0%	0%
egout	N2	3	0	0%	0%
egout	N3	3	0	0%	0%
egout	N4	3	0	0%	0%
egout	N5	3	0	0%	0%
egout	N6	3	0	0%	0%
egout	N7	3	0	0%	0%
egout	N8	3	0	0%	0%
egout	N9	3	0	0%	0%
egout	N10	3	0	0%	0%
egout	N11	3	0	0%	0%
egout	N12	3	0	0%	0%
egout	N13	3	0	0%	0%
enigma	N1	6887	112	0%	0%
enigma	N2	4657	33	0%	0%
enigma	N3	8432	65	0%	0%
enigma	N4	518	11	0%	0%
enigma	N5	4491	79	0%	0%
enigma	N6	518	11	0%	0%
enigma	N7	2480	42	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
enigma	N8	2256	39	0%	0%
enigma	N9	1080	23	0%	0%
enigma	N10	949	22	0%	0%
enigma	N11	2256	39	0%	0%
enigma	N12	15433	158	0%	0%
enigma	N13	3638	56	0%	0%
fast0507	N1	32	3600	5.747%	6.313%
fast0507	N2	54	3600	6.322%	6.948%
fast0507	N3	32	3600	5.747%	6.313%
fast0507	N4	33	3600	10.345%	10.220%
fast0507	N5	32	3600	2.874%	3.733%
fast0507	N6	33	3600	10.345%	10.220%
fast0507	N7	33	3600	9.770%	9.750%
fast0507	N8	55	3600	4.598%	5.415%
fast0507	N9	40	3600	10.345%	10.214%
fast0507	N10	37	3600	10.345%	10.219%
fast0507	N11	54	3600	4.598%	5.415%
fast0507	N12	39	3600	7.471%	7.887%
fast0507	N13	40	3600	5.172%	5.887%
fiber	N1	35	13	0%	0%
fiber	N2	105	24	0%	0%
fiber	N3	35	13	0%	0%
fiber	N4	58	18	0%	0%
fiber	N5	45	13	0%	0%
fiber	N6	58	17	0%	0%
fiber	N7	79	20	0%	0%
fiber	N8	95	25	0%	0%
fiber	N9	105	24	0%	0%
fiber	N10	105	24	0%	0%
fiber	N11	95	25	0%	0%
fiber	N12	95	25	0%	0%
fiber	N13	95	26	0%	0%
fixnet6	N1	79	15	0%	0%
fixnet6	N2	71	10	0%	0%
fixnet6	N3	79	15	0%	0%
fixnet6	N4	71	13	0%	0%
fixnet6	N5	91	14	0%	0%
fixnet6	N6	71	13	0%	0%
fixnet6	N7	70	13	0%	0%
fixnet6	N8	79	16	0%	0%
fixnet6	N9	86	15	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
fixnet6	N10	71	13	0%	0%
fixnet6	N11	76	13	0%	0%
fixnet6	N12	75	11	0%	0%
fixnet6	N13	75	12	0%	0%
flugpl	N1	4305	8	0%	0%
flugpl	N2	4091	5	0%	0%
flugpl	N3	4167	7	0%	0%
flugpl	N4	3551	6	0%	0%
flugpl	N5	2675	4	0%	0%
flugpl	N6	3551	5	0%	0%
flugpl	N7	3426	5	0%	0%
flugpl	N8	4112	7	0%	0%
flugpl	N9	3511	7	0%	0%
flugpl	N10	3665	7	0%	0%
flugpl	N11	6290	10	0%	0%
flugpl	N12	3846	6	0%	0%
flugpl	N13	3580	6	0%	0%
gen	N1	3	2	0%	0%
gen	N2	3	2	0%	0%
gen	N3	3	2	0%	0%
gen	N4	3	2	0%	0%
gen	N5	3	2	0%	0%
gen	N6	3	2	0%	0%
gen	N7	3	2	0%	0%
gen	N8	3	2	0%	0%
gen	N9	3	2	0%	0%
gen	N10	3	2	0%	0%
gen	N11	3	2	0%	0%
gen	N12	3	2	0%	0%
gen	N13	3	2	0%	0%
gesa2	N1	23169	3600	0.316%	0.329%
gesa2	N2	50309	3600	0.489%	1.331%
