Mathematical Programming manuscript No.
(will be inserted by the editor)

Branching on General Disjunctions

Miroslav Karamanov · **Gérard Cornuéjols**

July 2005 / revised November 2008, August 2009

Abstract This paper considers a modification of the branch-and-cut algorithm for Mixed Integer Linear Programming where branching is performed on general disjunctions rather than on variables. We select promising branching disjunctions based on a heuristic measure of disjunction quality. This measure exploits the relation between branching disjunctions and intersection cuts. In this work, we focus on disjunctions defining the mixed integer Gomory cuts at an optimal basis of the linear programming relaxation. The procedure is tested on instances from the literature. Experiments show that, for a majority of the instances, the enumeration tree obtained by branching on these general disjunctions has a smaller size than the tree obtained by branching on variables, even when variable branching is performed using full strong branching.

Keywords Branch and bound · Branch and cut · Branching · Split disjunctions

Mathematics Subject Classification (2000) 90C10

Introduction

Branch-and-cut is the most widely used algorithm for solving Mixed Integer Linear Programs (MILP). Its performance improved by several orders of magnitude in

This work was supported in part by NSF grant CMMI-0653419, ANR grant BLAN06-BLAN-0375, and ONR grant N00014-09-1-0133.

Miroslav Karamanov
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, 15213
Tel.: 704-619-3248
Fax: 980-387-9500
E-mail: miroslav.k@gmail.com
Present address: Bank of America, Charlotte, NC 28255, NC1-003-01-24

Gérard Cornuéjols
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, 15213
E-mail: gc0v@andrew.cmu.edu

the last decade due to advances in hardware but mostly due to modifications in the algorithm [11, 19]. In this paper, we propose a modification in the branching routine.

One of the important decisions made in the branch-and-cut algorithm is the choice of a branching object. Traditionally, in general purpose MILP solvers, branching objects are variables—the "best" candidate is chosen among the integer variables that have a fractional value in the current optimal solution of a linear programming relaxation of MILP. If an integer-constrained variable, x_j, has a fractional value $\bar{x}_j$, we impose the constraint $x_j \leq \lfloor \bar{x}_j \rfloor$ in one of the children and $x_j \geq \lceil \bar{x}_j \rceil$ in the other. This can be viewed as adding the constraints $\pi^T x \leq \pi_0$ and $\pi^T x \geq \pi_0 + 1$, respectively, where $\pi = e_j$, the j-th unit vector, and $\pi_0 = \lfloor \bar{x}_j \rfloor$. We propose to use a general integer vector π and $\pi_0 = \lfloor \pi^T \bar{x} \rfloor$ for branching, where $\pi_i \in \mathbb{Z}$ if x_i is an integer variable and 0 otherwise. We call a disjunction *simple* when $\pi = e_j$, for some j, and *general* otherwise. General disjunctions are also known as *split disjunctions* [12].

There is an evident trade-off between the two approaches. General disjunctions can lead to a smaller tree size. On the other hand, branching on variables produces LP subproblems that are easier to reoptimize because bounds on the variables do not increase the size of the basis. Branching on a general disjunction adds one row to the formulation of the children subproblems. When this is repeated at every node, the number of constraints can grow notably leading to an increased solution time of each subproblem. In our experiments, we observe that the decreased tree size usually more than offsets this increase.

One difficulty with the application of the idea for branching on general disjunctions is the infinite number of general disjunctions that are violated by a given basic solution. Optimizing over this set would give the best results but is impractical. A natural objective would be to maximize the improvement in the lower bound as a result of branching (we assume here that MILP is a minimization problem). But there is no known way to measure this value before solving the children nodes and, hence, no way to formulate this problem. The intimate relation between split disjunctions and intersection cuts, introduced by Balas [6], provides a proxy for the change in the lower bound. The *depth* of an intersection cut, or *distance cut off*, is a reasonable measure of the cut quality. One may use the depth of the cut as a heuristic measure of the quality of the corresponding disjunction. But even maximizing the depth over the set of all intersection cuts is a difficult MILP problem.

In this paper, we consider a specific class of general disjunctions—the ones defining mixed integer Gomory cuts derived from the tableau [16]. The advantages of this class are that it is finite and fast to generate. Furthermore, disjunctions corresponding to Gomory cuts can be viewed as strengthened simple disjunctions [6]. The algorithm we propose performs a heuristic pre-selection of the most promising disjunctions based on the distance cut off by the corresponding cut, followed by an exact evaluation of the quality of the pre-selected disjunctions. This idea can be applied to other classes of intersection cuts as well, e.g. lift-and-project[7], reduce-and-split [5], and mixed integer rounding cuts [24]. Our approach is explained in detail in Section 2.2 and a short introduction to intersection cuts is included in Section 2.1. A review of related earlier work is present in Section 1.

In Section 3, we describe the experiments we conducted and their results. These experiments measure the gap closed after branching for a fixed number of levels by

branch-and-bound, and show that general disjunctions perform better than simple ones. An interesting observation is that pruning of a child by infeasibility, which is a desirable effect, happens more often when branching on general disjunctions than when branching on simple disjunctions. In a final experiment, we study the performance of our algorithm in a cut-and-branch framework.

Branching on general disjunctions can reduce the amount of enumeration compared to branching on single variables, since the latter strategy is a special case of the former. The difficulty is to find the right disjunctions to branch on. In this paper, we show that such disjunctions can be generated and applied without much computational overhead.

1 Literature Review

The idea of branching on general disjunctions is not novel. One approach proposed in the literature is to find "thin" directions in the polyhedron of feasible solutions, transform the space so that these directions correspond to unit vectors, and solve the problem in the new space by regular branch-and-bound, branching on the new variables. Transformed back to the original space, this corresponds to branching on general disjunctions. For detailed descriptions, refer to the algorithms for solving integer programming problems in fixed dimensions by Lenstra [18], Grötschel, Lovász, and Schrijver [17], and Lovász and Scarf [22]. Finding thin directions is done by lattice basis reduction based on the work of Lenstra, Lenstra, and Lovász [20]. This approach proved very efficient for some instances where branch-and-bound fails due to huge enumeration trees. Aardal et al. [1] applied a related algorithm, developed by Aardal, Hurkens, and Lenstra [2], to market split instances of the type proposed by Cornuéjols and Dawande [13]. They managed to solve instances much larger than those that could be solved by regular branch-and-bound. For a recent paper in this direction, see Mehrotra and Li [25].

