Accepted for Publication in *Computers and Operations Research* (2010)

Faster Integer-Feasibility in Mixed-Integer Linear Programs by Branching to Force Change

Jennifer Pryor (jpryor@sce.carleton.ca)
John W. Chinneck (chinneck@sce.carleton.ca)

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

October 22, 2010

Abstract

Branching in mixed-integer (or integer) linear programming requires choosing both the branching variable and the branching direction. This paper develops a number of new methods for making those two decisions either independently or together with the goal of reaching the first integer-feasible solution quickly. These new methods are based on estimating the probability of satisfying a constraint at the child node given a variable/direction pair. The surprising result is that the first integer-feasible solution is usually found much more quickly when the variable/direction pair with the smallest probability of satisfying the constraint is chosen. This is because this selection forces change in many candidate variables simultaneously, leading to an integer-feasible solution sooner. Extensive empirical results are given.

1. Introduction

Mixed-integer linear programs (MILP) are composed of a linear objective function and linear constraints over a set of variables, some or all of which are restricted to take on integer or binary values ("integer" is assumed to include "binary" as a special case hereafter). The most popular solution approach is branch and bound, supplemented with cuts and various other heuristics such as local searching (see e.g. Johnson et al. (2000)). Nodes in the resulting search tree are variations on the original model with tightened bounds on the integer variables and/or added cut constraints. The bounding formula applied at a node in the search tree consists of the solution of a *linear programming (LP) relaxation* of the modified version of the original model represented by the node. The LP relaxation is a linear programming solution of the node model that simply ignores the integer restrictions on the variables.

A branch and bound node that is chosen for further expansion has an LP relaxation solution in which at least one of the integer variables does not have an integer value; such integer variables are *candidates* for branching. A candidate variable is chosen for branching and two child nodes are created: *branching up* adjusts the bound on the branching variable to be no less

than the current value rounded up, while *branching down* adjusts the bound on the branching variable to be no more than the current value rounded down. If there are *k* candidate variables, then there are 2*k* ways to proceed from the current node to the next node in the usual depth-first exploration of the search tree. The heuristic used for choosing which of the 2*k* potential child nodes to explore next can have a major impact on the speed of the MILP solution.

There are two main ways to *branch*, i.e. to choose the child node to explore next. The most common approach is to first choose the candidate variable, and then choose the *branching direction* (i.e. decide whether to branch up or down). A great deal of research exists on heuristics for choosing the candidate variable, but there is surprisingly little research on the best way to choose the branching direction once the candidate variable has been selected. The second approach is to choose the branching variable and the branching direction simultaneously. There is relatively little research on techniques in this category.

This paper addresses the question of the best branching heuristic (i.e. the heuristic that most often reaches the first feasible solution fastest), given the node that is to be expanded. Exploring this question sheds some light on the characteristics of MILP models that affect how well various branching techniques work. Influential characteristics include the presence or absence of equality constraints, the inclusion of "multiple choice" constraints, and the fraction of inequality constraints that are violated by adjusting the branching variable in the up vs. down directions. The analysis uncovers some important general principles in branching.

The metric of interest in this paper is speed in reaching the first integer-feasible solution. This is important for several reasons. First, several classes of MILP problems do not have an objective function and require only a feasible solution. Second, in very large MILPs it is wise to reach an incumbent solution early so that at least one integer-feasible solution is in hand should the time limit be reached, and because an incumbent is then available for pruning the developing tree, thereby reducing the overall solution time. Third, if the node selection heuristic is effective, then reaching an integer-feasible descendent of the chosen node quickly should lead to an optimum solution more quickly.

There are a variety of branching heuristics, though most are oriented towards fast optimality as opposed to fast integer-feasibility; see e.g. Achterberg et al. [2005] for an overview and assessment. Many algorithms first choose the branching variable and then the branching direction, using separate algorithms for each decision. Given the branching variable, the most common direction selection heuristics are: branch up always, branch down always, or branch to the closest integer. Branching to the farthest integer is also sometimes used. A number of commercial solvers include a parameter that allows the user to make any of these choices, along with the choice to let the solver decide the branching direction using its own heuristic.

Previous research provides no definite conclusions about which branching direction heuristic is best. Meyer et al. [2003] studied branching direction selection in the context of optimizing the placement of radioactive seeds for cancer treatment. They compared the branch up, branch down, and closest integer direction selection heuristics in combination with different node

selection, variable selection and scaling techniques, as well as with variations on the MILP model itself. They conclude that the branching direction heuristic has a significant impact on solution times, but could not identify an overall best heuristic: each method performed better or worse depending on the scaling method used. For example, branching down worked well with aggressive scaling, and branching up worked best with standard scaling.

Jariwala [1995] studied branch and bound solutions to the dynamic layout problem, comparing the usual three branching direction selection heuristics: branch up, branch down, and closest integer. He concludes that fixing the branching variable direction selection to either branch up or branch down is more effective than using the closest integer heuristic for this problem. Bernatzki et al. [1998] looked at branching direction selection heuristics in a MILP model for optimizing scrap combination for steel production. They test branching up and branching down on the binary variables, and concluded that branching down performs best for this model.

Driebeek [1966] developed a well-known branching heuristic that selects both the branching variable and the branching direction. Tomlin [1971] extended Driebeek's idea by considering the integrality of variables. The resulting Driebeek and Tomlin method is a penalty method that estimates the potential degradation of the objective function value due to selecting a candidate variable, as estimated by performing a dual simplex pivot, which generates a lower bound on the bound improvement possible if a given candidate variable is selected as the branching variable [Linderoth and Ralphs 2004]. Degradation bounds are calculated separately for branching up and for branching down. The largest degradation bound is used to choose the branching variable, and once chosen, the smaller of the bounds for that variable is used to choose the branching direction. This method is designed to reach optimality quickly, rather than integer feasibility. It is the default heuristic for branching variable and direction selection in the GLPK MILP solver [Makhorin 2008] that is used in the experiments reported in this paper.

There are a variety of specialized algorithms for reaching integer-feasibility quickly in MILPs that are not solely branching heuristics. The pivot-and-shift algorithm (Balas and Martin 1986) has a first phase that seeks integer feasibility using a variety of special techniques including rounding, specialized pivots and small neighbourhood searches. The OCTANE heuristic (Balas et al. 2001) for binary integer programs uses the intersection of the improving direction with the extended facets of the solution hyper-octagon to identify good binary solutions to try. The feasibility pump (Fischetti et al. 2005) alternates linear programming solutions with rounding. This paper concentrates on branching-related methods.

The general folklore is that branching in the up direction is usually best. This is definitely true in the case of so-called *"multiple choice" constraints*, which are composed entirely of binary variables and have the form $x_1 + x_2 + x_3 + ... x_n \{\leq,=\} 1$. For these constraints, branching up on the branching variable (i.e. setting it to 1) forces all other variables in the constraint to zero, hence all variables take on integer values simultaneously. On the other hand, branching down on the branching variable (i.e. setting it to 0) allows the other variables to take on non-integer values, hence fewer, if any, are forced to integer values. Branching up on a variable in a multiple choice constraint is decidedly preferable for reaching integer-feasibility quickly.

While not guaranteed, each branch in a branch and bound solution will more often than not force the branching variable to an integer value. To reach an integer feasible solution more quickly it is desirable to force as many additional candidate variables as possible to integer values at each branch. How to do this is straightforward in the case of multiple choice constraints, but not so obvious for other classes of constraints. However because every candidate variable must be forced to an integer value to reach an integer-feasible solution it is obviously a poor idea to branch in such a way that few candidate variables are affected. The generalization of this idea is that branching should force as many candidate variables as possible to change their values, whether or not it can be guaranteed that they will change to integer values: some *may* be forced to integer values, thereby speeding the solution.

This brings us to the central theme of this paper, namely branching to force change in the candidate variable values. This is extremely effective in the case of multiple choice constraints, where branching up forces all candidate variables in the constraint to integer values simultaneously. This principle has not been previously articulated as a central motivation in branching heuristics. Most branching variable selection heuristics concentrate on forcing change in the value of the objective function. The single exception is the active constraints branching variable selection method of Patel and Chinneck [2007], which concentrates on choosing the branching variable that has the greatest impact on the active constraints in the current LP-relaxation solution. Patel and Chinneck's *Method A* chooses the candidate variable that appears in the largest number of active constraints. In so doing it is also choosing the candidate variable that affects the values of the largest number of other candidate variables via their involvement in the active constraints. This is a very effective heuristic, outperforming state of the art commercial MILP solvers in reaching the first integer-feasible solution quickly. We will return throughout the paper to this theme of choosing the branching variable and branching direction so as to force change in the values of numerous candidate variables.

2. Experimental Setup

The conclusions in this paper are based on extensive computational experimentation. The experimental conditions are described below.

The open-source GLPK MILP solver version 4.28 [Makhorin 2008] was modified extensively to test a variety of existing and novel branching heuristics. All parameters were set at their default values with the following exceptions:

- *Stopping conditions*. Solutions ran for a maximum of two hours, or stopped earlier upon finding the first integer-feasible solution.
- *Node selection*. Depth-first, except where noted.
- *Branching variable selection and branching direction selection*. As required for the experiment at hand.