gesa2	N3	23146	3600	0.316%	0.329%
gesa2	N4	38871	3600	0%	0.559%
gesa2	N5	46188	3600	0%	0.538%
gesa2	N6	40983	3600	0%	0.559%
gesa2	N7	43730	3600	0.079%	0.927%
gesa2	N8	46169	3600	0%	0.205%
gesa2	N9	21116	3600	0.316%	0.331%
gesa2	N10	23062	3600	0.306%	0.329%
gesa2	N11	46842	3600	0.508%	1.351%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
gesa2	N12	45851	3600	0.012%	0.548%
gesa2	N13	46343	3600	0%	0.849%
gesa2_o	N1	22388	3600	0.220%	0.229%
gesa2_o	N2	49978	3600	0.002%	0.950%
gesa2_o	N3	22411	3600	0.220%	0.229%
gesa2_o	N4	40565	3600	0.009%	0.698%
gesa2_o	N5	40033	3195	0%	0%
gesa2_o	N6	41847	3600	0.009%	0.698%
gesa2_o	N7	45317	3600	0.008%	0.956%
gesa2_o	N8	45660	3600	0.089%	0.563%
gesa2_o	N9	21227	3600	0.101%	0.115%
gesa2_o	N10	27676	3600	0.014%	0.031%
gesa2_o	N11	46115	3600	0.013%	0.961%
gesa2_o	N12	42973	3600	0.011%	0.727%
gesa2_o	N13	45965	3600	0.011%	0.958%
gesa3	N1	839	142	0%	0%
gesa3	N2	4205	362	0%	0%
gesa3	N3	839	141	0%	0%
gesa3	N4	928	110	0%	0%
gesa3	N5	869	99	0%	0%
gesa3	N6	928	110	0%	0%
gesa3	N7	1031	114	0%	0%
gesa3	N8	803	122	0%	0%
gesa3	N9	726	123	0%	0%
gesa3	N10	4205	362	0%	0%
gesa3	N11	747	94	0%	0%
gesa3	N12	682	84	0%	0%
gesa3	N13	1754	170	0%	0%
gesa3_o	N1	971	142	0%	0%
gesa3_o	N2	1041	116	0%	0%
gesa3_o	N3	971	140	0%	0%
gesa3_o	N4	973	116	0%	0%
gesa3_o	N5	937	118	0%	0%
gesa3_o	N6	973	116	0%	0%
gesa3_o	N7	895	112	0%	0%
gesa3_o	N8	985	126	0%	0%
gesa3_o	N9	1041	117	0%	0%
gesa3_o	N10	1041	117	0%	0%
gesa3_o	N11	993	120	0%	0%
gesa3_o	N12	1039	122	0%	0%
gesa3_o	N13	1039	122	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
gt2	N1	318	3	0%	0%
gt2	N2	776554	3600	111.471%	69.928%
gt2	N3	262800	1281	0%	0%
gt2	N4	413	4	0%	0%
gt2	N5	667404	3600	63.229%	46.793%
gt2	N6	413	4	0%	0%
gt2	N7	110	1	0%	0%
gt2	N8	408	4	0%	0%
gt2	N9	202	3	0%	0%
gt2	N10	202	3	0%	0%
gt2	N11	604882	3600	45.332%	40.584%
gt2	N12	79	1	0%	0%
gt2	N13	71	1	0%	0%
harp2	N1	2550	3600	0.404%	0.453%
harp2	N2	21397	3600	0.116%	0.549%
harp2	N3	3755	3600	0.420%	0.424%
harp2	N4	12935	3600	0%	0.432%
harp2	N5	17146	3600	2.032%	2.386%
harp2	N6	13455	3600	0%	0.432%
harp2	N7	14240	3600	0.173%	0.606%
harp2	N8	2264	3600	0.407%	0.486%
harp2	N9	4164	3600	0.389%	0.458%
harp2	N10	4277	3600	0.270%	0.349%
harp2	N11	18466	3600	0.929%	1.317%
harp2	N12	9414	3600	0.023%	0.456%