Other examples of branching on general disjunctions are SOS branching and local branching. Given the presence of a Special Ordered Set (SOS) [9] constraint in the formulation (also called Generalized upper bound), branching can be done by replacing the original SOS constraint by a new SOS constraint, different in both children. This results in a significant reduction of the number of nodes that need to be enumerated. In their paper on local branching [15], Fischetti and Lodi propose a way to direct the search in the branch-and-bound algorithm. They branch on a special type of constraint that defines a neighborhood of the incumbent solution.

The methods cited above find a set of promising branching disjunctions before the start of branching or apply very specific types of general disjunctions. Our approach is to select general disjunctions at every node of the search tree based on a heuristic measure of their quality. Similar ideas have not been studied extensively in the literature. To our knowledge, there is one related study. Owen and Mehrotra [27] propose branching on general disjunctions generated by a neighborhood search heuristic. The neighborhood contains all disjunctions with coefficients in $\{-1,0,1\}$ on the integer variables with fractional values at the current node. The quality of the disjunctions is

evaluated by solving the children nodes in the spirit of strong branching (see Achterberg, Koch and Martin [3] for a discussion of strong branching).

Owen and Mehrotra tested their approach on 12 instances from MIPLIB 3.0 [10] and report a significant decrease in the total number of nodes in a majority of them compared to strong branching as implemented in CPLEX. The proposed procedure is not computationally efficient because of the large number of subproblems solved before each branching. Nevertheless, it emphasizes the important observation that branching on general disjunctions can decrease the size of the branching tree significantly.

The main differences between our approach and that of Owen and Mehrotra are:

- we consider a different class of general disjunctions, the ones defining mixed integer Gomory cuts, while Owen and Mehrotra propose $\{-1,0,1\}$-disjunctions.
- instead of an extensive heuristic search, we apply a two-phase disjunction selection procedure based on the depth of the corresponding intersection cuts in the first phase and on strong branching in the second. As a result,
- we propose an algorithm that runs in a reasonable amount of time and competes with branching on single variables in terms of tree size.

2 Branching on General Disjunctions

2.1 Theoretical Foundations of Intersection Cuts

We first present a summary of the theoretical foundations of intersection cuts. For a detailed discussion, refer to Balas [6] and Andersen, Cornuéjols, and Li [5].

Consider the Mixed Integer Linear Program:

$$\text{(MILP)} \qquad \min\{c^T x : Ax = b, x \geq 0_n, x_j \text{ integer for } j \in N_I\}, \tag{1}$$

where $c, x, 0_n \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$, and $N_I \subseteq N := \{1,2,\ldots,n\}$. Without loss of generality, assume A is of full row rank. The Linear Programming relaxation, denoted by (LP), is obtained from (MILP) by dropping the integrality constraint on x_j for $j \in N_I$. Let P_I and P denote the sets of feasible solutions to (MILP) and (LP), respectively. A *basis* for (LP) is an m-subset B of N such that the column submatrix of A induced by B is an invertible submatrix of A. Let $J := N \setminus B$ denote the index set of non-basic variables. A further relaxation of the set P with respect to a basis B is obtained by removing the non-negativity constraints on the basic variables. We denote it by $P(B)$:

$$P(B) := \{x \in \mathbb{R}^n : Ax = b \text{ and } x_j \geq 0 \text{ for } j \in J\}. \tag{2}$$

This set is a translate of a polyhedral cone: $P(B) = C + \bar{x}$, where $C = \{x \in \mathbb{R}^n : Ax = 0 \text{ and } x_j \geq 0 \text{ for } j \in J\}$ and $\bar{x}$ solves $\{x \in \mathbb{R}^n : Ax = b \text{ and } x_j = 0 \text{ for } j \in J\}$, i.e. $\bar{x}$ is the *basic solution* corresponding to the basis B. Typically B will be the optimal basis of an LP relaxation of MILP. The cone C can be expressed also in terms of its extreme rays, r^j for $j \in J$: $P(B) = Cone(\{r^j\}_{j\in J}) + \bar{x}$, where $Cone(\{r^j\})$ denotes the polyhedral cone generated by vectors $\{r^j\}$. The extreme rays of $P(B)$ can be found from the simplex tableau corresponding to the basis B.

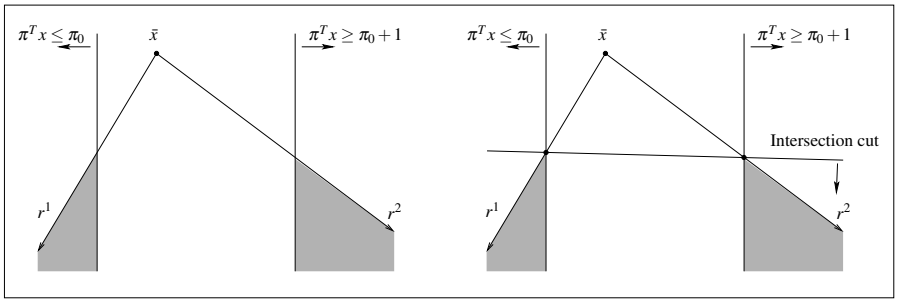

Fig. 1 Deriving the intersection cut

Define a *split disjunction* $D(\pi,\pi_0)$ to be a disjunction of the form $\pi^T x \le \pi_0 \vee \pi^T x \ge \pi_0 + 1$, where $(\pi,\pi_0) \in \mathbb{Z}^{n+1}$ and $\pi_j = 0$ for $i \notin N_I$. Clearly, any feasible solution to (MILP) has to satisfy every split disjunction. Any violated split disjunction can be used to define a cutting plane that cuts off points of P violating the disjunction. The generation of this *intersection cut*, as defined by Balas [6], is exemplified in Figure 1 and explained below.

Given a split disjunction $D(\pi,\pi_0)$, let $F_{D(\pi,\pi_0)} := \{x \in \mathbb{R}^n : \pi^T x \le \pi_0 \vee \pi^T x \ge \pi_0 + 1\}$ denote the set of points that satisfy the disjunction. Since $P_I \subseteq P(B) \cap F_{D(\pi,\pi_0)}$, a valid cut for $P(B) \cap F_{D(\pi,\pi_0)}$ is valid for P_I. In particular, the intersection cut is a half-space bounded by the hyperplane passing through the intersection points of $D(\pi,\pi_0)$ with the extreme rays of $P(B)$.