Hardware consisted of four computers running Windows XP: a Pentium 4 CPU at 3.40GHz with 1 GB of RAM, an Intel Core 2 CPU at 3.40GHz with 3GB of RAM (with solutions running as single

threads), a Pentium 4 CPU at 3.20GHz with 3GB of RAM, and a Pentium 4 CPU at 2.4 GHz with 1 GB of RAM. Multiple machines were needed to handle the computational load, which extended over numerous weeks. Given the different machine specifications, the major metric for speed is the total number of simplex iterations required for the MILP solution. This tracks clock time closely when node selection, variable selection, and branching direction selection heuristics are simple and do not require major computation.

A second metric is the number of models solved to integer-feasibility within the two-hour time limit by a particular method. This could be affected by the slightly different machine speeds. To minimize this effect, each model in a given comparison was normally solved on the same machine using the different methods being compared; exceptions are noted. In a number of test cases the same models were run using the same method on the fastest machine and again on a slower machine: there was no difference in the number of completed models, so this is likely a minor effect.

To provide a comparison to a known reference point across all experiments, all performance profiles include the data for GLPK with all parameters at default settings. The main default parameter settings of interest are branching variable and direction selection via the Driebeek and Tomlin heuristic [Tomlin 1971], and node selection via best value of the bounding function. Note that the GLPK default method was always run on the fastest machine, so the relative number of model completions was as large as possible, making for a conservative comparison.

Performance profiles [Dolan and Moré 2002] are used to summarize the results throughout the paper. These provide a graphical summary of the fraction of models (vertical axis) for which the solution simplex iterations for a given competing method is within some ratio of the number of simplex iterations used by the best competing method (horizontal axis). This is a much more meaningful generalization of the simple statistic giving the fraction of models for which a given method is the fastest. Performance profiles also provide information about the robustness of a competing algorithm, which is shown by the maximum height achieved on the vertical axis, reflecting the fraction of the models solved by the method within the time limit. We plot the ratio to the fewest simplex iterations only as far as 10, though there may be higher values in the data set.

Four well-known MILP problem sets comprising a total of 142 models were used: 56 models from MIPLIB2003 [Achterberg et al. 2006] (the *momentum3*, *msc98-ip*, *rd-rplusc-21*, and *stp3d* models were omitted because they were not solved by any method within the time limit), 11 models from MIPLIB3.0 [Bixby et al. 1998], 7 models from MIPLIB 2.0 [Bixby et al. 1992], and 68 models from Coral [Linderoth 2009]. These include a variety of real-world problems and are representative of a good general cross-section of MILPs [Achterberg et al. 2006]. At various points in the paper the models are divided into an "equality-containing" subset that consists of 47 models that include at least one equality constraint involving an integer variable, and an "equality-free" subset that consists of 95 models that include no equality constraints that involve integer variables. The complete set of models used in the experiments is shown in Table 1.

10teams	gesa2	*neos-1056905*	*neos-1436709*	*neos-785912*	*neos-957143*	roll3000
a1c1s1	gesa2-o	*neos-1122047*	*neos-1436713*	*neos-785914*	*neos-957323*	rout
aflow30a	glass4	*neos-1171448*	*neos-1439395*	*neos-787933*	net12	*sentoy*
aflow40b	*gt2*	*neos-1171692*	*neos-1440447*	*neos-841664*	noswot	set1ch
air04	harp2	*neos-1171737*	*neos-1440457*	*neos-856059*	*nsrand-ipx*	*seymour*
air05	*liu*	*neos-1200887*	*neos-1440460*	*neos-872648*	nw04	*seymour.disj-10*
arki001	*lseu*	*neos-1211578*	*neos-1442119*	*neos-873061*	opt1217	*sp97ar*
atlanta-ip	*manna81*	*neos-1228986*	*neos-1442657*	*neos-885086*	*p0033*	*sp97ic*
bell3a	markshare1	*neos13*	*neos-1467067*	*neos-885524*	*p0040*	*sp98ar*
bell3b	markshare2	*neos-1311124*	*neos-1480121*	*neos-886822*	*p0201*	*sp98ic*
bell4	*mas74*	*neos-1337489*	*neos-1516309*	*neos-932721*	*p0282*	*sp98ir*
bell5	*mas76*	*neos-1346382*	*neos-1599274*	*neos-932816*	*p0291*	*stein15*
bm23	misc07	*neos-1420205*	*neos-1616732*	*neos-933638*	*p0548*	*stein27*
cap6000	mkc	*neos-1426635*	*neos-495307*	*neos-933966*	*p2756*	*stein45*
dano3mip	*mod008*	*neos-1426662*	*neos5*	*neos-934278*	pk1	swath
danoint	mod011	*neos-1427181*	*neos-544324*	*neos-934441*	pp08a	t1717
disktom	modglob	*neos-1427261*	*neos-547911*	*neos-948346*	pp08aCUTS	timtab1
ds	momentum1	*neos-1429185*	*neos-565672*	*neos-953928*	protfold	timtab2
fast0507	momentum2	*neos-1429461*	*neos-702280*	*neos-954925*	qiu	tr12-30
fiber	mzzv11	*neos-1430701*	*neos-785899*	*neos-956971*	*ramos3*	vpm2
fixnet6	mzzv42z					

Table 1: Models Used in Experiments. Equality-Free Models Shown in Italics.

3. Evaluation of Simple Branching Direction Selection Heuristics

Given the branching variable, there are only two choices available for the branching direction: the up branch or the down branch. The three most common heuristics for making this choice are (i) branch up always, (ii) branch down always, or (iii) branch to closest integer (i.e. branch down if the fractional part of the branching variable has a value less than 0.5 and branch up otherwise). One might reasonably guess that branching up always would be no better or worse than branching down always, on the assumption that the composition of the model is more or less uniformly distributed in terms of coefficient signs and magnitudes and types of constraints. On the other hand, the folklore in the community is that branching up always is the superior choice. We tested the three basic branching direction heuristics over our complete set of test models, with results as shown in the performance profile in Figure 1.

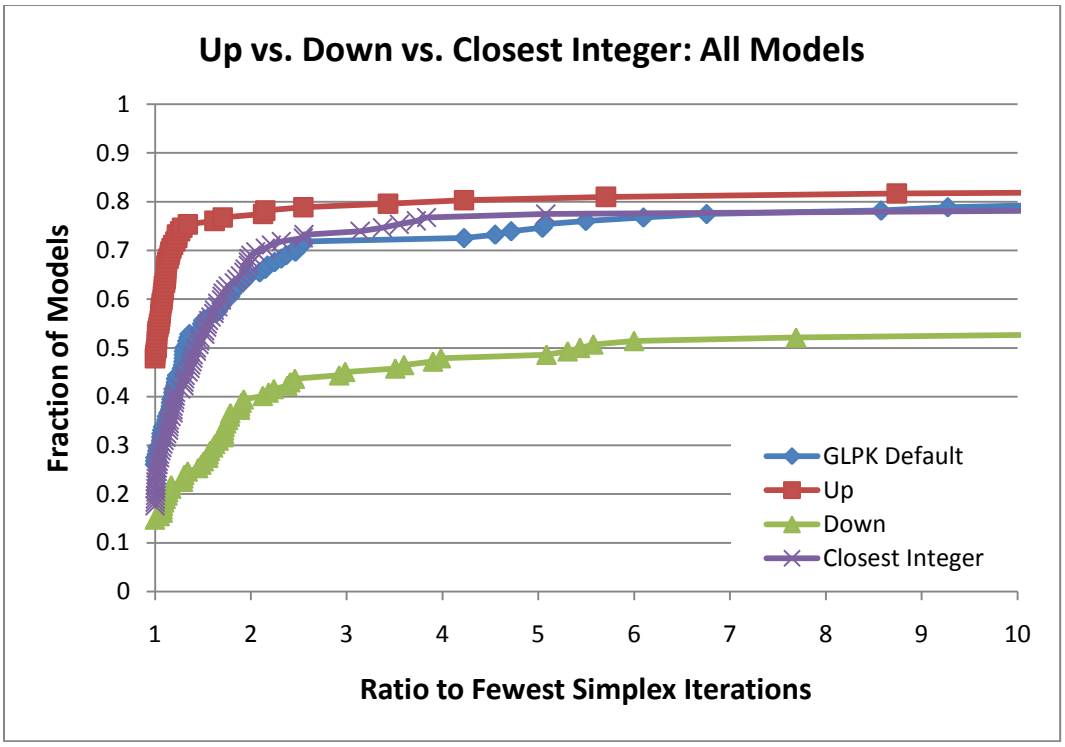

Figure 1: Simple Branching Direction Heuristics Over All Models

As Figure 1 shows, branching up always is indeed usually better, and is in fact quite dominant. Branching down always is the worst heuristic by a significant margin. While these results are for default GLPK with the simple substitution of the specified branching direction selection heuristic, the results are similar for a variety of other node and branching variable selection combinations that we tested.

The relative performance of the branching direction selection heuristics is also similar for the equality-free subset of models, but is quite different for the equality-containing subset, as shown in Figure 2. For the equality-containing subset, note that all methods are significantly less robust, with the three best methods tied in solving 74.5% of the models (vs. 85.9% for the up always method over all models). Second, when there is at least one equality constraint in the model then branching up always does not always dominate. Both GLPK default and branching to the closest integer have a larger fraction of solutions that are within a ratio of 2 of the smallest number of simplex iterations. This is so despite the fact that 33 of the 51 models (64.7%) in the equality-containing set include multiple choice constraints, for which it is known that branching up is the preferred option.

