Computers and Operations Research (2013), to appear.

Achieving MILP Feasibility Quickly Using General Disjunctions

Hanan Mahmoud (hamahmou@connect.carleton.ca)
John W. Chinneck (chinneck@sce.carleton.ca)

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

March 8, 2013

Abstract

Branch and bound algorithms for Mixed-Integer Linear Programming (MILP) almost universally branch on a single variable to create disjunctions. General linear expressions involving multiple variables are another option for branching disjunctions, but are not used for two main reasons: (i) descendent LPs tend to solve more slowly because of the added constraints, so the overall solution time is increased, and (ii) it is difficult to quickly find an effective general disjunction. We study the use of general disjunctions to reach the first MILP-feasible solution quickly, showing for the first time that general disjunctions can provide speed improvements for hard MILP models. The speed-up is due to new and efficient ways to (i) trigger the inclusion of a general disjunction only when it is likely to be beneficial, and (ii) construct effective general disjunctions very quickly. Our empirical results show performance improvements versus a state of the art commercial MILP solver.

1. Introduction

A Mixed-Integer Linear Program (MILP) is composed of a linear objective function to be maximized or minimized, a set of linear constraints over a set of variables, some or all of which are restricted to take on integer or binary values (collectively, *integer variables* hereafter), and a set of variable bounds. MILP problems are commonly solved using the tree-structured Branch-and-Bound (B&B) method. Different versions of B&B algorithms arise when different heuristics are used for making key decisions in the tree search, such as the selection of the next node to explore or the variable to branch on. These heuristics can greatly affect the speed and efficiency of the solution.

Our focus is on branching disjunctions. Most B&B algorithms consider only a small set of disjunctions that select a single *candidate variable* (an integer variable that has a fractional value in the current LP relaxation solution), and a *branching direction* (up or down). The resulting disjunction is axis-parallel: in the *down branch* the upper bound on the selected candidate variable is reset to its rounded-down value, and in the *up branch* the lower bound is reset to its rounded-up value. This creates two child nodes in the B&B tree.

Very little research has been conducted on *general disjunctions* that involve multiple variables in a single linear disjunction equation. This paper introduces new branching heuristics that use general disjunctions to find the first MILP-feasible solution as quickly as possible, the first work of this kind as far as we are aware. Quick attainment of a MILP-feasible solution is useful for several reasons: sometimes a feasible solution is all that is required (e.g. many scheduling problems), having an early incumbent can speed the attainment of the optimal solution due to greater pruning of the search tree, having an incumbent is required in some node selection methods (e.g. best-projection), etc.

1.1 Related Work

There is a small body of prior research on using general disjunctions to attain MILP optimality quickly. Mahajan and Ralphs [2009] conducted a number of computational experiments on general disjunctions of the form $\boldsymbol{\pi x} \leq \pi_0$ and $\boldsymbol{\pi x} \geq \pi_0 + 1$, where $\boldsymbol{\pi} \epsilon \mathbb{Z}^d$ and $\pi_0 \epsilon \mathbb{Z}$, there are *d* integer variables and *n-d* real variables, and *n* is the total number of variables. The disjunction is constructed such that the new lower bound $z_l = \min\{z_L^*, z_R^*\}$ is maximized for a minimization objective, where z_L^* is the objective function value of the LP relaxation at the left child node (containing the general disjunction constraint $\boldsymbol{\pi x} \leq \pi_0$), and z_R^* is the objective function value of the LP relaxation at the right child node (containing the general disjunction constraint $\boldsymbol{\pi x} \geq \pi_0 + 1$). Two criteria are used to evaluate the branching disjunction: (i) lower bound improvement after branching and (ii) width of the created LP sub-problem in the direction of the disjunction. This method reduces the number of nodes in the search tree.

Karamanov and Cornuéjols [2011] recognized that branching on general disjunctions can give smaller trees because it more often yields one feasible child node instead of two. To obtain an efficient algorithm, they considered only disjunctions that define Gomory Mixed Integer Cuts (GMICs) where the distance cut off by the intersection is used to assess the quality of the disjunction. They reported that using general disjunctions closes more of the integrality gap.

Cornuéjols, Liberti and Nannicini [2011] confirmed that general disjunctions can reduce the number of nodes in the enumeration tree by a factor of more than two. They constructed disjunctions arising from GMICs generated from linear combinations of the rows of the simplex tableau. They considered a general disjunction to be profitable if it generated the smallest number of feasible children or if it closed more of the integrality gap. Their computational experiments combined single-variable disjunctions and general disjunctions, which led to an increase in closed gap per node but did not translate into computational time reductions.

Owen and Mehrotra [2001] proposed branching on general disjunctions generated by a neighbourhood search heuristic. The neighbourhood heuristic uses disjunctions with coefficients only from the set {-1,0,1} on integer variables. These disjunctions are oblique with an angle of 45 degrees (in two dimensions), and pass through integer-feasible points. The heuristic search procedure tries to find a disjunction that has maximum value over all possible valid disjunctions. The authors conclude that such general disjunctions can significantly decrease the size of the branching tree, but they do not report on time improvements.

There is broad agreement in the literature that general disjunctions can reduce the size of the search tree, but there is no reported evidence that they reduce the solution time. Further, it can be time-consuming to construct an effective general disjunction. Mahajan and Ralphs [2010] determined that the complexity of selecting the general disjunction at each node is NP-hard. Even selecting a general disjunction in which all the coefficients of the disjunction function are in the set {-1,0,1} is an NP-complete problem.

Most researchers focus on improving the speed to the optimum solution, but in many applications finding any feasible solution is advantageous, as mentioned above. In addition, a focus on feasibility often improves the speed to optimality. For one thing, feasible solutions that are found quickly are often closer to the root node and hence have better objective function values [Wojtaszek and Chinneck, 2010]. As our experiments show, our MILP-feasibility seeking methods provide first feasible solutions that have generally better optimality gaps on average than the first feasible solutions provided by a well-known commercial MILP solver.

A brief summary of existing methods for seeking MILP-feasible solutions follows. Fischetti et al. [2004] suggested the Feasibility Pump heuristic, which alternates between rounding an LP-feasible point so that it is integer-feasible (but no longer LP-feasible), and solving an auxiliary LP problem that finds the LP-feasible point that is closest to the rounded integer solution. This "pumping action" often results in a MILP-feasible point, though success is not guaranteed. The authors report good computational results on binary MILPs, which makes the feasibility pump a useful root node heuristic for seeking MILP-feasibility.

Patel and Chinneck [2007] presented a new branching variable selection method for finding the first MILP-feasible solution as quickly as possible. They estimate the impact of candidate variables on the active constraints in the LP relaxation and select the one having the most impact as the branching variable. Their empirical results show significant improvements vs. a state-of-the-art commercial solver. One of the most successful heuristics, *Method A,* simply chooses the branching variable that appears in the largest number of active constraints. The algorithms developed in this paper use some of the core ideas in these schemes.

Pryor and Chinneck [2011] developed a number of new branching variable and branching direction selection methods for reaching MILP-feasibility quickly. Given a variable/direction pair, they estimate the probability of satisfying each constraint at the child nodes, and conclude that the child with the smallest probability is most likely to lead to the first integer-feasible solution quickly. This is because the lower-probability choice forces the greatest amount of variable value propagation, in turn forcing simultaneous changes in the greatest number of candidate variables. *Multiple choice* constraints consisting entirely of binary variables and having the form $x_1 + x_2 + x_3 + ... x_n$ {≤,=} 1 are an example of this phenomenon: branching up on a candidate variable has a single feasible solution that forces all of the other variables to take integer values; branching down has multiple possible solutions, but does not force integrality.