In order to find the intersection points, for all $j \in J$ we compute the scalars:

$$\alpha_j(\pi,\pi_0) := \begin{cases} -\frac{\varepsilon(\pi,\pi_0)}{\pi^T r^j} & \text{if } \pi^T r^j < 0, \\ \frac{1-\varepsilon(\pi,\pi_0)}{\pi^T r^j} & \text{if } \pi^T r^j > 0, \\ +\infty & \text{otherwise,} \end{cases} \tag{3}$$

where $\varepsilon(\pi,\pi_0) := \pi^T \bar{x} - \pi_0$ is the amount by which $\bar{x}$ violates the first term of the disjunction $D(\pi,\pi_0)$. The number $\alpha_j(\pi,\pi_0)$ for $j \in J$ is the smallest number α such that $\bar{x} + \alpha r^j$ satisfies the disjunction. In other words, $\bar{x} + \alpha_j(\pi,\pi_0) r^j$ lies on one of the disjunctive hyperplanes $\pi^T x = \pi_0$ and $\pi^T x = \pi_0 + 1$.

Now, the intersection cut associated with B and $D(\pi,\pi_0)$ is given by:

$$\sum_{j \in J} \frac{x_j}{\alpha_j(\pi,\pi_0)} \ge 1. \tag{4}$$

The Euclidean distance between $\bar{x}$ and this hyperplane is:

$$d(B,\pi,\pi_0) := \sqrt{\frac{1}{\sum_{j \in J} \frac{1}{(\alpha_j(\pi,\pi_0))^2}}} \tag{5}$$

This quantity, called *distance cut off* or *depth*, was used as a measure of cut quality by Balas, Ceria, and Cornuéjols [8].

An important result of Balas [6] is that Gomory cuts derived from the simplex tableau associated with B can be viewed as intersection cuts. Let $\bar{a}_{ij}$ be the entry of the simplex tableau in row i and column j. The mixed integer Gomory cut derived from the row in which x_i is basic can be obtained as an intersection cut from the disjunction $D(\hat{\pi}^i, \hat{\pi}^i_0)$:

$$\begin{aligned} \hat{\pi}^i_j &:= \begin{cases} \lfloor \bar{a}_{ij} \rfloor & \text{if } j \in N_I \cap J \text{ and } \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor \leq \bar{x}_i - \lfloor \bar{x}_i \rfloor, \\ \lceil \bar{a}_{ij} \rceil & \text{if } j \in N_I \cap J \text{ and } \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor > \bar{x}_i - \lfloor \bar{x}_i \rfloor, \\ 1 & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases} \\ \hat{\pi}^i_0 &= \lfloor (\hat{\pi}^i)^T \bar{x} \rfloor \end{aligned} \tag{6}$$

This disjunction can also be obtained by strengthening the simple disjunction $D(\pi^i = e_i,\ \pi^i_0 = \lfloor \bar{x}_i \rfloor)$ on the non-basic integer variables where the affected coefficients are modified so that the distance cut off is maximized.

2.2 Our Idea

This work is inspired by the relation between branching disjunctions and intersection cuts at the optimal basic solution of the current LP relaxation. A violated split disjunction can be used for generating an intersection cut but it can be used for branching as well. A good intersection cut cuts deeply into the polyhedron of feasible solutions of the LP relaxation and improves the lower, Linear Programming bound. Our suggestion is that a split disjunction defining a deep cut is good for branching too. The LP lower bound is often an important determinant of the amount of enumeration needed to complete the solution. (Because of this, improving the lower bound is the aim of common rules for selecting branching variables implemented in current MILP solvers.) The improvement in the lower bound caused by branching on a split disjunction is no less than the improvement by the corresponding intersection cut. We show this below.

A routine for branching on general disjunctions requires a procedure for selecting the disjunction to branch on, which, in turn, requires a criterion for comparing the quality of disjunctions, i.e. a criterion for comparing some measure of improvement in the lower bound.

Measure of Quality of a Disjunction

A common rule for choosing a branching variable (simple disjunction) is to maximize some function of the two improvements in objective value at the children nodes that would result from branching on this variable. Specifically, let $\bar{x}$ be the optimum solution at the current node and let $\bar{x}_1$ and $\bar{x}_2$ be the optimal solutions for the first and second child, respectively. Let $z(\bar{x}) = c^T\bar{x}$, $z(\bar{x}_1) = c^T\bar{x}_1$, and $z(\bar{x}_2) = c^T\bar{x}_2$ be the corresponding objective values. Let $\Delta_1 = z(\bar{x}_1) - z(\bar{x})$ and $\Delta_2 = z(\bar{x}_2) - z(\bar{x})$ be the improvements in objective value when branching on these two children. Typical functions used for variable selection are $\min(\Delta_1, \Delta_2)$, $\frac{1}{2}[\Delta_1 + \Delta_2]$ or $\Delta_1\Delta_2$. The last one is currently implemented in state-of-the-art MILP solvers, i.e. a variable that

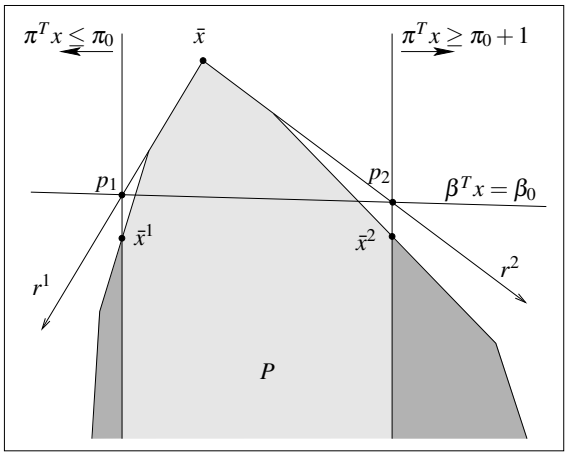

Fig. 2 The LP bound obtained by branching is different from the one obtained by cutting.

maximizes $\Delta_1\Delta_2$ is chosen for branching. Experiments with other options have been reported in the literature. For example, Linderoth and Savelsbergh [21] show good results with $\frac{2}{3}\min(\Delta_1,\Delta_2)+\frac{1}{3}\max(\Delta_1,\Delta_2)$, while Achterberg, Koch, and Martin [3] propose $\frac{5}{6}\min(\Delta_1,\Delta_2)+\frac{1}{6}\max(\Delta_1,\Delta_2)$. This is the function we will use in our experiments in this paper. The only exact way to compute these functions is to solve the linear programs at the children nodes. A commonly used method, called *strong branching*, does this for the candidate branching variables before choosing the "best" one. This is computationally expensive when applied to all integer variables with fractional values at the current basis. In practice, it is common to estimate these functions. Strong branching is impossible to apply when branching on general disjunctions because of the infinite number of such disjunctions. A procedure for pre-selecting a small finite set of disjunctions is needed before strong branching can be applied. Such a procedure requires a heuristic measure of disjunction quality.