harp2	N13	18958	3600	0%	0.432%
khb05250	N1	15	3	0%	0%
khb05250	N2	15	3	0%	0%
khb05250	N3	15	3	0%	0%
khb05250	N4	15	3	0%	0%
khb05250	N5	15	3	0%	0%
khb05250	N6	15	3	0%	0%
khb05250	N7	15	3	0%	0%
khb05250	N8	15	3	0%	0%
khb05250	N9	15	3	0%	0%
khb05250	N10	15	3	0%	0%
khb05250	N11	15	3	0%	0%
khb05250	N12	15	3	0%	0%
khb05250	N13	15	3	0%	0%
l152lav	N1	337	111	0%	0%
l152lav	N2	402	94	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
l152lav	N3	337	111	0%	0%
l152lav	N4	637	170	0%	0%
l152lav	N5	441	166	0%	0%
l152lav	N6	637	169	0%	0%
l152lav	N7	434	135	0%	0%
l152lav	N8	677	130	0%	0%
l152lav	N9	964	191	0%	0%
l152lav	N10	402	94	0%	0%
l152lav	N11	495	104	0%	0%
l152lav	N12	2836	423	0%	0%
l152lav	N13	810	146	0%	0%
lseu	N1	105	3	0%	0%
lseu	N2	105	2	0%	0%
lseu	N3	99	3	0%	0%
lseu	N4	115	3	0%	0%
lseu	N5	153	3	0%	0%
lseu	N6	115	2	0%	0%
lseu	N7	159	4	0%	0%
lseu	N8	139	3	0%	0%
lseu	N9	89	2	0%	0%
lseu	N10	105	2	0%	0%
lseu	N11	139	3	0%	0%
lseu	N12	79	2	0%	0%
lseu	N13	123	3	0%	0%
misc03	N1	371	12	0%	0%
misc03	N2	387	9	0%	0%
misc03	N3	371	12	0%	0%
misc03	N4	517	16	0%	0%
misc03	N5	437	13	0%	0%
misc03	N6	517	16	0%	0%
misc03	N7	385	11	0%	0%
misc03	N8	437	10	0%	0%
misc03	N9	401	12	0%	0%
misc03	N10	387	9	0%	0%
misc03	N11	437	10	0%	0%
misc03	N12	483	12	0%	0%
misc03	N13	437	10	0%	0%
misc06	N1	53	9	0%	0%
misc06	N2	298	17	0%	0%
misc06	N3	53	9	0%	0%
misc06	N4	60	7	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
misc06	N5	61	7	0%	0%
misc06	N6	60	7	0%	0%
misc06	N7	61	7	0%	0%
misc06	N8	96	10	0%	0%
misc06	N9	56	8	0%	0%
misc06	N10	73	10	0%	0%
misc06	N11	85	9	0%	0%
misc06	N12	58	7	0%	0%
misc06	N13	57	7	0%	0%
misc07	N1	39657	3449	0%	0%
misc07	N2	46249	1693	0%	0%
misc07	N3	39657	3448	0%	0%
misc07	N4	49055	3266	0%	0%
misc07	N5	40861	2226	0%	0%
misc07	N6	49055	2846	0%	0%
misc07	N7	44669	2270	0%	0%
misc07	N8	51593	1980	0%	0%
misc07	N9	41715	2598	0%	0%
misc07	N10	26613	1175	0%	0%
misc07	N11	49010	1826	0%	0%
misc07	N12	43749	1899	0%	0%
misc07	N13	52561	1934	0%	0%
mitre	N1	38	221	0%	0%
mitre	N2	40	214	0%	0%
mitre	N3	36	209	0%	0%
mitre	N4	35	209	0%	0%
mitre	N5	36	207	0%	0%
mitre	N6	35	209	0%	0%
mitre	N7	35	209	0%	0%
mitre	N8	36	209	0%	0%
mitre	N9	49	224	0%	0%
mitre	N10	40	215	0%	0%
mitre	N11	36	208	0%	0%
mitre	N12	36	207	0%	0%
mitre	N13	36	209	0%	0%
mod008	N1	101	9	0%	0%