1.2 Contributions

This paper studies the use of general disjunctions for reducing the time to reach the first MILP-feasible solution. Our contributions are methods for overcoming the two main

difficulties in using general disjunctions in MILPs: (i) deciding when to deploy a general disjunction, and (ii) deciding how to construct them. While it is known that general disjunctions can reduce the number of nodes in the B&B tree, they generally increase the total solution time, both because the search for an appropriate disjunction equation can be time-consuming, and because the LP is harder to solve at every descendent node because of the added constraints (which is not the case for the simple bound adjustments in standard axis-parallel disjunctions). We show that our methods shorten the time needed to find the first MILP-feasible solution for hard MILP instances (though it may not be obvious in advance which instances are hard).

2. Towards New General Disjunction Branching Heuristics

There is an infinite number of ways to construct a general disjunction, and it has been shown that selecting the best one is NP-hard [Mahajan and Ralphs 2010]. For this reason we concentrate on heuristic methods that can quickly construct a general disjunction that is likely to be effective. We use the principle discovered by Patel and Chinneck [2007] that effective feasibility-seeking disjunctions have the greatest impact on the active constraints in the current LP-relaxation solution. We apply this idea by deriving the general disjunction from the active constraint deemed to have the greatest impact. This *foundation constraint* can be identified quickly, and the general disjunction derived from it can be constructed rapidly.

General disjunctions take the form of a linear equation, i.e. $a_1x_1 + a_2x_2 + ... + a_nx_n \leq k$ and $a_1x_1 + a_2x_2 + ... + a_nx_n \geq k+1$, where the a_i and k are usually integer constants, and the x_i are variables. To further simplify their construction, we restrict the a_i to the values +1, -1, and 0, as in Owen and Mehrotra [2001], and refer to these as *45-degree general disjunctions* since they can take this angle in a two-variable model. 45-degree general disjunctions have the agreeable property of producing a clean separation having no integer solutions in the space between the branches, in a manner similar to branching on variables, as shown in Figure 1.

The four main elements in our method are: (i) deciding whether or not to deploy a general disjunction at all, (ii) selecting an original model constraint to use as the foundation constraint, (iii) using the foundation constraint to dictate how to assign the +1, -1, and 0 coefficients, and (iv) deciding on the branching direction. We describe these elements below.

2.1 Choosing the Foundation Constraint

Following Patel and Chinneck [2007], the foundation constraint is the active constraint that by some measure has the most impact on the current LP-relaxation solution. The active constraints include all equality constraints, and all inequality constraints that are tight. In the spirit of the measures used by Patel and Chinneck [2007] to choose the branching variable, we tested various measures of the impact of an active constraint on the current LP solution. Given the measure, the active constraint having the highest impact is chosen as the foundation constraint.

Measures tested for this purpose included:

- The largest number of candidate variables in the active constraint.

- The largest number of integer variables in the active constraint.
- The highest sum of candidate-variable absolute coefficients in the active constraint
- The highest sum of integer-variable absolute coefficients in the active constraint.
- The constraint having the largest feasibility-gap sum, i.e. the cumulative sum of the absolute difference between the current value of each candidate variable and the closest integer value, for all candidate variables in the active constraint.

Several combinations of two measures were tested: in each case a primary measure of ranking the active constraints and a secondary measure to break ties. Overall the most successful combination was (i) the number of candidate variables in the active constraint, with ties broken by (ii) the highest sum of candidate variable absolute coefficients. Test results are summarized in Appendix B. For full details on all algorithm design experiments see Mahmoud [2012].

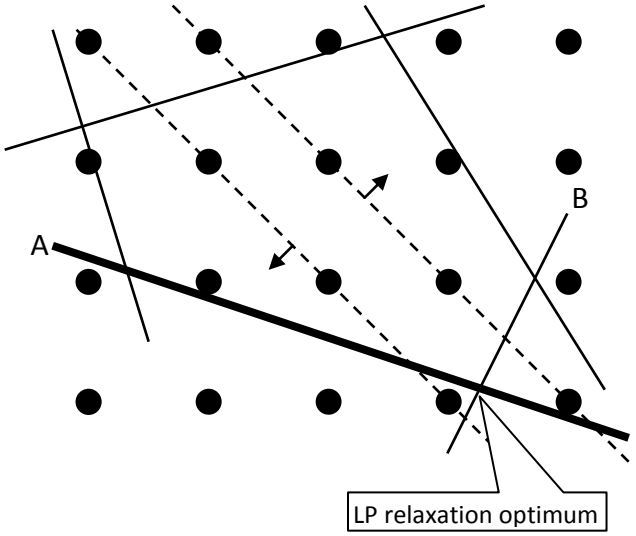

Figure 1: Approximately parallel 45-degree general disjunction.

2.2 Forming a 45-Degree General Disjunction

The general disjunction is formed to be either as parallel or as perpendicular as possible to the foundation constraint. Branching as parallel as possible to an equality foundation constraint is ineffective. There may be no feasible intersection at all between the original equality and the disjunction inequalities, or if one side of the disjunction lies exactly on the original equality then that side of the disjunction is ineffective and the other side yields an infeasible LP. To avoid these problems, we use approximately perpendicular general disjunctions for equality foundation constraints.

2.2.1 Approximately Parallel Disjunctions for Inequality Foundation Constraints

Approximately parallel general disjunctions work well with inequality foundation constraints. One side of the disjunction often forms a small feasible region that solves

quickly and both sides have nice edges that pass through integer lattice points, as illustrated in Figure 1.

An approximately parallel 45-degree general disjunction is formed by setting the signs of the coefficients in the disjunction equation to be the same as the signs of the corresponding coefficients in the foundation constraint. For example, if the foundation constraint is $2x_1 - 7x_2 + 15x_3 \leq 30$, where x_1, x_2, and x_3 are nonnegative integer variables with LP-relaxation values of 4.6, 3.2, and 2.88 respectively, then the approximately parallel 45-degree disjunction will be as follows:

Down branch: $x_1 - x_2 + x_3 \leq \lfloor 4.6 - 3.2 + 2.88 \rfloor = 4$

Up branch: $x_1 - x_2 + x_3 \geq \lfloor 4.6 - 3.2 + 2.88 \rfloor + 1 = 5$

2.2.2 Approximately Perpendicular Disjunctions for Equality Foundation Constraints

Creating an approximately perpendicular 45-degree general disjunction is not as simple. It is accomplished by creating a general disjunction that is exactly perpendicular to the nearly parallel 45-degree general disjunction, but there are many ways to do this. In the previous example, the normal vector of the nearly parallel 45-degree disjunction is v_{paral} = (1,-1,1), i.e. the set of non-zero variable coefficients of the disjunction equation. A vector v_{perp} is perpendicular to v_{paral} if their dot product is zero. For v_{paral} = (1,-1,1), there are many possible perpendicular vectors including (-1,-1,0), (0,1,1), (-1,0,1) and numerous others.

There are two main decisions in constructing v_{perp}. In general, we prefer to have as many nonzeros as possible in v_{perp} in order to involve as many variables as possible in the disjunction. If there is an odd number of nonzeros in v_{paral}, then at least one coefficient in v_{perp} must be set to zero. Hence we first choose a variable whose coefficient will be zero, and then determine how to set the signs of the remaining nonzero coefficients.

When the number of nonzeros is odd, we identify the variable that has the least impact on the MILP solution and set its coefficient to zero. General principles for the selection of the least-impact variable are:

- Continuous variables have less impact than integer variables.
- Non-candidate integer variables have less impact than candidate variables.
- Candidate variables having smaller integer infeasibilities have less impact than candidate variables having larger integer infeasibilities. Benichou et al. [1971] noted this effect.
- Variables having smaller absolute coefficients in the foundation constraint have less impact.
- Variables that are present in fewer active constraints at the current LP relaxation optimum have less impact than variables that appear in more active constraints.