We consider the integrality gap closed, or equivalently $\min(z(\bar{x}_1),z(\bar{x}_2))-z(\bar{x})$, an "exact" measure of the quality of a disjunction. Based on the relation between an intersection cut and the underlying disjunction, we propose to use the depth (distance cut off) of the cut as a proxy to this measure, since the distance cut off is correlated to the amount of integrality gap closed by adding the cut. Next, we show that the gap closed by branching on a split disjunction is always at least as large as the gap closed by the corresponding intersection cut.

Let $P(B)$ be defined as in (2) and consider a split disjunction $D(\pi,\pi_0)$. Let $\beta^T x \le \beta_0$ be the intersection cut defined by $P(B)$ and $D(\pi,\pi_0)$. (An example is shown in Figure 2.) The feasible sets of the children are $F_1 := P\cap\{x\in\mathbb{R}^n : \pi^T x\le\pi_0\}$ and $F_2 := P\cap\{x\in\mathbb{R}^n : \pi^T x\ge\pi_0+1\}$. Let $\bar{x}_1 := \arg\min\{c^T x : x\in F_1\}$ and $\bar{x}_2 := \arg\min\{c^T x : x\in F_2\}$ be corresponding optimal basic solutions. Let $p_1 := \arg\min\{c^T x : x\in P(B)$ and $\pi^T x\le\pi_0\}$ and $p_2 := \arg\min\{c^T x : x\in P(B)$ and $\pi^T x\ge\pi_0+1\}$. Then, $z(p_i)$ is a lower bound for $z(\bar{x}_i)$, for $i=1,2$, because $P(B)\supseteq P$. Therefore, the optimal solution of $\min\{c^T x : x\in P(B)$ and $\beta^T x\le\beta_0\}$, provides a lower bound for any measure of

At each node:

1. Generate the set $\mathcal{M}$ of all MIG disjunctions.
2. Select a subset $\mathcal{S} \subseteq \mathcal{M}$ of disjunctions with largest distance cut off, such that $|\mathcal{S}| \leq k$.
3. Apply strong branching to the disjunctions in $\mathcal{S}$ maximizing gap closed and choose a disjunction $D \in \mathcal{S}$. Branch on D.

Fig. 3 Procedure for branching on MIG disjunctions

disjunction quality which is a convex combination of $z(\bar{x}_1)$ and $z(\bar{x}_2)$. Consequently, branching on a general disjunction can provide a better lower bound than the corresponding intersection cut. In this respect, we cannot substitute branching by adding the corresponding intersection cut.

In our procedure, we will use the distance cut off by the corresponding intersection cut as a heuristic measure of the quality of a disjunction. Other measures can be used as well. Future research in this direction should be fruitful.

Procedure for Selecting the Branching Disjunction

We need a procedure for selecting promising split disjunctions for branching. As we discussed in the introduction, optimizing over the set of all split disjunctions is prohibitively expensive. Some form of heuristic search is needed and, in fact, several suggestions have been tried recently [14, 23]. Here, we simply suggest to concentrate on a finite class of general disjunctions generated directly from the current optimal basis—the set of split disjunctions defining mixed integer Gomory cuts, which we call *MIG disjunctions*. The reasons for our choice are the following. First, this set is not only finite but relatively small. Its cardinality at a given node of the branch-and-bound tree equals the number of integer variables with fractional values in the current basic solution. Second, these disjunctions are fast to obtain. They can be generated from the current tableau by a closed form formula (6). Third, as we explained at the end of Section 2.1, these disjunctions can be viewed as strengthened simple disjunctions (with respect to the cut depth) which suggests that they could perform better.

The branching procedure we propose is shown in Figure 3. We consider the set $\mathcal{M}$ of all MIG disjunctions for a specific basic solution and select a subset $\mathcal{S}$ of it, containing the most promising disjunctions according to the chosen criterion for comparison. (Here, the distance cut off by the underlying intersection cut.) We limit the cardinality of $\mathcal{S}$ to k. In our tests, we use a constant k throughout the branching tree (either $k = \infty$ or $k = 10$). The parameter k can be used to manage the computational effort at different levels, e.g. a larger k can be used close to the root where branching decisions are more important and a smaller k in the deep levels. Finally, we apply strong branching to the disjunctions in $\mathcal{S}$.

The computational complexity of this procedure at each node is dominated by Step 3 and it is comparable to applying strong branching to the k most fractional variables in branching on simple disjunctions. Note, however, that faster strategies exist for branching on simple disjunctions as investigated in Achterberg, Koch, and Martin [3].

3 Experimental Results

The test set for our experiments is the union of the Mixed Integer Linear Programming libraries MIPLIB 2.0 [28], MIPLIB 3.0 [10], and MIPLIB 2003 [4]. We exclude very easy instances that can be solved in less than 50 nodes by the algorithms we test. We also exclude some very difficult instances—those for which only less than 50 nodes can be processed in one hour. These two groups of instances are considered in the first experiment, where we compare the gap closed at the root, but they are excluded from the subsequent experiments. We also exclude instances with zero integrality gap. The final number of instances in the test set is 84. This is a heterogeneous set of benchmark instances of different sizes and with different origins and applications. It serves as a good testbed of our ideas.

All experiments are conducted on an IBM IntellistationZ Pro computer with an Intel Xeon 3.2GHz CPU (32-bit) and 2GB RAM. The MILP solver used is COIN-OR BCP, where some methods are modified for the purpose of our experiment. The LP solver is ILOG CPLEX 9.0. Mixed integer Gomory cuts, mixed integer rounding cuts, and knapsack cover cuts are generated using the cut generators in the library COIN-OR CGL.

In our experiments, we compare branching on single variables to branching on MIG disjunctions. When an instance is solved to optimality, we compare the solution time and the size of the branch-and-bound trees. When the solution of an instance is interrupted (due to time limit or bound on the depth of exploration), we compare the amount of integrality gap closed. We consider the *absolute gap closed*: the difference between the best lower bound at interruption and the lower bound at the root node, and the *relative* (*percentage*) *gap closed*: the absolute gap closed as a fraction of the integrality gap at the root.

In the first experiment, we study the gap closed after branching at the root node (Section 3.1). In the second experiment, we study the gap closed and the number of active nodes left after branching for eight levels (Section 3.2). In the third experiment, we impose a limit of 2000 processed nodes and study the amount of gap closed and the solution time per node of both algorithms (Section 3.3). In the fourth experiment, we let the cut-and-branch algorithm run till completion, or until a two-hour time limit is reached. We compare the running time and the tree size, or the amount of gap closed in case of interruption (Section 3.4).