mod008	N2	105	8	0%	0%
mod008	N3	101	9	0%	0%
mod008	N4	105	8	0%	0%
mod008	N5	103	9	0%	0%
mod008	N6	105	8	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
mod008	N7	105	8	0%	0%
mod008	N8	99	9	0%	0%
mod008	N9	105	8	0%	0%
mod008	N10	105	8	0%	0%
mod008	N11	99	9	0%	0%
mod008	N12	99	9	0%	0%
mod008	N13	99	9	0%	0%
mod010	N1	17	26	0%	0%
mod010	N2	37	25	0%	0%
mod010	N3	17	26	0%	0%
mod010	N4	45	46	0%	0%
mod010	N5	15	25	0%	0%
mod010	N6	45	46	0%	0%
mod010	N7	43	36	0%	0%
mod010	N8	27	23	0%	0%
mod010	N9	37	25	0%	0%
mod010	N10	37	25	0%	0%
mod010	N11	27	23	0%	0%
mod010	N12	27	23	0%	0%
mod010	N13	27	23	0%	0%
mod011	N1	347	3600	0.666%	5.339%
mod011	N2	1594	3600	1.796%	14.880%
mod011	N3	346	3600	0.666%	5.342%
mod011	N4	337	3600	0.826%	8.226%
mod011	N5	1040	3600	0.826%	12.587%
mod011	N6	337	3600	0.826%	8.226%
mod011	N7	474	3600	0.916%	9.029%
mod011	N8	1426	3600	2.027%	15.152%
mod011	N9	422	3600	1.603%	6.304%
mod011	N10	651	3600	0.600%	6.867%
mod011	N11	1441	3600	2.027%	15.152%
mod011	N12	624	3600	1.181%	10.399%
mod011	N13	764	3600	0.666%	10.201%
modglob	N1	569	96	0%	0%
modglob	N2	99287	3600	2.15%	3.131%
modglob	N3	569	95	0%	0%
modglob	N4	357	30	0%	0%
modglob	N5	36624	3600	2.7%	3.524%
modglob	N6	357	30	0%	0%
modglob	N7	1089	76	0%	0%
modglob	N8	9838	2157	0.0615%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
modglob	N9	703	138	0%	0%
modglob	N10	1174	204	0%	0%
modglob	N11	67942	3600	0.283%	1.324%
modglob	N12	442	29	0%	0%
modglob	N13	867	65	0%	0%
noswot	N1	66336	3600	4.651%	4.878%
noswot	N2	287126	3600	4.651%	4.878%
noswot	N3	177425	3600	4.651%	3.371%
noswot	N4	116427	3600	4.651%	4.878%
noswot	N5	180312	3600	6.977%	7.500%
noswot	N6	189713	3600	4.651%	4.878%
noswot	N7	239744	3600	4.651%	4.878%
noswot	N8	60689	3600	6.977%	7.500%
noswot	N9	50852	3600	4.651%	4.878%
noswot	N10	54369	3600	4.651%	4.878%
noswot	N11	192088	3600	4.651%	4.878%
noswot	N12	63430	3600	4.651%	4.878%
noswot	N13	176142	3600	4.651%	4.878%
p0033	N1	7	0	0%	0%
p0033	N2	14	0	0%	0%
p0033	N3	7	0	0%	0%
p0033	N4	7	0	0%	0%
p0033	N5	12	0	0%	0%
p0033	N6	7	0	0%	0%
p0033	N7	7	0	0%	0%
p0033	N8	16	0	0%	0%
p0033	N9	14	0	0%	0%
p0033	N10	14	0	0%	0%
p0033	N11	16	0	0%	0%
p0033	N12	16	0	0%	0%
p0033	N13	16	0	0%	0%
p0201	N1	87	8	0%	0%
p0201	N2	127	8	0%	0%
p0201	N3	87	8	0%	0%
p0201	N4	95	8	0%	0%