Using these principles, the variable to set to zero is selected as follows:

- If there is at least one continuous variable then:

    - Choose the first continuous variable that has the smallest absolute coefficient in the foundation constraint.
- Else (there are only integer variables):
    - If there is at least one non-candidate integer variable then:
        - Choose the first non-candidate that has the smallest absolute coefficient in the foundation constraint.
    - Else (there are only candidate integer variables):
        - Choose the candidate variable that appears in the fewest active constraints.

Once a coefficient has been set to zero, if necessary, the remaining coefficients must be set to +/-1. There is a combinatorially explosive number of ways to do this. We use a simple procedure: switching the sign of every second non-zero coefficient in the parallel normal vector. There are two ways to do this: switching sign on odd counter or on even counter. Our test results (Appendix B) show that the two choices are about the same. We choose to switch on odd counter. The right-hand side constant is set in a manner analogous to that for approximately parallel general disjunctions.

2.3 Choosing the Branching Direction for General Disjunctions

For every disjunction, whether axis-parallel or general, there are two possible branching directions, up or down, and various heuristics for making this decision. We considered several branching direction selection heuristics: (i) branch to the side of the disjunction that increases or decreases the value of the disjunction function, (ii) branch to the side of the disjunction that is closer to or farther from the current LP-relaxation optimum point, and (iii) branch in the satisfying direction of an inequality foundation constraint or in the opposite direction.

Branching farther from the LP-relaxation optimum point is in the spirit of the advice given by Pryor and Chinneck [2007] to branch in the direction that has the smallest chance of providing a feasible solution. Branching in the satisfying direction of the disjunction constraint further constricts the feasible region and hence has a similar effect. In both cases, many candidate variables are forced to change in order to satisfy the disjunction constraint.

Branching in the satisfying direction is ineffective when the foundation constraint is itself a 45-degree inequality (such as a multiple-choice constraint). In this case, one of the disjunction inequalities lies exactly on the foundation constraint, and hence has no effect. For this reason, we can offset the disjunction by adjusting the constant by 1, which further constricts the feasible region.

Our test results (Appendix B) support the following decisions:

- For parallel disjunctions: branch in the satisfying direction of the inequality, with a one unit offset. For example, if the parallel disjunction pair is $x_1 - x_2 + x_3 \leq 4$ and $x_1 - x_2 + x_3 \geq 5$ as in the example in Sec. 2.2.1, then we first branch by adding $x_1 - x_2 + x_3 \leq 3$ and may later backtrack to add $x_1 - x_2 + x_3 \geq 4$.

- For perpendicular disjunctions: branch farther from the LP relaxation optimum point.

2.4 When to Include a General Disjunction

Most researchers caution that the effort required to generate and use general disjunctions reduces overall speed [Mahajan and Ralphs, 2009], even though general disjunctions force multiple candidate variables to change at once, and hence more often lead to smaller search trees [Owen and Mehrotra 2001; Mahajan and Ralphs 2009; Cornuéjols, Liberti and Nannicini 2011; Karamanov and Cornuéjols 2011]. Because each general disjunction adds a constraint to the LP sub-problem at the child node and all of its descendents, if used too frequently the MILP model can grow considerably, increasing the time needed for each sub-problem and slowing the overall solution [Karamanov and Cornuéjols, 2011]. Axis-parallel disjunctions change only the bounds on candidate variables, so the LP sub-problems at the child nodes are no harder to re-optimize.

Given this, we use axis-parallel disjunctions by default, inserting general disjunctions only when it seems advantageous to do so. We consider two questions in deciding whether to insert a general disjunction: (i) Has the solution progress stalled while using axis-parallel branching? (ii) Are there sufficient candidate variables to make a general disjunction worthwhile?

2.4.1 Detecting Stalling

While branching using only axis-parallel disjunctions, we monitor solution progress by means of two basic metrics: (i) the number of candidate variables, and (ii) the *infeasibility sum* at the current LP relaxation optimum (the sum of the absolute differences between the current value of each candidate variable and its closest integer value over all candidate variables [Ilog 2009c]). If either or both of these measures increase a given number of times in a row, this indicates that axis-parallel disjunctions are not working effectively, so a single general disjunction is generated. Counting the increases and decreases in these measures is one possible trigger for including a general disjunction; another is to observe the increase or decrease in the ratios of these measures between the parent and child nodes.

The candidate variable count is incremented by 1 whenever the number of candidate variables at a child node increases over the number at the parent node. It is decremented by 1 whenever the number of candidate variables at a child node decreases or stays the same as the number at the parent node. The same concept applies to the infeasibility sum count.

We selected two different combinations of these counts to detect stalling:

- Trigger A: both the candidate variable count and the infeasibility sum count increase for 3 iterations in a row.
- Trigger B: either the candidate variable count or the infeasibility sum count increase for 10 iterations in a row.

We tested a variety of other levels and combinations of these count measures, as well as ratios of their values, as shown in Appendix B.

2.4.2 Sufficient Candidate Variables

If there are too few candidate variables, then the cost of inserting and maintaining a general disjunction outweighs its benefit. Consequently, a general disjunction is not triggered (even when the solution process is stalling) if the number of candidate variables is below a specified minimum. We require at least 60 candidate variables in order to trigger a general disjunction. Tuning experiments to arrive at this figure are shown in Appendix B.

2.4.3 General Disjunction at the Root Node

There is no solution history to consider at the root node, so the concept of stalling does not apply. Hence we simply include a general disjunction after the root node solution if the number of candidate variables exceeds the specified minimum of 60.

3. Experimental Setup

We conduct experiments in the Appendix to tune the configurations of our general disjunction algorithms, and in Section 5 to compare the tuned configurations to existing methods. The experimental conditions are identical throughout, except as otherwise noted.

3.1 Software

The state-of-the-art commercial solver Cplex 12.1 [Ilog 2009a, 2009b, 2009c] is used in all the experiments reported here. It is used in 3 different ways:

i. As the MILP framework in which our new general disjunction algorithms are implemented, labeled *GD-A* and *GD-B* for the two variants based on the two trigger criteria for including a general disjunction. Parameter settings match those of *Baseline_Cplex* or *Default_Cplex*, as appropriate to the comparison.

ii. As a basic B&B solver with most internal heuristics turned off (*Baseline_Cplex*).

iii. As a state-of-the-art comparator with most parameters at default settings (*Default_Cplex*).

In all three solvers, all parameters are at their default values with these exceptions (the corresponding Cplex parameter is shown in brackets):

- Number of MILP solutions to find before stopping (*CPX_PARAM_INTSOLLIM*): 1. The goal is to attain only the first MILP-feasible solution.
- Node selection strategy (*CPX_PARAM_NODESEL*): Depth-first.
- Time Limit (*CPX_PARAM_TILIM*): Two hours (7200 seconds) for the tuning experiments in the Appendix, eight hours (28,800 seconds) for the final testing experiments presented in Section 5.
- Default number of parallel threads invoked by any CPLEX parallel optimizer (*CPX_PARAM_THREADS*): all experiments run on single threads.
- MIP emphasis (*CPX_PARAM_MIPEMPHASIS*): Empasis on MILP-feasibility.

The settings for MIP emphasis and node selection strategy promote the early attainment of the first MILP-feasible solution. We chose depth-first node selection since this more frequently required fewer simplex iterations to reach the first MILP-feasible solution vs. Cplex default node selection, even for *Default_Cplex*. Solutions are run until the first MILP-feasible solution is obtained or until the time limit is reached.