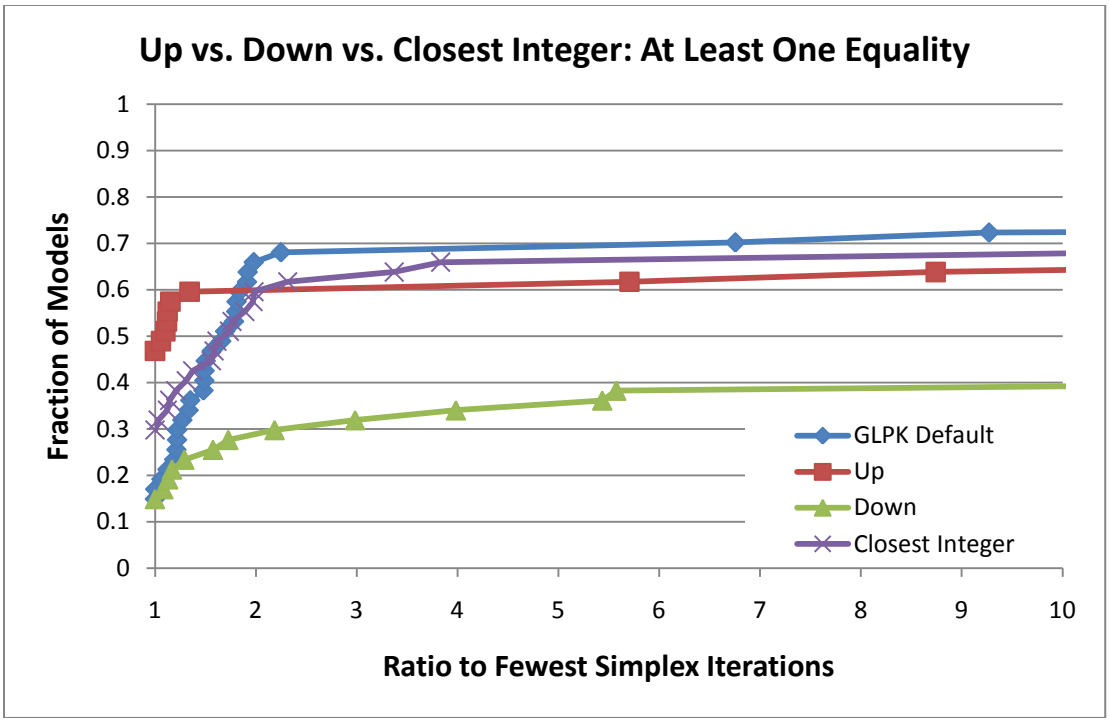

Figure 2: Simple Branching Direction Heuristics: Subset of Models Containing at Least One Equality Constraint

The presence or absence of equality constraints is a MIP characteristic that affects the performance of branching methods. As noted earlier, multiple choice constraints are a particularly influential form of equality constraint.

We will return in Section 8 to the question of why branching up is generally the better choice.

4. Probability-Based Branching Methods

Pesant and Quimper [2008] describe a method for counting solutions in knapsack constraints of the form $l \le \boldsymbol{cx} \le u$ where l and u are integer values representing the lower and upper bounds respectively, $\boldsymbol{c}$ is a row vector of integers, and $\boldsymbol{x}$ is a column vector of integer variables. A count can be established for variables whose values have not yet been fixed. For example, consider the simple constraint $x_1 + 5x_2 \le 10$ (in general, $g(\boldsymbol{x}) \le b$). We can count the number of solutions that satisfy the constraint when x_2 is fixed at each of its possible values, and calculate the "solution density" as shown in Table 2.

Value of x_2	Allowable range for x_1	Solution count	Solution density
x_2=0	[0,10]	11	11/18 = 0.61
x_2=1	[0,5]	6	6/18 = 0.33
x_2=2	[0]	1	1/18 = 0.06
Total solutions		18	

Table 2: Solution Density for $x_1 + 5x_2 \le 10$.

This information can then be used in assigning values to the unfixed variables. For example, you could choose to fix a variable at the value that has the maximum solution density in order

to provide the largest likelihood that you will be able to assign values to other unfixed variables in a way that will satisfy the constraint. This reasoning would lead you to set x_2=0 in this example because this value has the largest solution density.

More generally, the solution counting approach can be applied to examine the possible values of $g(\boldsymbol{x})$ when the n variables (x_1, x_2, …, x_n) each have the range [l_i, u_i], coefficient c_i in the constraint, and are assumed to be uniformly distributed in their ranges [Pesant and Quimper 2008]. This linear combination of uniformly distributed random variables results in an approximately Gaussian normal distribution for $g(\boldsymbol{x})$, whose mean and variance are:

$$\mu = \sum_{i=1}^{n} c_i \frac{l_i + u_i}{2}, \tag{1}$$

$$\sigma^2 = \sum_{i=1}^{n} c_i^2 \frac{(u_i - l_i + 1)^2 - 1}{12}, \tag{2}$$

Consider the constraint $g(\boldsymbol{x}) = 3x_1 + 2x_2 + 5x_3 \leq 12$ in which all variables are integer with range [0, 5]. Ignoring the integrality of the variables, this yields an approximately Gaussian normal distribution for $g(\boldsymbol{x})$ shown in Figure 3 which has a mean of 25 and variance of 110.83. Since we require $g(\boldsymbol{x}) \leq 12$, the cumulative probability (shaded in Figure 3), gives an estimate of the likelihood of satisfying the constraint if the variable values are uniformly distributed in their ranges.

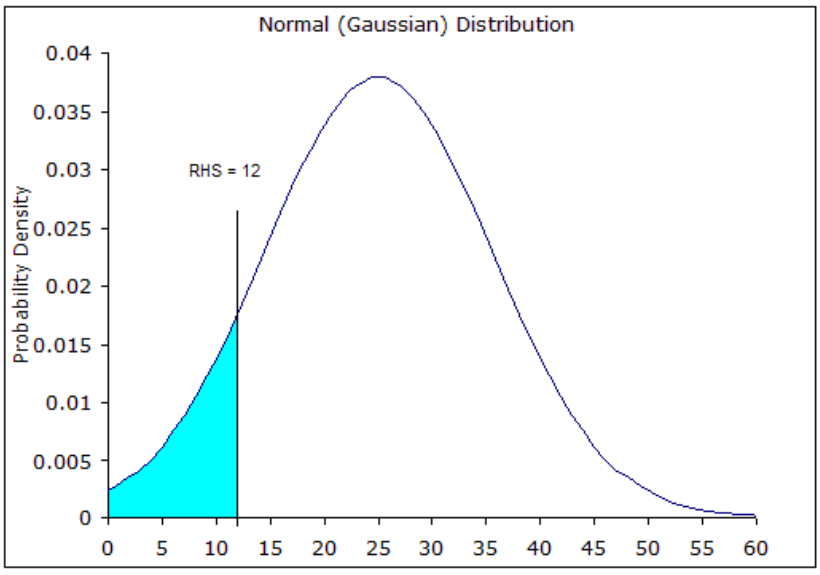

Figure 3: Gaussian distribution for $3x_1 + 2x_2 + 5x_3 \leq 12$

The cumulative probability for a Gaussian normal distribution can be analytically estimated by the method of Abramowitz and Stegun [1965]. For this example, the cumulative probability of the shaded portion of Figure 3 is 0.1084. Thus the probability of satisfying the constraint is 10.84% under the assumption of uniform distribution of the variable values.

We extend these ideas to develop a number of new branching methods for general mixed-integer linear programs. Let us first consider the case in which the branching variable has already been selected and only the choice of whether to branch up or down remains. Where the branching variable x_i has the range $[l_i, u_i]$ and current value $\bar{x}_i$ then the adjusted ranges in the two child nodes are $[l_i, \lfloor \bar{x}_i \rfloor]$ for the down branch and $[\lceil \bar{x}_i \rceil, u_i]$ for the up branch. This means that the normal distributions derived for the up and down branches will be different and the resulting probability of satisfying the constraint will be different for each branch. For the same constraint as in Figure 3, branching on $x_1 = 1.5$ yields a down branch with range [0,1] for x_1 and an up branch with range [2,5] for x_1. The two resulting Gaussian distributions for $g(\boldsymbol{x})$ are shown in Figure 4 and Figure 5.