p0201	N5	112	9	0%	0%
p0201	N6	95	8	0%	0%
p0201	N7	89	6	0%	0%
p0201	N8	105	7	0%	0%
p0201	N9	87	6	0%	0%
p0201	N10	127	8	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
p0201	N11	105	7	0%	0%
p0201	N12	105	7	0%	0%
p0201	N13	119	7	0%	0%
p0282	N1	33	5	0%	0%
p0282	N2	35	4	0%	0%
p0282	N3	33	5	0%	0%
p0282	N4	33	4	0%	0%
p0282	N5	51	5	0%	0%
p0282	N6	33	4	0%	0%
p0282	N7	31	5	0%	0%
p0282	N8	37	4	0%	0%
p0282	N9	35	4	0%	0%
p0282	N10	35	4	0%	0%
p0282	N11	37	5	0%	0%
p0282	N12	37	5	0%	0%
p0282	N13	37	5	0%	0%
p0548	N1	5	1	0%	0%
p0548	N2	50	4	0%	0%
p0548	N3	5	1	0%	0%
p0548	N4	9	1	0%	0%
p0548	N5	5	1	0%	0%
p0548	N6	9	1	0%	0%
p0548	N7	9	2	0%	0%
p0548	N8	50	4	0%	0%
p0548	N9	12	2	0%	0%
p0548	N10	16	2	0%	0%
p0548	N11	50	4	0%	0%
p0548	N12	12	2	0%	0%
p0548	N13	16	3	0%	0%
p2756	N1	39	34	0%	0%
p2756	N2	431	103	0%	0%
p2756	N3	39	34	0%	0%
p2756	N4	66	44	0%	0%
p2756	N5	369	106	0%	0%
p2756	N6	66	44	0%	0%
p2756	N7	146	49	0%	0%
p2756	N8	1306	374	0%	0%
p2756	N9	59	47	0%	0%
p2756	N10	251	87	0%	0%
p2756	N11	1297	343	0%	0%
p2756	N12	56	34	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
p2756	N13	223	58	0%	0%
pk1	N1	60336	3600	0%	34.923%
pk1	N2	99208	3600	36.364%	100%
pk1	N3	60344	3600	0%	34.921%
pk1	N4	40038	3600	0%	83.690%
pk1	N5	89867	3600	18.182%	100%
pk1	N6	52472	3600	0%	82.855%
pk1	N7	53292	3600	18.182%	100%
pk1	N8	67081	3600	0%	59.655%
pk1	N9	41121	3600	9.091%	46.882%
pk1	N10	45924	3600	0%	40.443%
pk1	N11	112867	3600	36.364%	100%
pk1	N12	118310	3600	0%	100%
pk1	N13	121653	3600	9.091%	100%
pp08a	N1	64136	3600	3.265%	8.274%
pp08a	N2	183600	3600	0.408%	25.737%
pp08a	N3	64147	3600	3.265%	8.274%
pp08a	N4	51687	3600	0.408%	14.066%
pp08a	N5	143848	3600	1.361%	25.186%
pp08a	N6	77151	3600	0.408%	14.066%
pp08a	N7	115009	3600	0%	19.628%
pp08a	N8	190906	3600	0.272%	25.636%
pp08a	N9	64694	3600	1.224%	8.466%
pp08a	N10	128733	3600	1.361%	11.909%
pp08a	N11	189001	3600	0.272%	25.636%
pp08a	N12	162219	3600	1.497%	19.968%
pp08a	N13	178327	3600	0%	20.722%
pp08aCUTS	N1	55400	3600	3.129%	8.314%
pp08aCUTS	N2	144271	3600	2.177%	27.023%
pp08aCUTS	N3	55407	3600	3.129%	8.313%
pp08aCUTS	N4	50147	3600	0.408%	13.686%
pp08aCUTS	N5	127428	3600	1.633%	25.386%
pp08aCUTS	N6	73080	3600	0%	13.221%
pp08aCUTS	N7	98183	3600	0%	19.628%
pp08aCUTS	N8	150225	3600	0.408%	25.737%
pp08aCUTS	N9	59422	3600	0.136%	8.033%