In *Baseline_Cplex*, most of the node pre-processing and node heuristics in the underlying Cplex solver are also turned off:

- Pre-solving (*CPX_PARAM_PREIND*): off.
- Aggregation (CPX_PARAM_AGGIND): off.
- Internal Node Heuristic (CPX_PARAM_HEURFREQ): off.
- Cut generation (*CPX_PARAM_CLIQUES*, *CPX_PARAM_COVERS*, *CPX_PARAM_CUTPASS*, *CPX_PARAM_CUTSFACTOR*, *CPX_PARAM_DISJCUTS*, *CPX_PARAM_FLOWCOVERS*, *CPX_PARAM_FLOWPATHS*, *CPX_PARAM_FRACCUTS*, *CPX_PARAM_GUBCOVERS*, *CPX_PARAM_IMPLBD*, *CPX_PARAM_MCFCUTS*, *CPX_PARAM_MIRCUTS*, *CPX_PARAM_ZEROHALFCUTS*): off.

Baseline_Cplex is intended as a relatively simple MILP solver to allow the evaluation of our algorithm alternatives without the confounding effects of the many interacting heuristics available in an advanced solver like Cplex.

The settings in *GD-A* and *GD-B* are the same as those in whichever version of Cplex (*Baseline* or *Default*) it is being compared to. A C program manages the interface between the Cplex solver and our algorithms in *GD-A* and *GD-B* using the Cplex C API [Ilog 2009b]. Interfacing with the solver during the MILP solution is done via callback routines that are provided by Cplex. Since a few solver parameters such as *dynamic search* [Ilog 2009a] are switched off by default when a callback routine is invoked, *Baseline_Cplex* and *Default_Cplex* are provided with dummy callback routines which do nothing.

3.2 Hardware

Three computers running Microsoft Windows XP were used in the experiments:

- Two machines having Intel Core 2 CPUs running at 2.40GHz and 3GB of RAM,
- One machine having an Intel Core 2 Quad CPU running at 2.40GHz and 8GB of RAM .

All methods included in a given experiment are run on the same computer to allow for a fair comparison. It is also possible to compare methods run on different computers as the specifications for the cores in all three computers are similar.

3.3 Test Models

350 MILP models of different types representing a variety of real-world problems were taken from MIPLIB2010 [Koch et al. 2011]. These were divided into tuning and testing sets, and further subdivided in various other ways to suit the purpose at hand. The

tuning set consists of 127 models that are solved by *Baseline_Cplex* in less than 1 hour, and is used in Appendix B to tune the design of the new algorithms.

The testing set of 223 models is used in Section 5 to compare our general disjunction algorithms to existing solvers. It consists of the remaining 112 models solved by *Baseline_Cplex* in less than 1 hour (the *easy1* set, see Appendix A.4), plus the 111 models that required more than 1 hour of solution time using *Baseline_Cplex* (the *hard1* set, Appendix A.5). For a second experiment, the 223 testing set models are subdivided according to their solution time using *Default_Cplex*: 156 models solve in less than 1 hour using *Default_Cplex* (the *easy2* set), and the remaining 67 models require more than 1 hour (the *hard2* set).

3.4 Metrics

Three metrics are commonly used to compare MILP algorithms: total CPU time, number of simplex iterations, and the number of solved nodes. The number of solved nodes mainly captures the average memory consumption of an algorithm and can sometimes be related to the total computational effort [Patel and Chinneck, 2007], but it is not always a good measure of speed. A solution that has many solved nodes may finish in less time than one with fewer solved nodes because the LPs in successive nodes are very similar. This is the case for depth-first node selection, which allows an advanced start for the next LP sub-problem, yielding a solution in just a few iterations, whereas selecting a node elsewhere in the tree does not have this advantage.

Total CPU time is the most important metric from the user's point of view, but it is difficult to measure in a consistent manner in modern multi-core machines. There is no guarantee that the timestamp counters of the different cores on a single computer are synchronized, and the CPU speeds may be adjusted for power-saving purposes depending on the load on the cores and periodic tasks carried out by the operating system. For these reasons, we instead rely on the number of simplex iterations as the main metric for computational speed. It is a good proxy for computational time provided there is no significant computational effort outside of the LP solutions. Since our algorithms use axis-parallel disjunctions by default and only occasionally use general disjunctions, the non-simplex computational time is mainly for creating and applying general disjunctions, and hence depends heavily on the number of general disjunctions created. It also depends on the method used for choosing the foundation constraint and for creating the general disjunctions. The average time spent in the callback routines for setting up general disjunctions is about 0.3% of the total time spent solving the set of models in Appendix A.4. The maximum percentage of time spent in the callback routines for any model is 4%. These times are negligible, so we use the more reproducible number of simplex iterations as the main metric for comparing algorithm speed. This also allows us to use heterogeneous machines for the experiments.

Algorithms in a given experiment are compared by means of *performance profiles* [Dolan and Moré, 2002] on the number of simplex iterations. For a given metric, a performance profile plots the ratio of a given method's simplex iterations to the fewest simplex iterations used by any method on each model, versus the percentage of models for which the method attains that ratio or better. This eliminates problems such as skewed averages due to a minority of models dominating the results. Performance

profiles also provide a way of comparing the *robustness* of algorithms, defined here as the ability to solve models within the specified time limit (shown by the maximum height achieved on the vertical axis).

The number of backtracks is also recorded. This shows the number of dives in the B&B tree required before reaching the first MILP-feasible solution. Zero backtracks is ideal, meaning that a MILP-feasible solution is found at the first leaf reached. When using depth-first node selection, the number of backtracks can also be thought of as the number of infeasible leaf nodes that are solved before a MILP-feasible node is reached.

The *optimality gap* (the difference between a known solution and a value that bounds the best possible solution [Mathematical Programming Glossary 2011]) is used to measure the quality of the MILP solution returned by an algorithm.

4. Tuning the Disjunction Algorithm

The elements in our general disjunction algorithms are (i) deciding whether or not to include a general disjunction (at any node in the tree, with special rules at the root node), (ii) selecting the foundation constraint, (iii) deciding whether to construct a disjunction that is approximately parallel or perpendicular to the foundation constraint, (iv) constructing the disjunction function, and (v) deciding the branching direction.

We conducted experiments to tune the rules for these decisions, based on several possibilities for each of the listed decisions. We first winnowed the possible decision methods by altering a single method within a reference combination of the methods. Details are given in Appendix B (Stage 1). The resulting best one or two methods for making each decision (summarized in Table 1) were then combined to form complete algorithms (see Appendix B, Stage 2), and the best of these complete algorithms are compared against a state-of-the-art MILP solver in Section 5.

Table 1 leads to eight combinations of the decision methods. Since the test models are solved by *Baseline_Cplex* in less than an hour, robustness (i.e. fraction of models reaching a feasible solution within the time limit) is the main metric. See Appendix B for the full analysis.

The two most promising variants are 2A-3B-5A and 2B-3B-5A. These are compared to a commercial MILP solver in Section 5. Since these two algorithms differ only in how the inclusion of a general disjunction is triggered (choice 2A or 2B), we label these as *GD-A* and *GD-B* in the sequel.

5. Comparison to the State of the Art

We compared *GD-A* and *GD-B* to a state-of-the-art commercial MILP solver Cplex 12.1 [Ilog 2009a, 2009b, 2009c] over the subset of MILP problems that make up the testing set (see Appendices A.4 and A.5). We focus on the results for the hard MILP models (in the *hard1* and *hard2* sets) since these are the ones that generally trigger the use of a general disjunction, and are also the ones that can most benefit from them. We impose a time limit of eight hours on each solution.

Table 1: Summary of top methods for making each decision in a complete algorithm.