We apply pure branch-and-bound or cut-and-branch in these experiments in order to avoid the influence of adding different cutting planes in the compared algorithms. This ensures a clean comparison between the two branching procedures. Below, we describe features of the algorithms that we implemented. No preprocessing is applied.

Cutting planes

In cut-and-branch, we generate ten rounds of mixed integer Gomory cuts, mixed integer rounding cuts, and knapsack cover cuts at the root before proceeding to branching. Inactive cuts are discarded from the formulation after each round.

Branching

We apply strong branching with no limit on the number of simplex pivots. We select the branching object that maximizes a function of the LP bounds at the chil-

dren as suggested by [3]: $\frac{5}{6}\min(\Delta_1,\Delta_2)+\frac{1}{6}\max(\Delta_1,\Delta_2)$, where Δ_1 and Δ_2 are the improvements in LP value at the children nodes. If a branching object creates an infeasible child, this object is preferred to all others. If several branching objects create an infeasible child, the LP values of their feasible children are compared.

In most of the experiments, we apply strong branching on all candidates—variables or MIG disjunctions. This provides a fair comparison since the number of MIG cuts obtained from the tableau rows, hence the number of MIG disjunctions, equals the number of fractional variables at the current basic solution. In the final experiment, we select a subset of the candidates for strong branching. Again, we consider an equal number of branching objects in both cases.

In all experiments, the node selection is performed by the best-bound-first rule: the next node to explore is the one with minimum LP bound. This enables the comparison of the gap closed by both algorithms.

3.1 Gap Closed at the Root

As a first experiment, we compare the power of the two branching objects by measuring the gap closed at the root by ordinary branching on single variables and branching on MIG disjunctions. The variable to branch on is chosen by full strong branching on all fractional variables, as described above. For short, we call this setup for branching on simple disjunctions SIMDI. When branching on general disjunctions, we select the disjunction by full strong branching on all MIG disjunctions. This guarantees a fair comparison for simple and general disjunctions since the number of branching candidates in both cases is the same. We call this setup for branching on general disjunctions GENDI.

Detailed results of the experiment are provided in Table 5. The comparison of the absolute gap closed shows that GENDI performs better for 42 out of 94 instances, while SIMDI is better for 15 instances. For 92 instances the optimal objective value is known and this allows to compute the percentage gap closed. The average percentage gap closed at the root by GENDI and SIMDI over this set of instances is 11.5% and 8.6%, respectively. The average difference is 2.9% in favor of GENDI. A statistical hypothesis test shows that this difference is significantly greater than zero. We performed a one-sided paired t-test of the null hypothesis "the gap closed by GENDI is no greater than the gap closed by SIMDI." The null hypothesis was rejected at 95% confidence (p-value=0.048).

A closer look at the cases when one method substantially dominates the other shows that for 12 instances GENDI closes a positive amount of gap while SIMDI cannot close any gap. In contrast, there are only two instances for which GENDI is unsuccessful while SIMDI manages to improve the lower bound. If we consider the instances for which both methods managed to improve the lower bound, the gap closed by GENDI is an order of magnitude larger than that of SIMDI for 14 instances, while SIMDI is an order of magnitude better for only 3 instances. Some examples where GENDI is clearly more successful are: `10teams`, `air05`, `fiber`, `harp2`, `mas76`, `nsrand-ipx`, `nw04`, `pipex`, `qnet1`, `roll3000`, `sp97ar`, `swath`, `timtab1`, and `timtab2`. Similarly, SIMDI dominates in: `bell3b`, `fixnet6`, `gesa3`, and `sample2`.

Table 1 Comparison of SIMDI and GENDI after eight levels of branching. Branch-and-bound.

	SIMDI	GENDI
Percentage gap closed		
Average	32.1%	41.7%
Count better	20	48
Active nodes at level nine		
Average	114.6	66.7
Count better	16	53
Gap closed and active nodes together		
Count better	6	45

These results are a strong indication that GENDI closes more gap than SIMDI. Next, we test whether these good results at the root proliferate throughout the tree by branching for eight levels and by solving the instances until completion. The experiment with branching for eight levels helps observe another positive effect of branching on MIG disjunctions: a decrease in the number of active nodes.

3.2 Branching for Eight Levels

In the second experiment, we branch at the top eight levels of the branch-and-bound tree and compare the resulting gap closed. As before, GENDI performs better. This is mainly due to the larger gap closed by branching on general disjunctions, which we recorded at the root as well. But now we observe an interesting secondary effect: branching on MIG disjunctions tends to produce more infeasible children, which additionally decreases the amount of enumeration. We record this phenomenon by counting the number of active nodes at the ninth level.

Detailed results of the experiment are shown in Table 6. Table 1 contains a summary of the results. In Table 1, comparison criteria are shown in italics. Lines labeled "Average" contain the average value of the criterion. Lines labeled "Count better" contain the number of instances for which one method dominates the other according to the criterion.

In terms of amount of gap closed, SIMDI dominates in 20 cases, GENDI in 48 cases out of 84. The average gap closed by SIMDI and GENDI is 32.1% and 41.7%, resp. The difference in the average gap closed is 9.6%. It is statistically significantly larger than zero with 99% confidence, according to a one-sided paired t-test (p-value=0.0021). These results support our earlier observation that GENDI closes more gap.

A graphical representation of the gap closed by SIMDI and GENDI is shown in Figure 4.A. In the figure, dots correspond to test instances. The gap closed by SIMDI is shown on the abscissa while that closed by GENDI is shown on the ordinate. The diagonal line represents equality in the gap closed by both methods. We observe that most points lie in the upper-left triangle, corresponding to "GENDI outperforms SIMDI." Furthermore, most of the points that lie in the lower-right triangle are close

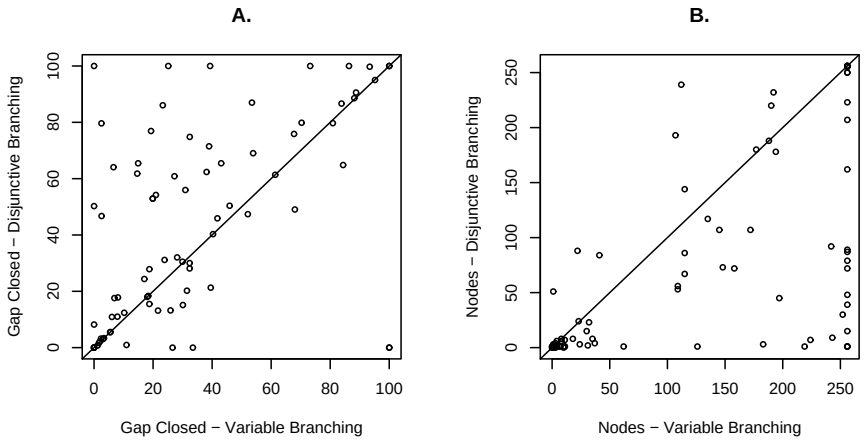

Fig. 4 A. Gap closed (in percentage) after eight levels of branching: GENDI vs. SIMDI. B. Number of active nodes after eight levels of branching: GENDI vs. SIMDI. Every data point represents a test instance.

to the diagonal line – there are few cases in which SIMDI outperforms GENDI significantly.