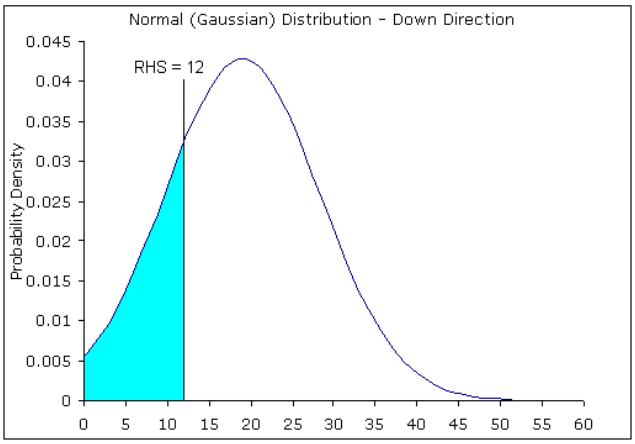

Figure 4: Gaussian distribution for the down branch

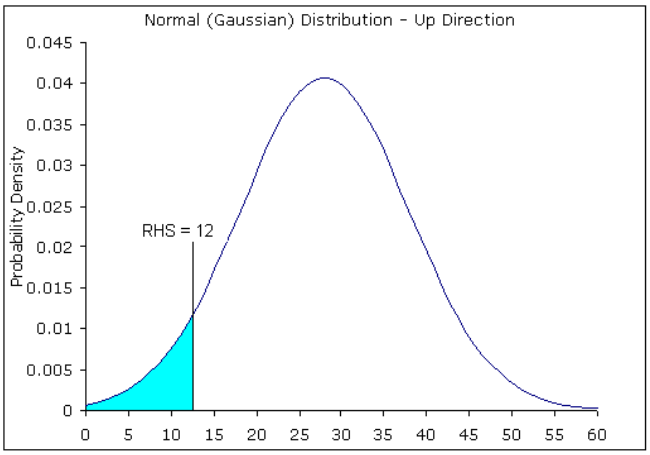

Figure 5: Gaussian distribution for the up branch

The cumulative probability of satisfying the constraint is 0.2262 for the down branch and 0.0511 for the up branch. These probabilities can then be used to help decide the branching direction. For example, one may choose a policy of always branching in the direction that provides the greatest probability of satisfying the constraint, or one may choose the opposite policy. We examine this choice in detail in Section 6.

The examples thus far have dealt with inequalities of ≤ form. Inequalities of ≥ are easily dealt with by calculating their satisfaction probability as 1 minus the cumulative probability of $g(\boldsymbol{x}) \leq b$.

Dealing with equality constraints is more challenging. Given the assumption of uniform distribution of the values of the variables within their ranges, the probability of exactly satisfying an equality constraint is vanishingly small when real-valued variables are also included in the constraint. For this reason, we instead develop a new measure that reflects how centered the value of b is within the probability density function for $g(\boldsymbol{x})$. The more centered b is within the density function, the more likely it is that values can be found for the variables in $g(\boldsymbol{x})$ that result in $g(\boldsymbol{x})=b$. "Centeredness" is measured by taking the ratio of the cumulative probability for $g(\boldsymbol{x})\leq b$ to the cumulative probability for $g(\boldsymbol{x})\geq b$. Specifically, the centeredness measure is:

$$Equality\ Centredness\ Measure = \frac{\min\left[P(g(\boldsymbol{x}) \leq b), P(g(\boldsymbol{x}) \geq b)\right]}{\max\left[P(g(\boldsymbol{x}) \leq b), P(g(\boldsymbol{x}) \geq b)\right]} \tag{3}$$

The maximum value for the equality centeredness measure is 1, which occurs when $P(g(\boldsymbol{x})\leq b) = P(g(\boldsymbol{x})\geq b)$, i.e. when b is exactly centered in the probability density function for $g(\boldsymbol{x})$. The minimum value of the equality centeredness measure is 0, which is achieved when b is at one of the ends of the probability density function for $g(\boldsymbol{x})$, i.e. is the least centered. Intermediate values between 0 and 1 indicate the degree of centeredness between least and most. This metric thus conveniently measures the degree of centeredness on the same 0 to 1 scale as the satisfaction probabilities associated with inequalities.

Both the 0 to 1 range and the meaning of the equality centeredness measure conveniently align with the probability measures used for inequalities. A higher value of the equality centeredness measure indicates a higher likelihood of satisfying the equality constraint in the same way that a higher cumulative probability indicates a higher likelihood of satisfying an inequality constraint. For this reason we will apply the centeredness measure for equalities in the same way that we apply the cumulative satisfaction probability in developing branching methods.

A number of branching methods can be developed using the probability and centeredness measures. Though the methods are given names related to probability, the equality centeredness measure is used for equality constraints. Given the branching variable, four such methods are:

- *Lowest Cumulative Probability (LCP)*: Suppose that the branching variable appears in k constraints. For each of the k constraints, calculate the probability of satisfying the constraint (or the centredness measure if the constraint is an equality) in the up and the down directions. Select the direction associated with the smallest of the $2k$ values.
- *Highest Cumulative Probability (HCP)*: Same as LCP except choose the direction associated with the largest of the $2k$ values.

- *Lowest Cumulative Probability Votes (LCPV)*: For each constraint that the branching variable appears in, calculate the relevant measure for the up and down directions. Assign a vote to the branching direction that has the lower value. Choose the branching direction that has the most votes over all constraints.
- *Highest Cumulative Probability Votes (HCPV)*: For each constraint that the branching variable appears in, calculate the relevant measure for the up and down directions. Assign a vote to the branching direction that has the higher value. Choose the branching direction that has the most votes over all constraints.

Similar methods for choosing the branching variable and the branching direction simultaneously can also be developed Two such methods are:

- *Variable and Direction Selection using Lowest Cumulative Probability (VDS-LCP)*: for every candidate variable, calculate the relevant measure for branching up and branching down in every constraint in which it appears and choose the candidate variable and branching direction that gives the lowest value of the measure in any constraint.
- *Variable and Direction Selection using Highest Cumulative Probability (VDS-HCP)*: for every candidate variable, calculate the relevant measure for branching up and branching down in every constraint in which it appears and choose the candidate variable and branching direction that gives the highest value of the measure in any constraint.

Numerous variations on these probability-measure-based methods are possible including branching in the opposite of the directions returned by any of the methods listed above, using values from the probability density function instead of the cumulative probability function, multiplying the probabilities, etc. See Pryor [2009] for a listing and evaluation of these other possibilities.

5. Branching Based on Constraint Violation and Satisfaction

Given the current LP relaxation solution, we can examine the effect on each active constraint of branching up or down on a candidate variable while all other variables are fixed at their current values. For inequality constraints this is simple: branching in one direction will violate an active inequality while branching in the other direction will continue to satisfy the constraint, and in fact will render the inequality inactive. Actual calculation of the new value of g(***x***) for each active constraint is not needed since the sign of the coefficient of the candidate variable and the type of inequality contain all of the needed information, as summarized in Table 3:

Coefficient Sign	Constraint Sense	Branching Direction	Result
+	≤	up	violated
+	≤	down	satisfied
+	≥	up	satisfied
+	≥	down	violated
–	≤	up	satisfied
–	≤	down	violated
–	≥	up	violated
–	≥	down	satisfied

Table 3: Violating or Satisfying Active Inequality Constraints.

In the case of equality constraints, any unilateral change in the value of a variable will violate the constraint. However branching in the up or down direction will change the range of the branching variable, and thus there will be different probabilities for $g(\boldsymbol{x})\leq b$ and $g(\boldsymbol{x})\geq b$ in each direction. We label the four resulting probabilities as follows:

- If branching down: DB=P($g(\boldsymbol{x})\leq b$) and DA=P($g(\boldsymbol{x})\geq b$).
- If branching up: UB=P($g(\boldsymbol{x})\leq b$) and UA=P($g(\boldsymbol{x})\geq b$).

DA and DB are used in Equation 3 if we branch down, and UA and UB are used in Equation 3 if we branch up.

It is more efficient to use the four values DB, DA, UB, and UA directly to discover the more and less centered branching directions instead of calculating the value of Equation 3 once for each branching direction. The more violating direction, i.e. the direction that leads to the smallest value of Equation 3, is associated with min(DB, DA, UB, UA). The less violating direction, i.e. the direction that leads to the largest value of Equation 3, is associated with max[min(DB, DA), min(UB, UA)]. In the algorithms that follow below, the "satisfying" direction for an equality constraint is associated with the less violating direction and the "violating" direction for an equality constraint is associated with the more violating direction.

These definitions can be used to construct branching methods given the branching variable:

- *Most Satisfied Votes (MSV)*: for each active constraint containing the branching variable, register a vote in favour of the satisfying direction. Choose the direction with the largest number of votes.
- *Most Violated Votes (MVV)*: for each active constraint containing the branching variable, register a vote in favour of the violating direction. Choose the direction with the largest number of votes.

6. Branching to Force Change in the Candidate Variables

For our purposes, the measure of the quality of a branching method is how quickly the branch and bound method reaches integer feasibility. It follows that a good branching heuristic will cause as many as possible of the current candidate variables to take on integer values in the selected child node. In some special cases, such as multiple choice constraints, it is obvious how to do this: branching up forces all of the binary variables to integer values simultaneously.

In more general constraints it is difficult to know in advance which branch will result in integer values for the largest number of candidate variables. It is reasonable to assume, however, that branching in a way that forces change in the candidate variables is more likely to produce integer values than branching in a way that causes few candidate variables to change their values. In the latter case the candidate variables will simply maintain their current non-integer values whereas in the former case they may be forced to an integer value.

There are some existing indications that branching to force change results in faster achievement of integer feasibility. This was demonstrated by the active constraint branching variable selection methods of Patel and Chinneck [2007] which branch on the candidate variable that has the largest impact on the active constraints by some measure. Choosing the branching variable in this way propagates changes to a large number of additional candidate variables. These methods are highly effective in reaching integer feasibility quickly.

The concept that branching to force change in the candidate variable values will speed the achievement of the first integer feasible solution is now empirically testable given the new branching methods developed in Sections 4 and 5. For inequalities, branching in the direction that is most likely to satisfy the constraint will usually have less impact on the other variables in the constraint. Branching in the direction that is least likely to satisfy the constraint will have the opposite effect. For equalities, branching in the more centered direction should have less impact on the other candidate variables while branching in the less centered direction should have the most impact.