	Decision	Method
1	Which type of disjunction to use at the root node?	A general disjunction is generated if the condition on the minimum number of candidate variables is satisfied.
2	When to perform a general disjunction?	***2A*****:** The number of candidate variables and the infeasibility sum both increase simultaneously more than 3 times in a row, and there are more than 60 candidate variables
		2B**:** Either the number of candidate variables or the infeasibility sum (or both) increase more than 10 times in a row, and there are more than 60 candidate variables.
3	Which active constraint to choose as the foundation constraint?	***3A*****:** Choose the active constraint with the largest number of integer variables. Break ties using the highest sum of integer-variable absolute coefficients.
		3B**:** Choose the active constraint with the largest number of candidate variables. Break ties using the highest sum of candidate-variable absolute coefficients.
4	How to construct a 45-degree general disjunction that is approximately perpendicular to an equality constraint?	Switch sign of non-zero coefficients on even counter. If an odd number of non-zeros exist, then use *SC-var* to set one of the non-zero coefficients to zero in the disjunction function
5	Which branching direction to use for parallel general disjunctions?	***5A*****:** Satisfying direction of the branching constraint (with offset).
		5B**:** Farther from the LP-relaxation optimum point (without offset)
6	Which branching direction to use for perpendicular general disjunctions?	Farther from the LP-relaxation optimum point

5.1 Experiment 1: All Heuristics Off

In Experiment 1 we compare our two general disjunction algorithms against *Baseline_Cplex* in which all heuristics are turned off, providing a straightforward comparison without the confounding effects of the interacting heuristics. All algorithms, including *Baseline_Cplex*, exceeded the time limit for 62 of the 111 *hard1* models listed in Appendix A.5, leaving 49 models that are solved by at least one of the algorithms within the time limit (listed in *italics* in Appendix A.5). Figure 2 compares the performances of the algorithms over these 49 models.

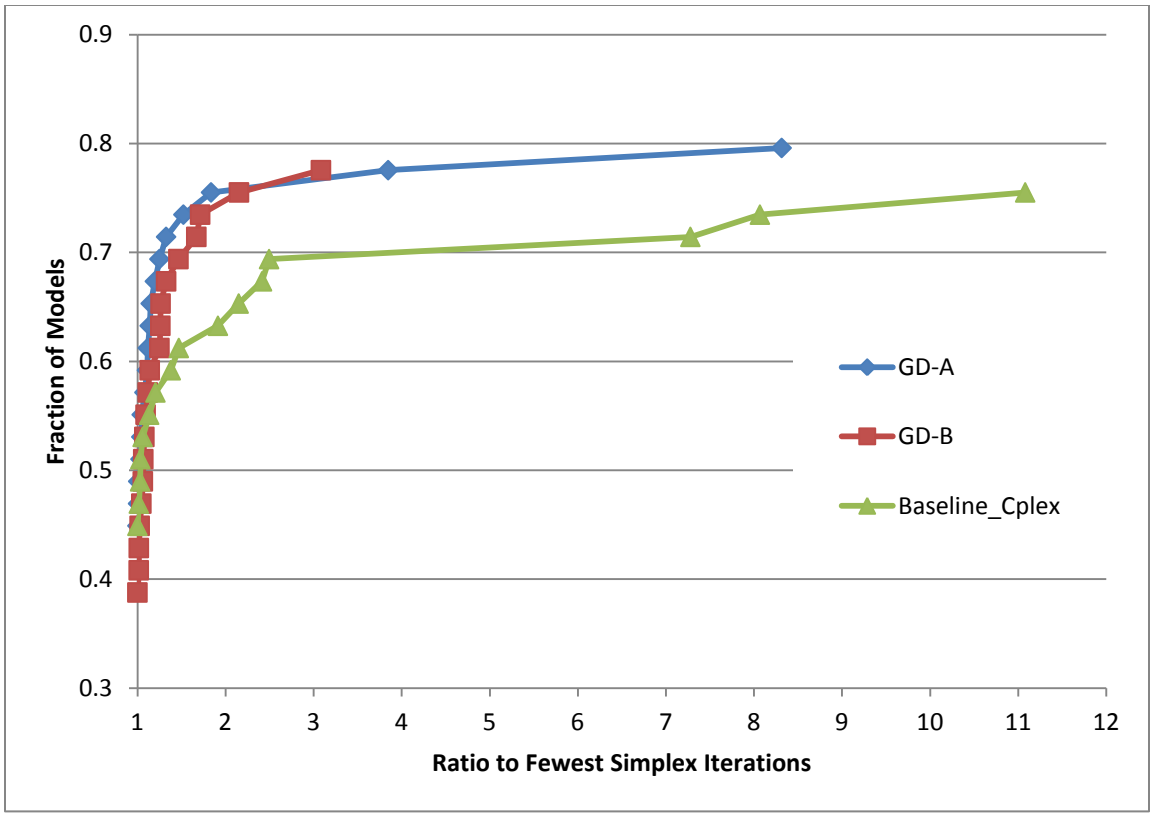

Figure 2: Comparing to *Baseline_Cplex* over the *hard1* models

Both of the new algorithms significantly outperform *Baseline_Cplex* in terms of speed as measured by simplex iterations, and both are also more robust than *Baseline_Cplex*, solving more models within the time limit (39 for *GD-A*, 38 for *GD-B* and 37 for *Baseline_Cplex*). *GD-A* and *Baseline_Cplex* are the fastest for 22 (44.9%) of the models, while *GD-B* is fastest for 19 (38.8%). The node counts for the new algorithms are also generally smaller than for *Baseline_Cplex*.

Table 2 shows the number of models solved by each of the new algorithms at a faster or slower rate than *Baseline_Cplex* (provided that both methods arrive at a feasible solution) and the number of models solved by the new algorithms within the time limit that are not solved by *Baseline_Cplex*, and vice versa. Both new algorithms achieve better results with respect to speed and robustness vs. *Baseline_Cplex*, with *GD-A* bettering *GD-B* by a small margin.

Table 2: Comparing *GD-A* and *GD-B* to *Baseline_Cplex*.

	Number of Models	
	GD-A	*GD-B*
Solved at least as fast as *Baseline_Cplex*	17	16
Solved slower than *Baseline_Cplex*	11	13
Solved that *Baseline_Cplex* did not solve	11	9
Not solved but are solved by *Baseline_Cplex*	9	8

Figure 3 shows a comparison of the algorithms over the 112 models in the *easy1* set. Results are very similar for about 80% of the models, but *Baseline_Cplex* is a little better for the remainder. This is partly due to selection bias (the models are selected because they can be solved by *Baseline_Cplex* within the time limit), but it also illustrates that our particular methods for inserting general disjunctions may not be worth the trouble on smaller models.