It is interesting to observe that GENDI typically produces a smaller number of active nodes at the ninth level. On this criterion, SIMDI performs better in 16 cases while GENDI does so in 53 cases. Out of the maximum possible 256 nodes at level nine, SIMDI generates 113 while GENDI generates 65, on average. A statistical t-test rejects the null hypothesis "GENDI produces at least as many active nodes at level nine as SIMDI" at 99.9% level of confidence (p-value=1.30e-6). This indicates that the number of active nodes created by GENDI is significantly smaller.

The difference in the performance is best seen graphically. In Figure 4.B, we plot the number of active nodes at level nine produced by GENDI vs. that produced by SIMDI. Not only do most of the points lie below the equality line but many of them reside in the bottom-right corner, corresponding to a significant difference in the number of nodes. On the other hand, out of the 16 instances for which SIMDI outperforms GENDI, only eight lie visibly far from the equality line.

The effect of a smaller number of active nodes is important not by itself but in combination with improvement in the gap. Combining both criteria, we count the cases in which an algorithm strictly dominates in one of the criteria and performs at least as well in the other criterion. SIMDI is better than GENDI in only 6 cases, while GENDI outperforms SIMDI in 45 cases out of 84.

The reason for the smaller number of active nodes is that GENDI often generates disjunctions that produce only one feasible child. For some instances, this happens at most nodes of the branching tree, resulting in only a few nodes at level nine. Although SIMDI generates many infeasible children, GENDI generates even more. Sometimes, this is combined with an impressive improvement of the gap closed over SIMDI,

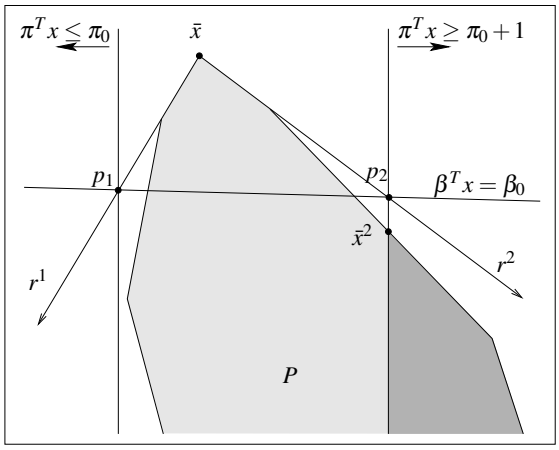

Fig. 5 Disjunction with only one feasible child

e.g. `lseu`, `nsrand-ipx`, `pipex`, `roll3000`, `swath`, and `vpm1`. See also `10teams`, `aflow30a`, `arki001`, `fiber`, and `manna81`. (Table 6.)

The combination of a larger improvement in the gap and a smaller number of active nodes is a very desirable effect and it deserves more attention. Branching on a disjunction that generates only one feasible child is equivalent to adding a single cut to the formulation. One may argue that this cut would be added by a branch-and-cut algorithm anyway. This is true in some cases but in others the disjunction inequality is stronger than the corresponding MIG cut. Figure 5 is an example. The cut generation procedure considers the polyhedral cone pointed at $\bar{x}$, relaxing some of the constraints defining P, and generates the intersection cut $\beta^T x \leq \beta_0$. But it cannot detect the fact that one of the feasible sets of the children is empty. (Here, $P \cap \{x \in \mathbb{R}^n : \pi^T x \leq \pi_0\}$.) When branching on $D(\pi, \pi_0)$, we essentially add the cut $\pi^T x \geq \pi_0 + 1$, which is stronger than $\beta^T x \leq \beta_0$.

Consequently, branching on a general disjunction that generates only one child can be viewed as strengthening the underlying intersection cut. Thus, branching on a general disjunction cannot be substituted by adding the corresponding intersection cut even when one of the disjunctive sets is empty. When both disjunctive sets are non-empty, branching on a general disjunction can still close more gap than the corresponding cut, as we showed in Section 2.2.

We do not consider branching on general disjunctions a substitute for cutting planes. Our procedure comes into play when branch-and-cut decides to start branching. It is important to note that the observed good effects of branching on general split disjunctions are not neutralized by adding cuts. We repeat the above experiment in a cut-and-branch framework where we add ten rounds of mixed integer Gomory cuts, mixed integer rounding cuts, and knapsack cover cuts. As expected, aggressive cut generation closes a significant amount of gap (63% on average), leaving less work for the branching phase. As a result, the amount of gap closed by branching on the top eight levels is smaller and the difference between the two methods is smaller. Never-

Table 2 Comparison of SIMDI and GENDI after eight levels of branching. Cut-and-branch.

	SIMDI	GENDI
Percentage gap closed		
Average	5.6%	7.4%
Count better	11	52
Active nodes at level nine		
Average	107.6	81.6
Count better	23	44
Gap closed and active nodes together		
Count better	6	39

theless, the mutual relation in performance is preserved, as seen in Table 2. Detailed results of the experiment are shown in Table 7.

3.3 Branching for Two Thousand Nodes

As a next step in comparing the strength of the two branching strategies under consideration, we let the algorithms run for 2000 nodes and observe the amount of integrality gap closed, the running time, as well as the size of the branching trees when the instances are solved. The limit of 2000 nodes allows the algorithms to explore a larger part of the branching tree while terminating in reasonable time. It also places both algorithms under equal terms in the comparisons. We apply full strong branching within branch-and-bound as before. Summary results of the experiment are shown in Table 3 and detailed results are presented in Table 8.

Considering the amount of gap closed, GENDI is better than SIMDI in 33 out of 84 instances while SIMDI dominates in 8 cases. On average GENDI closes 4.5% more gap than SIMDI and, if we consider only the 41 instances for which the performance of the two algorithms differs, the advantage in favor of GENDI increases to 9.2%. In addition, GENDI achieves this result with a smaller number of visited nodes, 15% smaller on average. The larger efficiency of branching on MIG disjunctions is evident from the ratios of gap closed per node by GENDI and gap closed per node by SIMDI for all instances. The geometric mean of these ratios is 1.58.