We test this concept by examining the various new branching methods in pairs. One method in each pair works towards forcing change in the candidate variables by choosing the direction that is least likely to satisfy the constraints, and the other method in the pair works towards satisfying the constraints so that fewer candidate variables are forced to change their values.

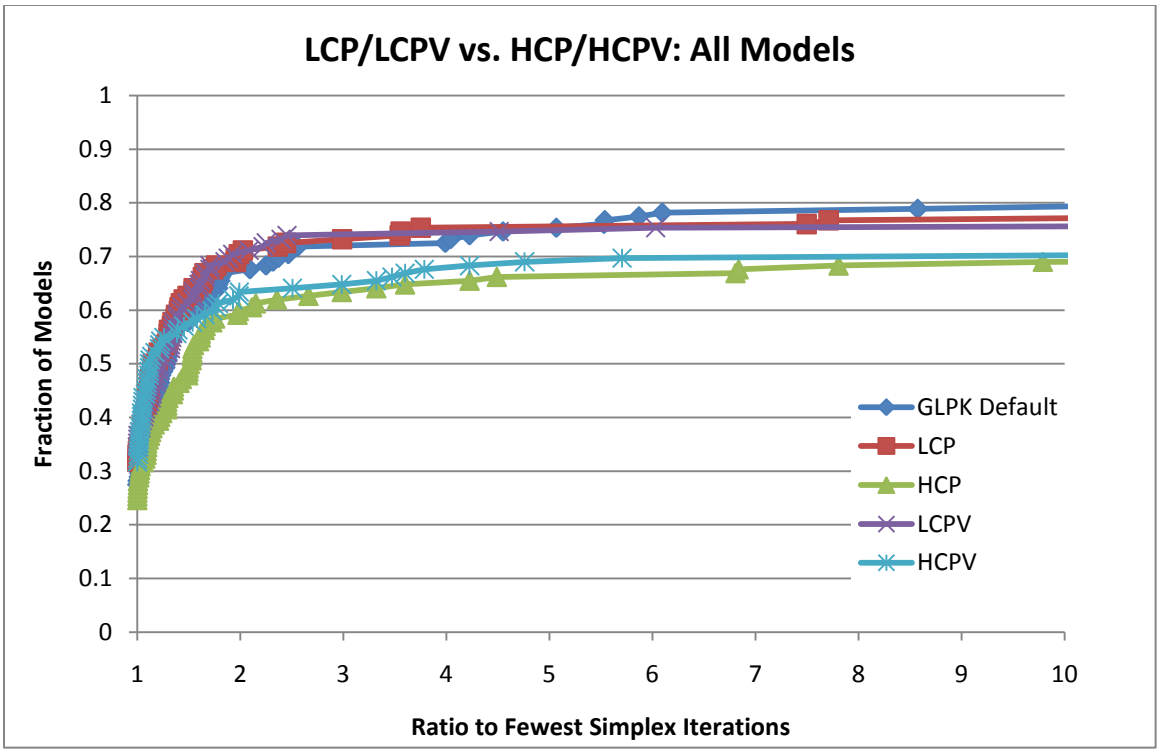

Figure 6: LCP/LPV vs. HCP/HCPV Over All Models

Figure 6 shows the results over all of the models in the test set when using the LCP, LCPV, HCP and HCPV methods to choose the branching direction, where the branching variable is chosen by the default GLPK method in all cases. Figure 6 shows that the two methods that work towards forcing change by choosing the direction having the smallest likelihood of satisfying the active constraints (LCP and LCPV) both give better results than their opposites (HCP and HCPV). The results are also slightly better than those for default GLPK.

The effect is much stronger when the probability-based methods select both the branching variable and the branching direction, as shown in Figure 7 for the VDS-LCP method, which forces change, vs. the VDS-HCP method, which does not. Both of these methods outperform default GLPK, but VDS-LCP dominates in terms of fewer simplex iterations required, as well as robustness in that it is able to reach a feasible solution within the time limit for 4 more models than VDS-HCP and 6 more models than default GLPK.

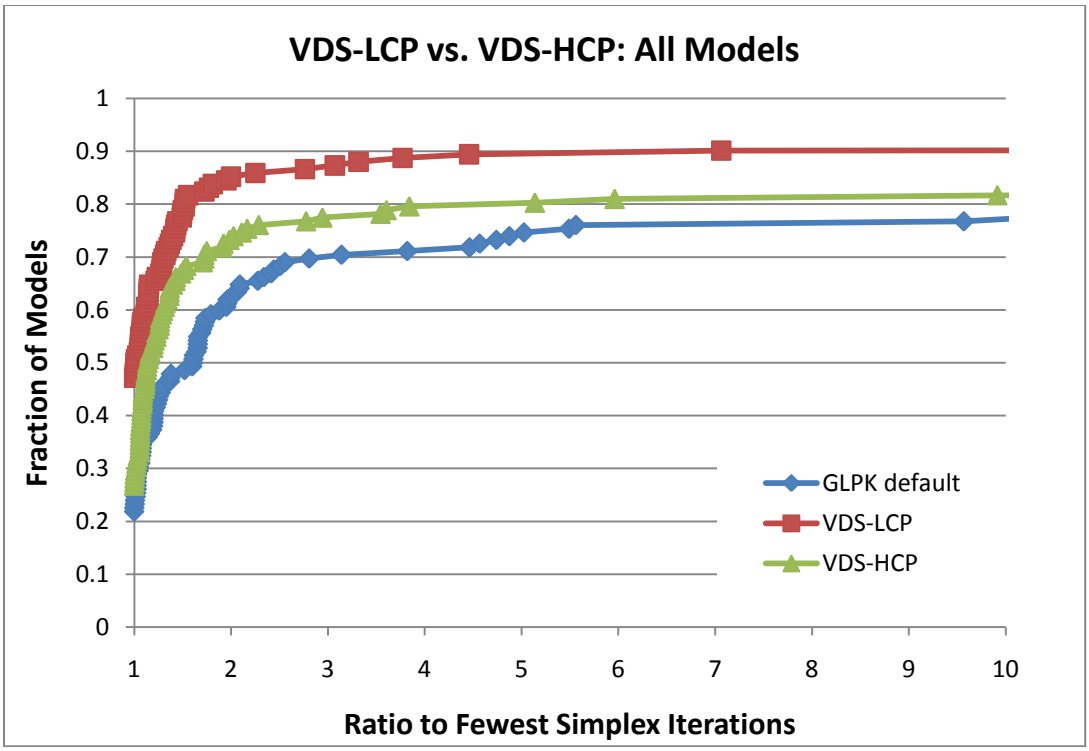

Figure 7: VDS-LPC vs. VDS-HCP Branching Over All Models

The pattern in Figure 7 is similar when the subset of equality-free models is examined, but including equality constraints has a significant impact, as shown in Figure 8. Now VDS-LCP dominates by a considerable margin, whereas VDS-HCP is only about as good as default GLPK. VDS-LCP also reaches a feasible solution for 7 more models than default GLPK and 8 more than VDS-HCP. Moving in the less-centered direction for the equality constraints has a significant impact.

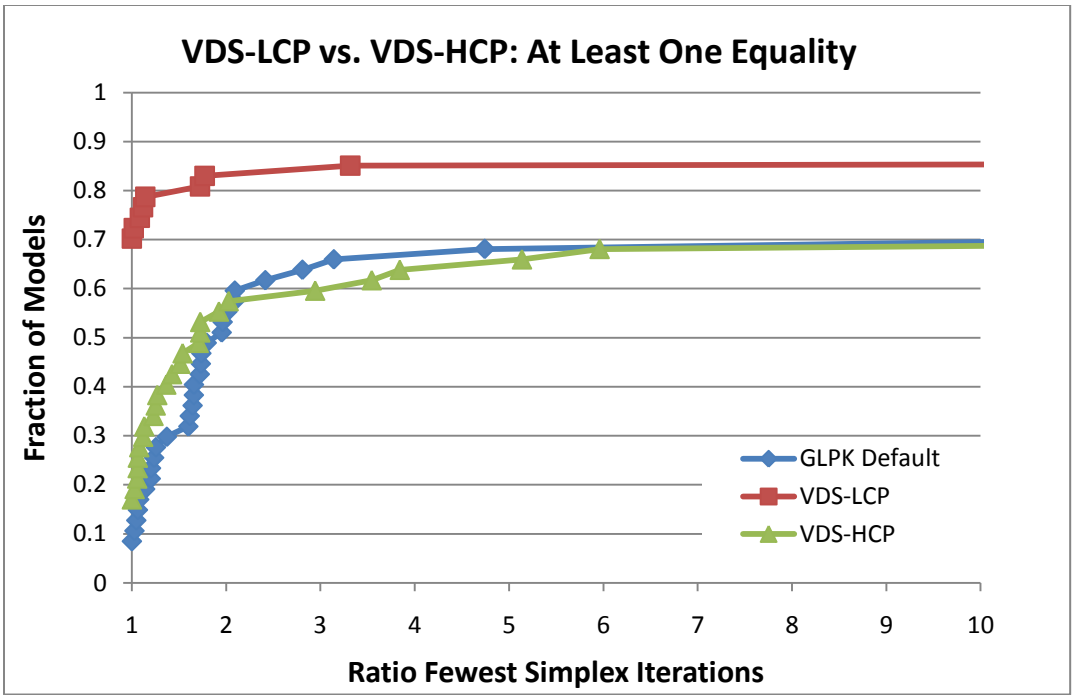

Figure 8: VDS-LCP vs. VDS-HCP When Equality Constraints are Present

Figure 9 shows the effect of branching based on an assessment of the number of active constraints violated or satisfied by a unilateral adjustment in the branching variable. The branching variable is selected by the default GLPK method, but the branching direction is selected by the MVV (most violated votes) or MSV (most satisfied votes) method. MVV direction selection far outperforms MSV direction selection, and outperforms default GLPK. These results also support the notion that branching to force change leads to integer feasibility sooner.