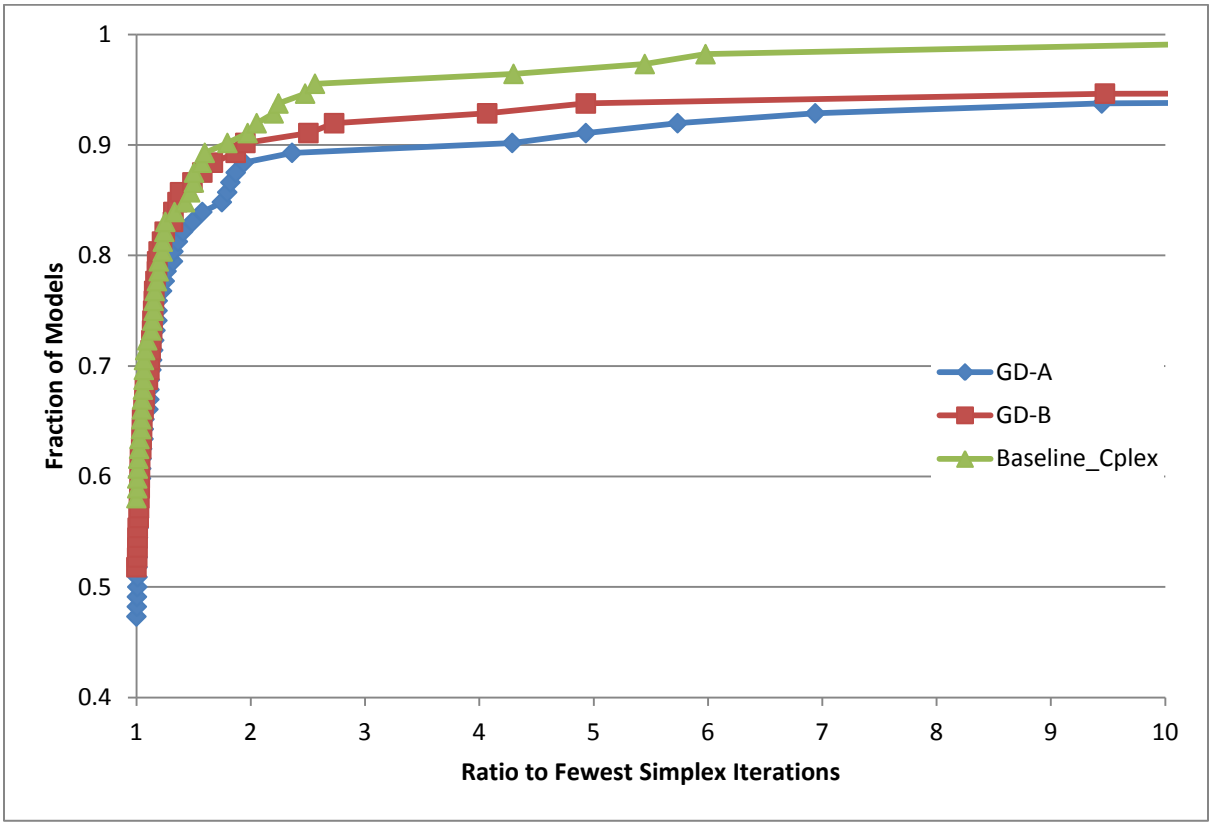

Figure 3: Comparing to *Baseline_Cplex* over the *easy1* models

The average optimality gaps of the first MILP-feasible solutions for *GD-A* and *GD-B* were compared against *Baseline_Cplex* for all problems in the *hard1* and *easy1* testing set in which the three compared algorithms all completed successfully. On average, the optimality gap for *GD-A* and *GD-B* is better than (or equal to) that of *Baseline_Cplex* for 59 and 60% of the models, respectively.

On average, the number of backtracks performed by *GD-A* or *GD-B* is less than (or equal to) those performed by *Baseline_Cplex* in 61% of the models in the *hard1* and *easy1* sets.

Statistics on the number of general disjunctions triggered by *GD-A* and *GD-B* over the models in the *easy1* and *hard1* sets are summarized in Table 3. A larger fraction of models in the *easy1* set trigger no general disjunctions than in the *hard1* set. In contrast, a smaller fraction of models in the *easy1* set trigger two or more general disjunctions than in the *hard1* set. This shows that general disjunctions are triggered more frequently on harder problems that have larger numbers of candidate variables and more frequently stall while branching on variables.

Table 3: Number of general disjunctions triggered in the *easy1* and *hard1* sets.

	% of models that trigger					
	0 Disjunctions		1 Disjunction		2+ Disjunctions	
	easy1	*hard1*	*easy1*	*hard1*	*easy1*	*hard1*
GD-A	17.0	7.2	48.2	26.1	34.8	66.7
GD-B	17.0	6.3	60.7	44.1	22.3	49.6

Figure 4 illustrates how general disjunctions can speed up the attainment of the first MILP-feasible solution. It shows the number of candidate variables for *Baseline_Cplex* and for *GD-B* on the dive in the B&B tree that leads to the first MILP-feasible solution for the *Satellites1-25* model from the *easy1* test set. *GD-B* triggers the inclusion of four general disjunctions (including one at the root node) during the solution: their effects are seen in the sudden drops in the number of candidate variables after a period of stalling. *Baseline_Cplex* requires more nodes to solve the problem and stalls frequently, taking 12 minutes to attain a feasible solution vs. less than a minute for *GD-B*.

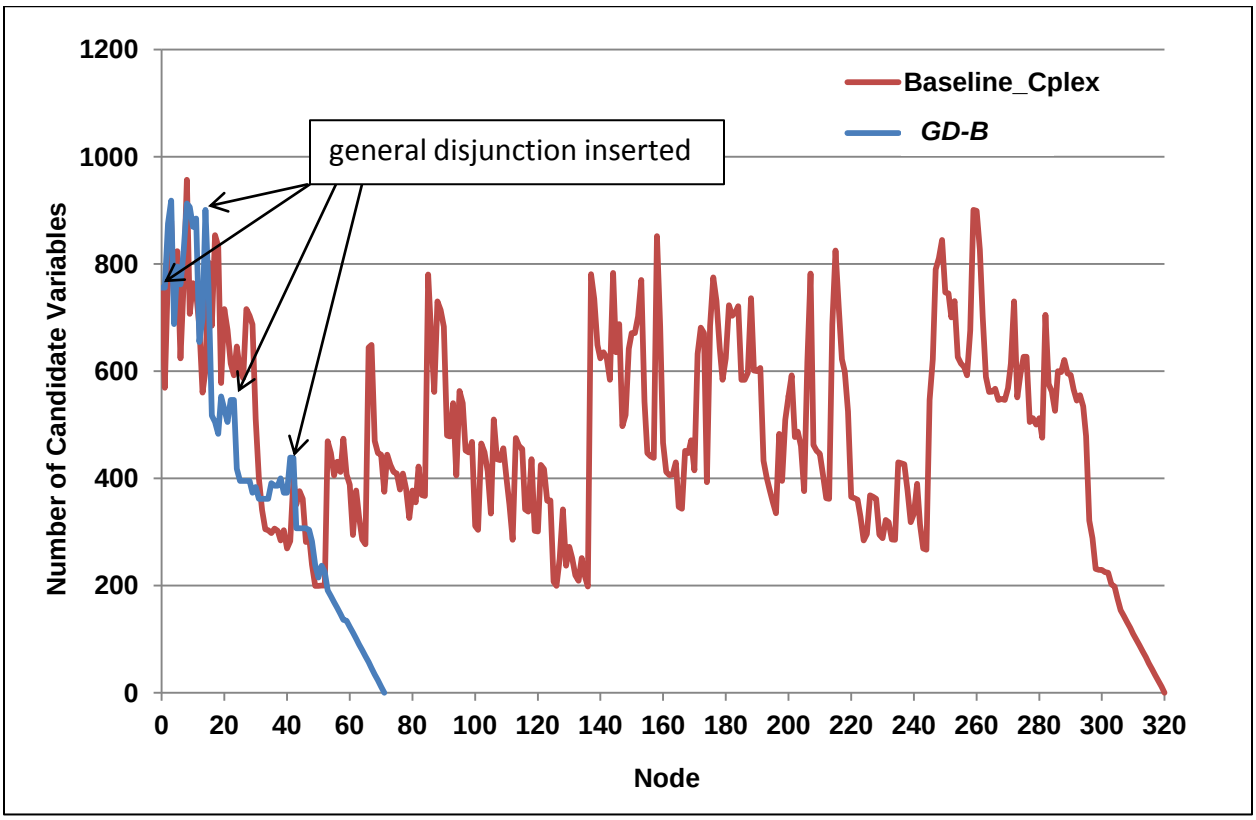

Figure 4: Number of candidate variables for *Baseline_Cplex* vs *GD-B* for *Satellites1-25*.