Not only does GENDI close more gap but it is faster. The solution time per node of GENDI is in fact 10% smaller than that of SIMDI, on average. (These results exclude the solution time of the root relaxation.) This may sound surprising. Adding branching constraints to the formulation should increase the solution time per LP subproblem. A more detailed look shows that this is indeed the case. Recall that we perform strong branching at each node in this experiment. This means that, at each node, we solve two LP subproblems for each integer infeasibility, i.e., for each integer variable x_j that has a fractional value $\bar{x}_j$ in the optimal basic solutions $\bar{x}$ of the node LP subproblem. As expected, the solution time per subproblem of GENDI is significantly larger than that of SIMDI (11.7 vs. 6.0 milliseconds on average; the difference is statistically significant with 95% confidence, p-vaule=0.024). However,

Table 3 Comparison of SIMDI and GENDI after 2000 nodes of branch and bound.

	SIMDI	GENDI
Percentage gap closed		
Average	73.1%	77.6%
Count larger	8	33
Processed nodes		
Average	107.6	81.6
Count smaller	11	31
Solution time per node [seconds]		
Average	0.94	0.84
Count smaller	46	32
Solution time per suproblem [milliseconds]		
Average	6.0	11.7
Count smaller	71	7
Number of strong-branching subproblems per node		
Average	93.7	65.6
Count smaller	22	55

the average number of subproblems per node that GENDI solves is significantly less than that of SIMDI (65.6 vs. 93.7 subproblems per node on average; the difference is statistically significant with 95% confidence, p-value=0.042).

The last observation is important in itself. Fewer strong-branching subproblems per node means that the optimal basic solutions of the node LP subproblems tend to have fewer integer infeasibilities, hence fewer MIG disjunctions are available to choose from. Table 8 shows the average number of infeasibilities per node. The number of LP subproblems solved at each node is double that number.

In order to perform a more detailed analysis of the results of this experiment, we partition the test instances into three sets: those solved by both algorithms, those which cannot be solved by either algorithm in 2000 nodes, and those solved by only one of the algorithms.

Instances Solved by Both Methods

Thirty-three instances are solved by both algorithms in less than two thousand nodes. We compare the size of the branching tree and the solution time of both algorithms. Branching on MIG disjunctions performs much better than branching on variables on both metrics. GENDI enumerates a smaller number of nodes in 21 cases while SIMDI dominates in 10 cases. We test the hypothesis that the number of nodes processed by GENDI is no less than that processed by SIMDI (paired t-test). The hypothesis is rejected with more than 90% confidence (p-value = 0.07). The ratios of processed nodes by GENDI and SIMDI vary between 0.045 and 11. Their geometric mean is 0.69.

Not only does branching on general disjunctions require less enumeration but it is significantly faster. Recall that we are not using a state-of-the-art implementation of variable branching here but instead an implementation that chooses the branching

variable based on full strong branching. In this context, faster branching essentially means that the LP subproblems have optimal basic solutions with fewer integer infeasibilities. Thirty of these instances were solved faster by GENDI and only three were solved faster by SIMDI. GENDI is 40% faster on average. The ratios of solution time per node by GENDI and SIMDI vary between 0.1 and 1.9, and their geometric mean is 0.61. A hypothesis test shows that the mean of this ratio is smaller than one, with 99.9% confidence (p-value = 1.2e-4).

Some of the instances for which branching on general disjunctions is much more efficient than branching on variables are `flugpl` and `nw04`, where GENDI needs only 5% of the nodes enumerated by SIMDI, as well as `p0040` (9%), `gen` (22%), `p0033` (23%), and `pipex` (23%). Instances for which SIMDI clearly dominates are `qnet1` and `qnet1_o`, where SIMDI enumerated only 9% and 40% of the number of nodes visited by GENDI, respectively.

Instances Solved by Only One of the Methods

One instance, `stein27`, is solved by SIMDI in 1966 nodes but cannot be solved by GENDI in 2000 nodes. GENDI closes only 80% of the integrality gap.

On the other hand, ten instances are solved by GENDI but cannot be solved by SIMDI. In many cases, the difference in performance is striking. Instance `gt2` is solved by GENDI in 47 nodes. SIMDI manages to close the integrality gap in 2000 nodes but cannot prove optimality. GENDI solves `fiber`, `vpm1` and `mod008` in 179, 273, and 525 nodes resp., while 2000 processed nodes by SIMDI close 76%, 48%, and 78% of the gap, resp. Most impressively, GENDI solves instance `manna81` in 273 nodes and 22 minutes while SIMDI closes only 4.5% of the gap in 2000 nodes and 15 hours.

These results clearly support our previous observation that branching on MIG disjunctions closes a larger amount of gap per branching.

Instances Not Solved by Any Method

Forty instances cannot be solved in two thousand nodes by either of the algorithms. Branching on MIG disjunctions performs better on these instances as well. We measure the performance of both methods by the amount of integrality gap closed after 2000 branchings. Branching on MIG disjunction closes a larger amount of gap in 25 cases while branching on variables is more efficient in 7 cases. The mean ratio of the gap closed by GENDI and SIMDI is 1.12 (geometric mean). The range of these ratios is 0.68 to 4.88. SIMDI is faster on most of these instances. The solution time per node is practically equal for both algorithms, on average. The mean ratio of the solution time per node of GENDI and SIMDI is 1.01. The extreme cases when one of the algorithms closes significantly more gap than the other are shown in Table 4.

For this subset of larger and more difficult instances, we again observe that branching on general disjunctions causes fewer integer infeasibilities in the node subproblems. For each instance, we computed the ratio of the number of strong branching subproblems solved per node by GENDI and SIMDI. The geometric mean of these

Table 4 Extreme cases of outperformance in terms of integrality gap closed.

Instance	Gap Closed by SIMDI	Gap Closed by GENDI
opt1217	5%	25%
swath	11%	37%
roll3000	46%	71%
nsrand-ipx	37%	54%
harp2	57%	42%
misc07	70%	54%
danoint	24%	17%

ratios over all instances is 0.78 (with a range of 0.26 to 1.47), showing that branching on general disjunctions is associated with about 22% decrease in the number of integer infeasibilities per node, compared with branching on variables.

The results of this experiment confirm that branching on MIG disjunctions is significantly more efficient than branching on variables in a branch-and-bound algorithm, when strong branching is performed at every node. We observe that branching on MIG disjunctions closes more gap per branching and that it is usually no slower than branching on variables.