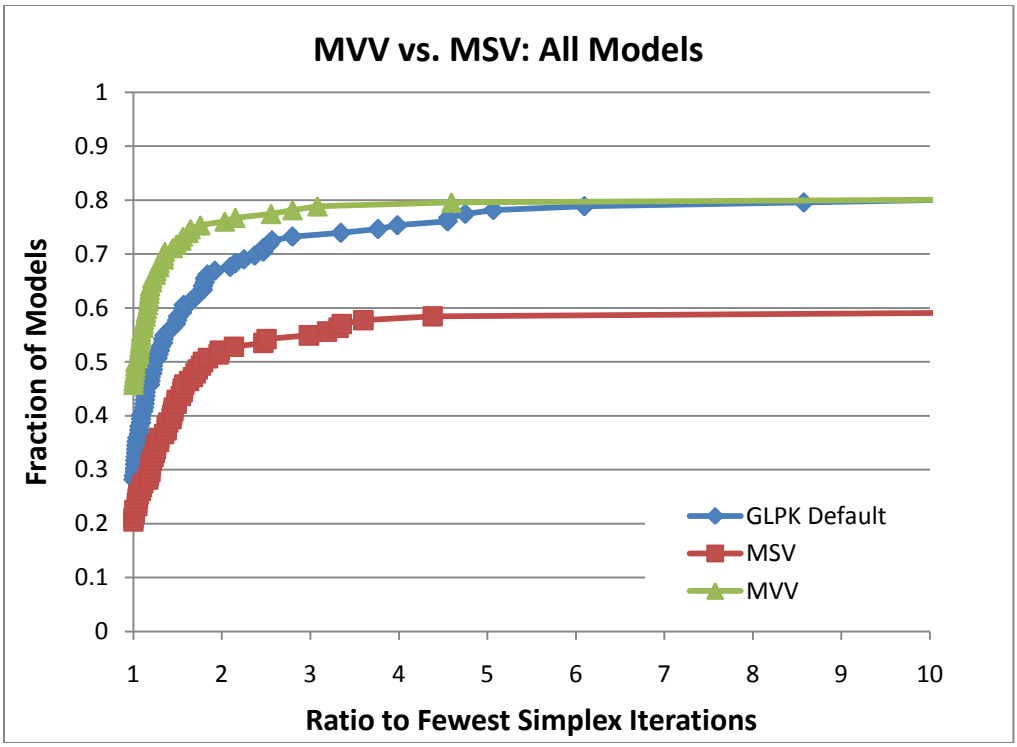

Figure 9: MVV vs. MSV Over All Models

When there are only inequalities in the model, MVV direction selection is strongly dominant when using the variable selected by GLPK, as shown in Figure 10. It is also more robust, solving 10 more models than MSV, and one more model than default GLPK within the time limit.

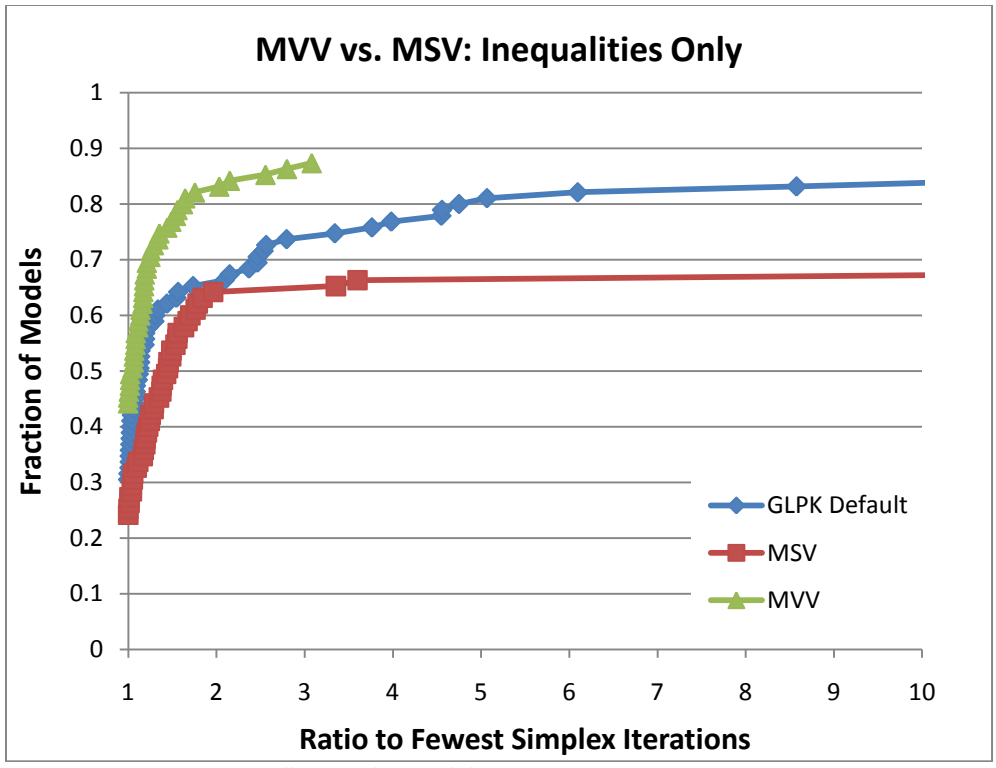

Figure 10: MVV vs. MSV in All-Inequality Models

However the picture is more mixed when the model contains equality constraints, as shown in Figure 11. In this case, MVV direction selection using the branching variable selected by GLPK uses the fewest simplex iterations on most models, but runs into difficulties on a few models.

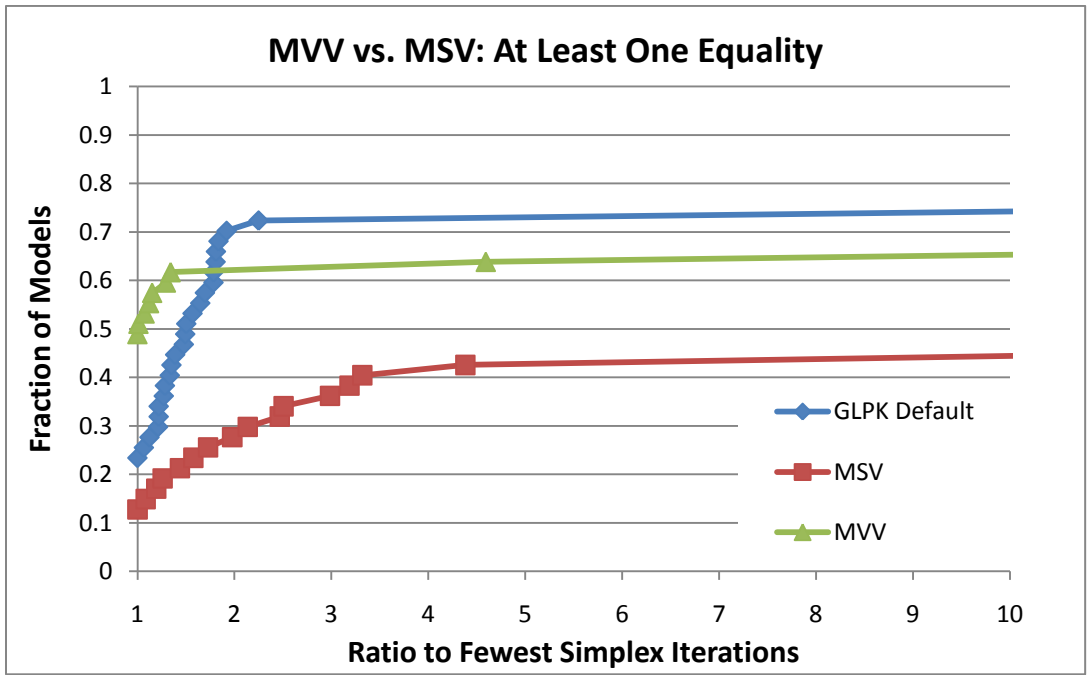

Figure 11: MVV vs. MSV When Equalities are Included

The active-constraints methods of Patel and Chinneck [2007] choose the candidate variable that has the most impact on the active constraints, in keeping with the goal of forcing the most change in the candidate variables. One of the simplest and most effective of these methods is Method A, which simply chooses the candidate variable that appears in the largest number of active constraints at the LP relaxation solution in the parent node. The effect of substituting Method A for branching variable selection in place of the GLPK default method for branching variable selection is examined in Table 4. The best value in each pair of results is shown in boldface. All four machines were used to generate these results, so the fraction solved results are not strictly comparable; the trends are clear nonetheless.

variable selection – direction selection	fraction fewest simplex iterations	fraction solved
GLPK Default	0.1620	0.8239
GLPK-UP	0.2887	0.8592
A-UP	**0.3662**	**0.8944**
GLPK-LCP	0.1831	0.8310
A-LCP	**0.3028**	**0.8592**
GLPK-LCPV	0.1901	0.7958
A-LCPV	**0.2394**	**0.8521**
GLPK-MVV	0.2042	0.8310
A-MVV	**0.3028**	**0.8521**

Table 4: Effect of Branching Variable Selection Method

As shown in Table 4, replacing the GLPK default branching variable selection method by Method A is always beneficial, whether the direction selection heuristic is always-up (UP), LCP, LCPV or MVV. When the 4 pairs of methods shown in the table are compared, the fraction of the models for which the method is best (in terms of simplex iterations) always increases, and the total fraction of models that are solved to integer feasibility within the time limit always increases when Method A replaces the GLPK default branching variable selection procedure.