5.2 Experiment 2: Default Heuristics On

This experiment compares *GD-A* and *GD-B* vs. Cplex with most *Cplex* heuristics at their default settings in both *Default_Cplex* and the *GD* variants. This provides supplementary results examining whether the new algorithms have a negative impact when combined with the Cplex internal heuristics.

The root node heuristics as well as internal node heuristics and cut generation heuristics (see Section 3.1 for complete list) are all turned on by default. Enabling these does not necessarily guarantee a faster MILP solution, and can actually have a negative impact. For example, models *ns1686196*, *ns1856153*, and *ns894244* are solved within the time limit by *Baseline_Cplex* but are prematurely terminated by *Default_Cplex* before reaching an integer-feasible solution.

Our heuristics trigger the inclusion of a general disjunction only if variable branching is performing poorly, and hence generally perform at least as well if not better than *Default_Cplex*. If variable branching is doing well, then a general disjunction may never be triggered.

General disjunctions have more impact when the problem is difficult to solve, thus we focus on their performance on the *hard2* set. Figure 5 compares the algorithms over the 21 models from the *hard2* set that are solved by at least one of the three algorithms within the time limit. *GD-A* and *GD-B* improve significantly over *Default_Cplex,* solving 19 models (90.5%) within a ratio to best of around 2.5, versus 14 (70%) for *Default_Cplex*. Both methods are more robust than *Default_Cplex*, solving 1 more model within the time limit.

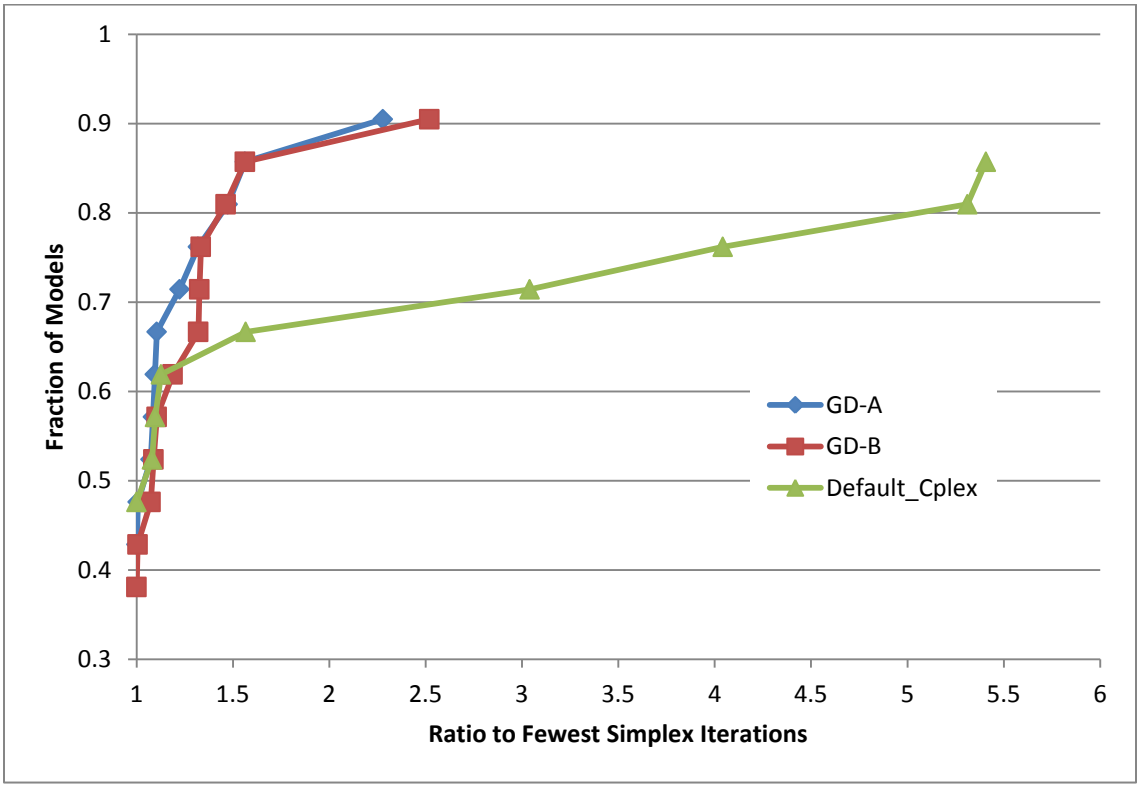

Figure 5: Comparing to *Default_Cplex* over the *hard2* models.

Figure 6 compares *GD-A* and *GD-B* to *Default_Cplex* over the 156 models in the *easy2* set. As in Experiment 1, results are similar for a substantial portion of the models (around 87%), but *Default_Cplex* is a little better for the rest. While *Default_Cplex* reaches a feasible solution for all 156 models (due to the way in which the models are selected), GD-A and GD-B are almost as good, finding feasible solutions for 152 and

153 models, respectively, within the time limit. This again shows that our particular methods are best used for larger and more difficult MILP models.

5.3 Node Selection Experiments

We also evaluated the algorithms vs. Cplex using its default node selection method, but with all other parameter settings as given for *Default_Cplex* in Sec. 3.1 to provide maximum encouragement for finding a first feasible solution quickly. Using slightly different subsets of the models in the *hard2* and *easy2* sets, we obtained similar results when comparing (i) *GD-A* with depth-first search (DFS), (ii) *GD-B* with DFS, and (iii) *Default_Cplex* with default node selection. *GD-A* with DFS and *GD-B* with DFS outperform *Default_Cplex* with default node selection for both the *hard2* and *easy2* subsets that we used, based on simplex iterations. Note that both new algorithms outperformed Cplex with default node selection on the *easy2* subset. Since our experiments show that *Default_Cplex* with DFS node selection generally requires fewer simplex iterations than *Default_Cplex* with default node selection, these results are not surprising.