pp08aCUTS	N10	70251	3600	1.497%	10.197%
pp08aCUTS	N11	149317	3600	0.408%	25.737%
pp08aCUTS	N12	130325	3600	1.361%	18.032%
pp08aCUTS	N13	138787	3600	0.272%	21.350%
qiu	N1	3201	3600	0%	115.106%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
qiu	N2	19331	3600	151.747%	1123.264%
qiu	N3	3201	3600	0%	115.106%
qiu	N4	10163	3600	0%	379.090%
qiu	N5	11591	3600	0%	405.558%
qiu	N6	10177	3600	0%	379.090%
qiu	N7	11213	3481	0%	0%
qiu	N8	15339	3600	0%	258.406%
qiu	N9	4040	3600	0%	112.739%
qiu	N10	12235	2988	0%	0%
qiu	N11	19423	3600	151.747%	1123.264%
qiu	N12	15221	3600	0%	370.153%
qiu	N13	17682	3600	0%	379.090%
qnet1	N1	73	35	0%	0%
qnet1	N2	89	24	0%	0%
qnet1	N3	73	35	0%	0%
qnet1	N4	75	27	0%	0%
qnet1	N5	180	47	0%	0%
qnet1	N6	75	27	0%	0%
qnet1	N7	104	27	0%	0%
qnet1	N8	63	21	0%	0%
qnet1	N9	89	24	0%	0%
qnet1	N10	89	24	0%	0%
qnet1	N11	84	21	0%	0%
qnet1	N12	59	19	0%	0%
qnet1	N13	59	19	0%	0%
qnet1_o	N1	291	43	0%	0%
qnet1_o	N2	285	24	0%	0%
qnet1_o	N3	291	43	0%	0%
qnet1_o	N4	321	43	0%	0%
qnet1_o	N5	419	44	0%	0%
qnet1_o	N6	321	42	0%	0%
qnet1_o	N7	298	32	0%	0%
qnet1_o	N8	271	23	0%	0%
qnet1_o	N9	374	45	0%	0%
qnet1_o	N10	285	24	0%	0%
qnet1_o	N11	271	23	0%	0%
qnet1_o	N12	275	26	0%	0%
qnet1_o	N13	271	23	0%	0%
rentacar	N1	21	102	0%	0%
rentacar	N2	47	116	0%	0%
rentacar	N3	21	102	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
rentacar	N4	17	78	0%	0%
rentacar	N5	25	95	0%	0%
rentacar	N6	17	78	0%	0%
rentacar	N7	17	77	0%	0%
rentacar	N8	37	116	0%	0%
rentacar	N9	22	112	0%	0%
rentacar	N10	39	123	0%	0%
rentacar	N11	37	115	0%	0%
rentacar	N12	21	92	0%	0%
rentacar	N13	37	115	0%	0%
rgn	N1	2189	29	0%	0%
rgn	N2	2053	20	0%	0%
rgn	N3	2189	29	0%	0%
rgn	N4	2119	30	0%	0%
rgn	N5	1987	22	0%	0%
rgn	N6	2119	28	0%	0%
rgn	N7	2051	25	0%	0%
rgn	N8	2209	25	0%	0%
rgn	N9	2143	27	0%	0%
rgn	N10	2187	27	0%	0%
rgn	N11	2085	21	0%	0%
rgn	N12	2175	26	0%	0%
rgn	N13	2141	24	0%	0%
rout	N1	2940	3600	1.462%	5.555%
rout	N2	33620	3600	23.625%	26.294%
rout	N3	3406	3600	1.260%	5.851%
rout	N4	3502	3600	8.161%	14.751%
rout	N5	14503	3600	0%	8.852%
rout	N6	3504	3600	8.161%	14.751%
rout	N7	4776	3600	0%	7.794%
rout	N8	6338	3600	3.867%	8.383%
rout	N9	4446	3600	4.422%	8.420%
rout	N10	4652	3600	3.780%	7.695%
rout	N11	36215	3600	13.711%	19.867%
rout	N12	7248	3600	0.151%	7.153%
rout	N13	10068	3600	1.260%	8.116%