The overall best method among those compared in Table 4 is Method A for variable selection coupled with always branching in the up direction. This is the combination of methods suggested by Patel and Chinneck [2007] and is the best of the branching methods in the literature for seeking MIP feasibility.

The dominant conclusion from these experiments is clear: branching to force change in as many candidate variables as possible is a superior tactic for reaching integer feasibility quickly. This finding is consistent across a number of different branching methods that assess the probability of reaching an LP-feasible solution for the constraints under consideration, as well as across methods based on assessing whether a unilateral change in the branching variable will violate or satisfy the active constraints.

The single exception to this conclusion that we have identified thus far is the case of set-covering problems, whose constraints are all the form $x_1 + x_2 + \ldots + x_n \geq 1$. Branching to force

change suggests that branching down will provide superior results in terms of reaching a feasible solution quickly, however, this is not the case. This is not a surprise since a feasible (but poor) solution is readily available: simply set all variables equal to one. Given this, it follows readily that branching up will provide better results; we confirmed this experimentally.

7. A-UP vs. VDS-LCP

Choosing the branching variable via Method A coupled with always branching up (A-UP) represents the current state of the art in branching methods for seeking integer feasibility in MILPs. In A-UP, the branching variable selection is disconnected from the branching direction selection, unlike method VDS-LCP, which makes both decisions simultaneously. These two approaches are compared in Figure 12, which shows that the two different methods produce remarkably similar results, in fact an analysis of variance shows no significant difference between the results. VDS-LCP is slightly slower on some models, but is able to solve 3 more models to feasibility than A-UP (it solves 91.5% of the models vs. 89.4% for A-UP). This shows that a probability-based branching method that operates to force change is equivalent to the state of the art branching method in seeking integer feasibility (which is itself based on the principle of branching to force change).

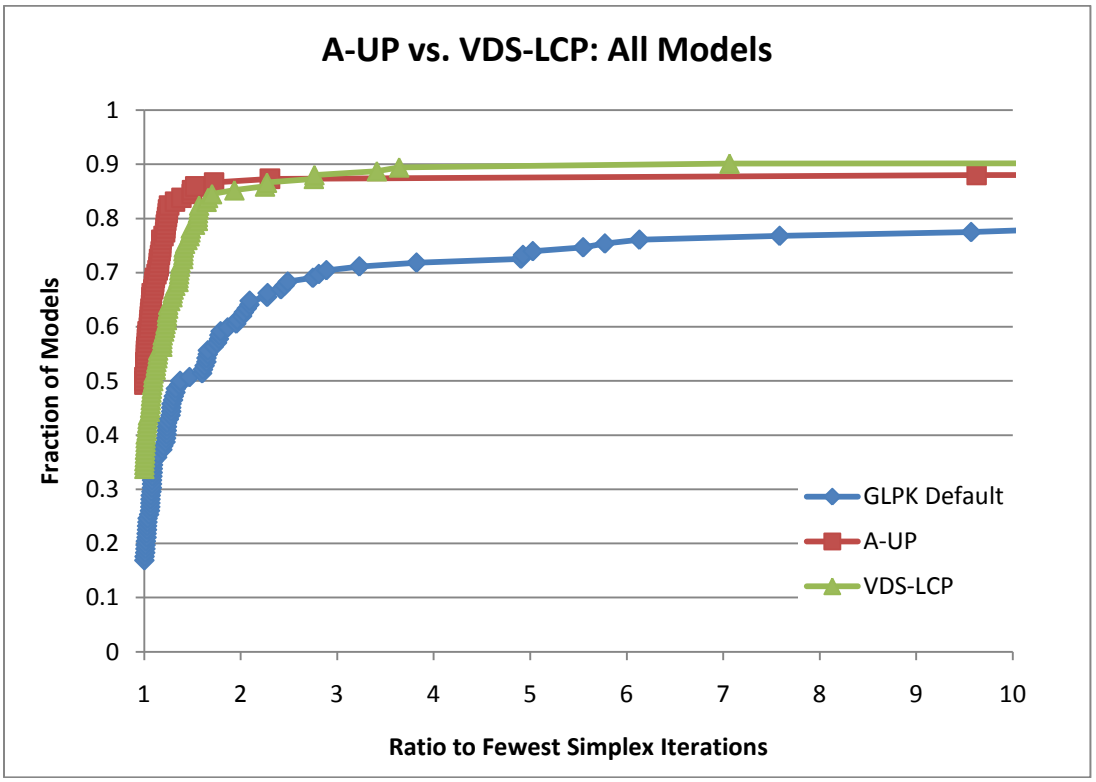

Figure 12: A-UP vs. VDS-LCP

VDS-LCP actually outperforms A-UP on the subset of 47 equality-containing models, having 4 more fastest solutions and reaching integer feasibility within the time limit on 2 more models. This raises the possibility of hybrid methods based on using VDS-LCP when there are equality constraints in the model and A-UP otherwise; this is left for future research.

8. Branching Up Revisited

We now revisit the question of why branching up always is such a good heuristic, as shown in Section 3. As will be seen, this is mainly due to the characteristics of the set of test MILPs.

Recall that branching up is always the best policy in a multiple choice constraint because this forces all variables in the constraint to integrality simultaneously. As it happens, 104 of the models in our test set contain at least one multiple choice constraint, versus just 38 without multiple choice constraints. With almost three times as many models having multiple choice constraints as not, it would be expected a priori that always branching up is a good policy. We can examine this in detail by looking at how well the branching techniques work when the model does or does not contain multiple choice constraints.

Figure 13 provides a performance profile for simplex iterations for the 104 models that include at least one multiple choice constraint. Method A branching up provides the best performance, followed by VDS-LCP, as expected.

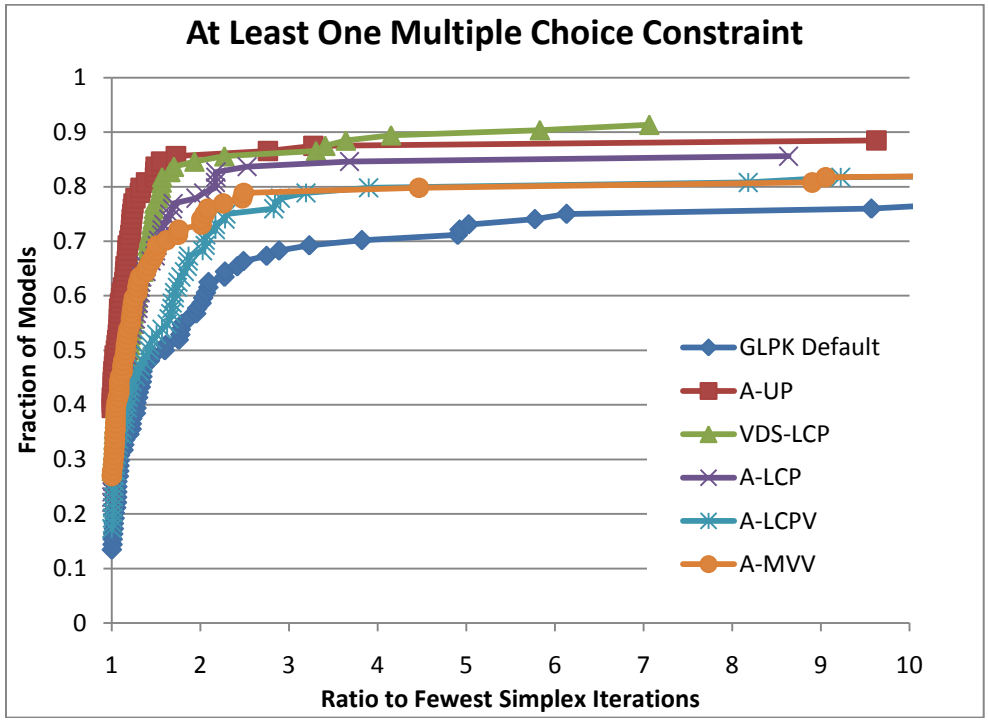

Figure 13: Models Having at Least One Multiple Choice Constraint

However, when there are no multiple choice constraints, the conclusion is a little different, as shown in Figure 14. Branching up exclusively is no longer the best policy: it is dominated by both MVV and LCPV, both of which originate in the notion of branching to force change in the candidate variable values.