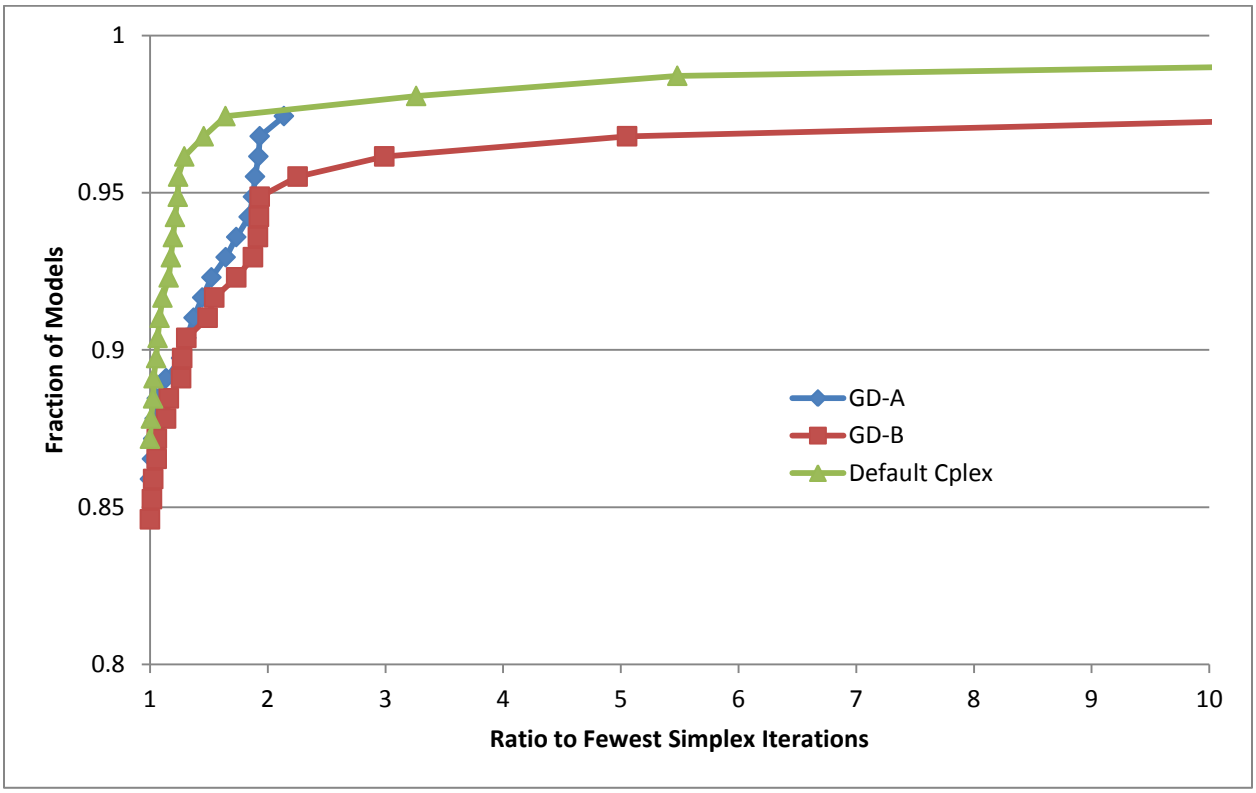

Figure 6: Comparing to *Default_Cplex* over the *easy2* models.

5.4 Other Experiments

Preliminary experiments not reported here, but similar to those in Sections 4 and 5, were also conducted using the open-source GLPK 4.38 MILP solver [Makhorin 2009]. Several general disjunction algorithms were compared with default GLPK with parameters set to encourage early integer feasibility, and with Method A [Patel and Chinneck 2007] for choosing the branching variable, one of the fastest algorithms for

reaching integer feasibility quickly. Significant speed improvements over both of these methods were observed. The qualities of the solutions returned using general disjunctions were also better than GLPK default on average 60% of the time.

6. Conclusions and Future Work

This paper demonstrates for the first time that general disjunctions can speed the identification of the first integer-feasible solution in large and difficult MILP models. The key contributions are (i) identifying some conditions under which it is most likely advantageous to insert a general disjunction, and (ii) proposing efficient ways of constructing effective general disjunctions. We have developed specific algorithms for these purposes, but there is undoubtedly much scope for future improvements. Our current methods simply constitute the first proof of concept.

We discovered that it is generally worthwhile to insert a general disjunction only when conventional axis-parallel branching is stalling and there are sufficient candidate variables. We developed new measures to recognize stalling by monitoring changes in the number of candidate variables and the infeasibility sum, and determined that a relatively large minimum number of candidate variables should be present.

Several general principles proved useful.

- Conventional branching on variables is preferable except under certain conditions. Specifically, general disjunctions are probably worthwhile when branching on variables is stalling and there are numerous candidate variables which can be influenced simultaneously by a general disjunction.
- A general disjunction should be constructed to impact as many candidate variables as possible. This is valuable when the goal is to force change in these variables. Our methods base the general disjunction on an active constraint that is known to influence numerous candidate variables.
- Limiting the disjunction coefficients to +1, -1, or 0 limits the infinite range of choices for the disjunction equation, and the resulting "45-degree" disjunctions have appealing qualities including alignment with the lattice points and having an empty interior.
- It is useful to branch first in the direction that forces the most change.

Our two new algorithms based on these principles (*GD-A* and *GD-B*) demonstrate improvements over a state-of-the-art commercial MILP solver when applied to difficult models. They generally reach the first feasible solution more quickly, and more often provide a better optimality gap at the first feasible solution.

There are many opportunities for potential improvement in these first algorithms:

- Instead of choosing a single foundation constraint from the set of active constraints, an artificial foundation constraint that is based on a set of active constraints can be created, e.g. by combining those active constraints that do not have any candidate variables in common. The resulting general disjunction would affect even more candidate variables.

- Determining whether a general disjunction is worthwhile by monitoring the time improvement realized by a general disjunction versus the time penalty for creating and using it. If the speed improvement outweighs the time cost, then continue using general disjunctions as opportunities arise, otherwise stop using them entirely. This idea can be extended by keeping track of the performance gains achieved by branching on particular foundation constraints and then using this information to break the ties between equally-ranked foundation constraints.
- Explore the use of perpendicular general disjunctions for inequality foundation constraints. This may be especially useful when the coefficients in the resulting general disjunction are the same as those in the foundation constraint (e.g. in "multiple choice" constraints).
- Explore new ways of monitoring the progress of the MILP solution to trigger the inclusion of a general disjunction, such as the growth rate in the number of active nodes.

7. References

Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G., and Vincent, O. Experiments in mixed-integer linear programming. *Mathematical Programming* **1**, 76-94 (1971).

Chinneck, J.W. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*. International Series in Operations Research and Management Sciences 118, Springer, Boston, MA (2008).

Cornuéjols, G., Liberti, L. and Nannicini, G. Improved Strategies for Branching on General Disjunctions. *Math. Program.* **130**, no. 2, 225-247 (2011).

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

Fischetti, M., Glover, F. and Lodi, A. The Feasibility Pump. *Math. Program.* **104**, 91-104 (2005).

ILOG Corporation. Cplex 12.1 Parameters Reference Manual (2009a).

ILOG Corporation. Cplex 12.1 C API Reference Manual (2009b).

ILOG Corporation. Cplex 12.1 User's Manual (2009c).

Karamanov, M., Cornuéjols, G., Branching on General Disjunctions. *Math. Program.* **128**, no. 1-2, 403-436 (2011).

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K. “MIPLIB 2010”, *Mathematical Programming Computation*, **3**, no. 2, 103-163 (2011).

Mahajan, A. and Ralphs, T.K. Experiments with Branching using General Disjunctions. *Operations Research and Cyber-Infrastructure* **47**, 101-118 (2009).

Mahajan, A. and Ralphs, T.K. On the Complexity of Selecting Disjunctions in Integer Programming. *SIAM J. on Optimization* **20**, 2181–2198 (2010).

Mahmoud, H. Achieving Integer Feasibility Quickly by Alternating Axis-Parallel and General Disjunctions, M.A.Sc. thesis, Systems and Computer Engineering, Carleton University (2012).

Makhorin, A., GNU Linear Programming Kit Reference Manual Version 4.38, 2009, http://www.gnu.org/software/glpk/, [Accessed December 2011].

Mathematical Programming Glossary, INFORMS Computer Society, http://glossary.computing.society.informs.org/index.php?page=O.html, [Accessed December 2011].

Owen, J.H. and Mehrotra, S. Experimental Results on using General Disjunctions in Branch-and-Bound for General-Integer Linear Programs. *Comput. Optim. Appl.* **20**, 159–170 (2001).

Patel, J. and Chinneck, J.W. Active-Constraint Variable Ordering for Faster Feasibility of Mixed Integer Linear Programs. *Math. Program.* **110**, 445-474 (2007).

Pryor, J. and Chinneck, J.W. Faster Integer-Feasibility in Mixed-Integer Linear Programs by Branching to Force Change. *Computers & Operations Research* **38**, 1143-1152 (2011).

Wojtaszek, D.T. and Chinneck, J.W. Faster MIP Solutions via New Node Selection Rules. *Computers & Operations Research* **37**, 1544-1556 (2010).