set1ch	N1	12311	3600	(N/A)	(N/A)
set1ch	N2	64360	3600	10.826%	32.728%
set1ch	N3	11997	3600	12.406%	24.865%
set1ch	N4	11861	3600	(N/A)	(N/A)
set1ch	N5	47331	3600	3.177%	21.117%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
set1ch	N6	11962	3600	(N/A)	(N/A)
set1ch	N7	11964	3600	5.140%	25.390%
set1ch	N8	54687	3600	7.606%	29.667%
set1ch	N9	12181	3600	8.245%	24.968%
set1ch	N10	12331	3600	7.130%	24.310%
set1ch	N11	54682	3600	7.606%	29.667%
set1ch	N12	12461	3600	7.065%	26.714%
set1ch	N13	13186	3600	6.789%	26.582%
seymour	N1	342	3600	(N/A)	(N/A)
seymour	N2	2956	3600	1.418%	5.863%
seymour	N3	435	3600	3.073%	6.698%
seymour	N4	306	3600	(N/A)	(N/A)
seymour	N5	2613	3600	3.546%	7.798%
seymour	N6	307	3600	(N/A)	(N/A)
seymour	N7	271	3600	(N/A)	(N/A)
seymour	N8	2917	3600	1.418%	5.863%
seymour	N9	300	3600	(N/A)	(N/A)
seymour	N10	333	3600	(N/A)	(N/A)
seymour	N11	2914	3600	1.418%	5.863%
seymour	N12	302	3600	(N/A)	(N/A)
seymour	N13	328	3600	(N/A)	(N/A)
stein27	N1	3389	42	0%	0%
stein27	N2	3361	21	0%	0%
stein27	N3	3389	42	0%	0%
stein27	N4	3341	36	0%	0%
stein27	N5	3345	29	0%	0%
stein27	N6	3341	34	0%	0%
stein27	N7	3185	29	0%	0%
stein27	N8	3269	21	0%	0%
stein27	N9	3351	25	0%	0%
stein27	N10	3325	21	0%	0%
stein27	N11	3269	20	0%	0%
stein27	N12	3519	25	0%	0%
stein27	N13	3253	21	0%	0%
stein45	N1	48569	2837	0%	0%
stein45	N2	51188	947	0%	0%
stein45	N3	48569	2840	0%	0%
stein45	N4	57461	1707	0%	0%
stein45	N5	48317	1312	0%	0%
stein45	N6	57461	1592	0%	0%
stein45	N7	49401	1068	0%	0%

Problem	Selection Method	Nodes	Time	Best Sol.	Final Gap
stein45	N8	49262	924	0%	0%
stein45	N9	49609	1019	0%	0%
stein45	N10	49663	1065	0%	0%
stein45	N11	49262	921	0%	0%
stein45	N12	48483	936	0%	0%
stein45	N13	49777	941	0%	0%
vpm1	N1	59	1	0%	0%
vpm1	N2	61	1	0%	0%
vpm1	N3	59	1	0%	0%
vpm1	N4	57	1	0%	0%
vpm1	N5	77	2	0%	0%
vpm1	N6	57	1	0%	0%
vpm1	N7	91	2	0%	0%
vpm1	N8	61	1	0%	0%
vpm1	N9	61	1	0%	0%
vpm1	N10	61	1	0%	0%
vpm1	N11	61	1	0%	0%
vpm1	N12	77	2	0%	0%
vpm1	N13	61	1	0%	0%
vpm2	N1	9453	278	0%	0%
vpm2	N2	17199	308	0%	0%
vpm2	N3	9453	278	0%	0%
vpm2	N4	7460	203	0%	0%
vpm2	N5	6923	152	0%	0%
vpm2	N6	7460	194	0%	0%
vpm2	N7	11931	312	0%	0%
vpm2	N8	11956	367	0%	0%
vpm2	N9	9622	285	0%	0%
vpm2	N10	10279	296	0%	0%
vpm2	N11	15417	296	0%	0%
vpm2	N12	8210	175	0%	0%
vpm2	N13	7909	156	0%	0%