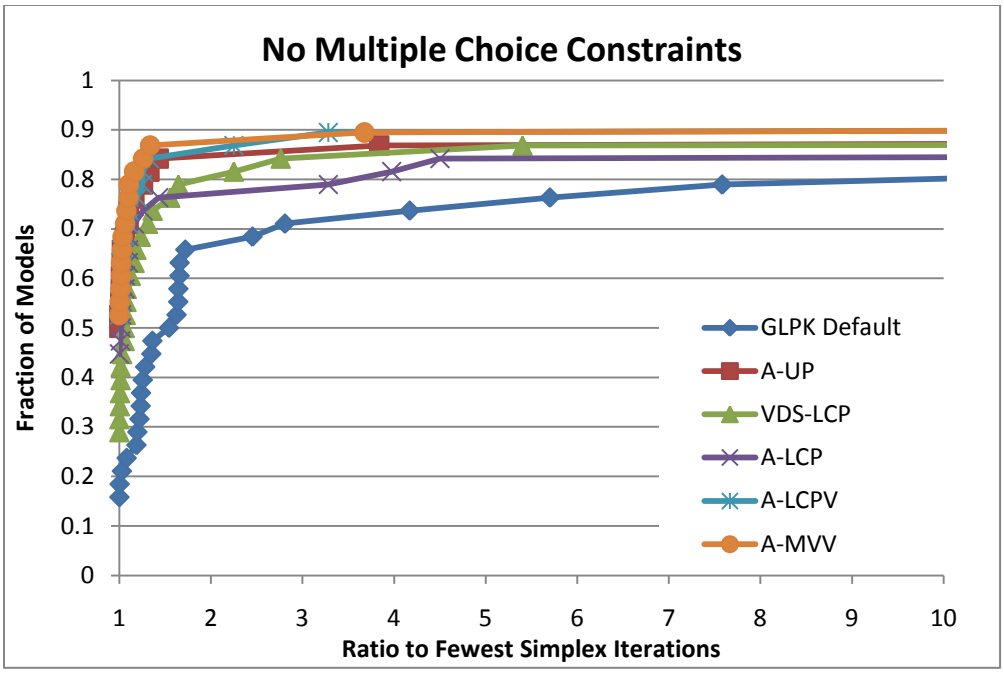

Figure 14: Models Without Multiple Choice Constraints

It is interesting to analyze the behavior of the LCP heuristic on multiple choice constraints. LCP always branches up on multiple choice inequality constraints. This is because all the variables have the same set of possible values and the same coefficients, and hence the values calculated in Eqns. 1 and 2 are always the same for a given number of candidate variables. Consider Table 5, which shows the cumulative probabilities of satisfying an inequality multiple choice constraint when branching up or down. LCP, which chooses the lower cumulative probability, will always choose to branch up, as shown in boldface.

# Variables	Cumulative Probability Up	Cumulative Probability Down
2	**0.158655254**	0.841344746
3	**0.078649604**	0.5
4	**0.041632258**	0.281851431
5	**0.022750132**	0.158655254
6	**0.012673659**	0.089856247

Table 5: Cumulative Probabilities for the Branching Direction in an Inequality Multiple Choice Constraint

The situation is a little different when the multiple choice constraint is an equality because now Eqn. 3 is used, which measures the ratios of the cumulative probabilities. LCP chooses the direction whose ratio is farthest from 1.0. As shown in Table 6, LCP will always choose to branch up, except when there are two candidate variables, in which case there is no preference for branching up or down, as shown in boldface. This is exactly as expected since in an equality multiple choice constraint with two binary variables ($x_1 + x_2 = 1$), choosing to branch down will force the other candidate variable to a value of 1, i.e. both variables will have integer values. Hence when there are two candidate variables in an equality multiple choice constraint, both candidate variables are forced to integrality no matter which branching direction is chosen.

# Variables	Equality Ratio Up	Equality Ratio Down
2	**0.188573417**	**0.188573417**
3	**0.085363401**	1
4	**0.043440797**	0.392469529
5	**0.023279749**	0.188573417
6	**0.012836343**	0.098727533

Table 6: Cumulative Probability Ratios for the Branching Direction in an Equality Multiple Choice Constraint

9. Conclusions

This paper makes several useful contributions:

- The discovery of the principle that branching to force change in the candidate variables results in faster achievement of integer feasibility in a MILP.
- The extension of probability-based branching methods for general mixed-integer programming, most notably by the development of new methods for handling equality constraints by assessing the "centeredness" of the constant value in the probability distributions that result from branching up vs. branching down. As it turns out, the probability-based branching methods are particularly effective on models having equality constraints.
- The development of a number of useful new branching methods based on probability methods and the principle of branching to force change. One of these methods (VDS-LCP) chooses both the branching variable and the branching direction, and is as good as the existing state-of-the art branching method (A-UP) which treats these two decisions separately.

It is worth noting that the methods are unaffected by scaling. The MVV and MSV methods for inequality constraints use only the signs of the coefficients and not their magnitudes. All other methods are based on probability calculations (including MVV and MSV for equality constraints) which use only the lower and upper bounds on the variables and the coefficient values (Equations 1 and 2). Scaling will not affect the probability calculations and hence will not affect the heuristics.

There is wide scope for future research to explore the full potential of these ideas:

- Hybrid methods. There is evidence that the presence or absence of equality constraints and the presence or absence of multiple choice constraints impact the effectiveness of the branching methods. By observing which methods do best under which combination of conditions, hybrid methods can be developed which apply the best method under the current conditions. The decision as to which branching method to use can be made once at the beginning of the solution process after assessing the relevant factors, or can be made at each node of the search tree.
- A number of combinations of the elements of the new methods were not assessed. For example a new method for simultaneous variable and direction selection based on the MVV method could be developed. This would assess the violation votes in the up and

down directions for all candidate constraints and choose the variable and direction having the most votes.

- Dealing with ties for the selection of the branching variable and direction. When these decisions are made separately there are frequently numerous ties. For example Method A for the selection of the branching variable simply counts the number of active constraints that each candidate variable appears in, and ties are frequent. Using a second method to break this tie may improve the results.

References

Abramowitz, M., and Stegun, I.A., eds. (1965), Handbook of Mathematical Functions, National Bureau of Standards, Applied Mathematics Series, pp. 932.

Achterberg, T., Koch, T., and Martin, A. (2005), "Branching Rules Revisited", Operations Research Letters 33:42-54.

Achterberg, T., Koch, T., and Martin, A. (2006), “MIPLIB 2003”, Operations Research Letters 34:1-12.

Balas E., Ceria S., Dawande M., Margot F., Pataki G. (2001) "OCTANE: a New Heuristic for Pure 0-1 Programs", Operations Research 49:207-225.

Balas E., Martin C. (1986) "Pivot and Shift – a Heuristic for Mixed Integer Programming", GSIA Technical Report, Carnegie Mellon University.

Bernatzki, K.P., Bussieck, M.R., Lindner, T., and Lübbecke, M.E. (1998), “Optimal Scrap Combination for Steel Production”, OR Spektrum 20:251-258.

Bixby, R.E., Boyd, E.A., and Indovina, R.R. (1992), “MIPLIB: A Test Set of Mixed Integer Programming Problems”, SIAM News 25(2), March 1992.

Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P. (1998), “An Updated Mixed Integer Programming Library: MIPLIB 3.0”, Optima 58:12-15.

Dolan, E.D., and Moré, J. (2002), “Benchmarking Optimization Software with Performance Profiles”, Mathematical Programming, Series A 91:201-213.

Driebeek, N.J. (1966), “An Algorithm for the Solution of Mixed Integer Programming Problems”, Management Science 12:576–587.

Fischetti M., Glover F., Lodi A. (2005) "The Feasibility Pump", Mathematical Programming A 104:91-104.

Jariwala A. (1995), “Efficient Branch and Bound Algorithm for the Dynamic Layout Problem”, Master’s thesis, College of Engineering and Technology, Ohio University.

Johnson E.L., Nemhauser G.L., Savelsbergh M.W.P. (2000), "Progress in Linear-Programming-Based Algorithms for Integer Programming: An Exposition", INFORMS Journal on Computing 12:2-23.

Linderoth, J., (2009), "Coral: Mixed Integer Programming Instances", http://coral.ie.lehigh.edu/mip-instances/, accessed May 2009.

Linderoth, J., and Ralphs, T. (2004), “Noncommercial Software for Mixed-Integer Linear Programming”, Technical Report 04T-023, Department of Industrial and Systems Eng., Lehigh Univ. Online: www.lehigh.edu/~jtl3/teaching/ie418/papers/MILP04.pdf.

Makhorin, A. (2008), “GNU Linear Programming Kit Reference Manual Version 4.28”, http://www.gnu.org/software/glpk/, [Accessed May 2008].

Meyer, R.R., D’Souza,W.D., Ferris, M.C., and Thomadsen, B.R. (2003), “MIP Models and BB Strategies in Brachytherapy Treatment Optimization”, Journal of Global Optimization 25:23–42.

Patel, J., and Chinneck, J.W. (2007), “Active-Constraint Variable Ordering for Faster Feasibility of Mixed Integer Linear Programs”, Mathematical Programming, Series A 110:445-474.

Pesant, G., and Quimper, C.G. (2008), “Counting Solutions of Knapsack Constraints”, CPAIOR 2008, Lecture Notes in Computer Science 5015:203–217.

Pryor, J. (2009), "Branching Variable Direction Selection in Mixed Integer Programming", Master's thesis, Systems and Computer Engineering, Carleton University.

Tomlin, J.A. (1971), “An Improved Branch-and-Bound Method for Integer Programming”, Operations Research 19:1070–1075.