3.4 Cut-and-Branch

Above, we applied strong branching on all candidate variables or disjunctions. This approach allowed us to make a rigorous and fair comparison of the strength of these branching objects. Clearly, exhaustive strong branching is not an efficient solution technique for general MILP problems. Below, we suggest a more practical version of the algorithm where strong branching is performed only on ten candidates: the ten most fractional variables or the ten MIG disjunctions with greatest depth. This strategy for variable branching is not state-of-the art: We use it here because it can be implemented very similarly for simple and MIG disjunctions and therefore allows us to make a direct comparison. We test the performance of this approach in a cut-and-branch framework.

We make two minor modifications to the algorithm for branching on general disjunctions in order to make it more practical. The first aims at avoiding an unnecessary increase in the size of the subproblems and the second aims at avoiding numerical problems. First, when the π vector of a disjunction is a singleton, we branch on that variable instead of adding explicit constraints of the type $x_j \leq \lfloor \bar{x}_j \rfloor$ or $x_j \geq \lfloor \bar{x}_j \rfloor + 1$. Second, we do not consider dense disjunctions for branching. We define dense disjunctions to be those whose vector $\pi \in \mathbb{Z}^n$ has support of cardinality greater than $\max(10, 0.1n)$. If all generated disjunctions at a node are dense, we branch on variables instead.

The limit on the solution time is two hours. Our goal is to compare the tree size and the running time. For the instances not solved to optimality, the amount of gap closed is compared. We also compute the increase in computing time caused

by branching on general disjunctions by computing the ratio of the average solution time per node required by the two different branching schemes. For simplicity, we continue calling both algorithms SIMDI and GENDI regardless of the modifications we introduce in this section. Detailed results from this experiment are presented in Table 9.

Out of the 84 test instances, SIMDI and GENDI solved an equal number: 63. Overall, the difference in the performance of both algorithms is smaller than that observed in the previous experiments. The reason is the addition of aggressive cut generation at the root that closes about 60% of the integrality gap, on average. This leaves less work for the branching phase, hence the smaller difference between the two tested algorithms. Nevertheless, branching on general disjunctions significantly outperforms branching on variables for a number of instances, leading to orders of magnitude improvement in the solution time and the number of processed nodes. Most importantly, GENDI is clearly more efficient on the most difficult instances that cannot be solved within the time limit of two hours.

The solution time per node is about 9% larger when branching on MIG disjunctions, computed as a geometric mean of the ratios of time per node by GENDI to time per node by SIMDI over all test instances. This contrasts our observations from the previous experiments where GENDI was usually no slower than SIMDI. The reason most likely is the increased size of the subproblems due to adding explicit branching disjunctions to the formulation in GENDI. Sometimes these rows are dense, which affects the solution time adversely. In the previous experiments, all or much of the branching occurred at the top 10-20 levels of the branching trees where the number of rows added by GENDI is relatively small. As branching proceeds to the deeper levels, the contribution of the MIG branching disjunctions to the size of the subproblems becomes more significant. This effect can be mitigated by restricting branching on MIG disjunction to the top n levels, e.g. $n = 15$, and proceeding with branching on variables at the deeper levels. This would exploit the greater power of MIG disjunctions in closing integrality gap at the most important nodes close to the root while minimizing the computational burden of adding many rows to the formulation.

The larger computational time per node is offset by the larger amount of gap closed per node by branching on MIG disjunctions: 65% larger, computed as a geometric mean of the ratios of gap closed per node. This allows GENDI to outperform SIMDI on the most difficult instances.

We proceed with detailed analysis of the results of this experiment partitioning the instances in three sets.

Instances Solved by Both Methods

Sixty-one instances are solved by both methods within the two hours limit. The average integrality gap closed by cuts for these instances is 68%. The remaining 32% are closed by branching. All these instances are solved to completion, therefore we compare the number of nodes processed by the two algorithms.

Fourteen of the instances are solved in less than ten nodes by both SIMDI and GENDI. Most of them are practically solved by cuts and the others require only a few

branchings in order to close the small remaining integrality gap. We exclude these easy instances from further analysis.

Out of the remaining 47 instances, 24 are solved in fewer nodes by SIMDI while GENDI is more efficient for 23 instances. The extreme cases are `bell4`, `bell5`, `flugpl`, and `p2756` where GENDI needs 0.23%, 0.79%, 2.3%, and 2.4%, resp., of the nodes that SIMDI requires to solve the problems. At the other extreme are instances `fixnet4` and `vpm2` which SIMDI solves in 24% and 27%, resp., of the nodes GENDI requires. A careful look at the ratios of nodes processed by GENDI and SIMDI on a particular instance shows that GENDI usually outperforms SIMDI with a much larger margin. The geometric mean of the ratios for the 45 analyzed instances is 0.71. The range of the ratio is from 0.0023 to 4.20.

These results show that on the set of relatively easier instances, branching on general disjunctions typically outperforms branching on variables about half the time but when it does, the difference is often significant, while SIMDI outperforms GENDI by a smaller margin.

Instances Solved by Only One of the Methods

Two instances are solved by GENDI but not solved by SIMDI within two hours. GENDI solves instance `p0548` in two seconds and 50 nodes while SIMDI cannot solve it in two hours and 437,000 nodes. Instance `rout` is solved in 18 minutes and 18,000 nodes by GENDI and cannot be solved in 160,000 nodes and two hours by SIMDI.

Two other instances are solved by SIMDI but not solved by GENDI within two hours. Instance `aflow30a` takes 40 minutes and 20,000 nodes with SIMDI but more than 60,000 nodes with GENDI. Instance `cap6000` requires 15 minutes and 7900 nodes with SIMDI. GENDI cannot solve it in two hours and 8200 nodes.

Instances Not Solved by Either Method

Twenty instances are not solved by either method within the time limit. We compare the gap closed by the two algorithms in the two-hour interval of time. Neither of the algorithms closes any gap by branching for five instances: `10teams`, `liu`, `markshare1`, `markshare2`, and `opt1217`. For the remaining 15 instances, comparing the gap closed in the branching phase of cut-and-branch shows that GENDI closes 63% more gap than SIMDI, on average. This number is computed as a geometric mean of the ratios of gap closed by GENDI and SIMDI.

In addition, GENDI achieves this larger improvement in gap closed by processing significantly fewer nodes than SIMDI. GENDI visits 58% of the nodes that SIMDI visits in the two-hour solution time. As we discussed, the reason is likely the larger computational time per subproblem due to the numerous added MIG disjunctions. Comparing the ratios of gap closed per node shows that a single branching on MIG disjunctions closes 2.8 times more gap than a branching on variables, on average, for these 15 instances. This observation emphasizes the large potential of branching on general disjunctions.

SIMDI was more efficient in five cases while GENDI closed more gap than SIMDI in ten cases. For example, on instance `mkc` GENDI closes 11.2 times more gap than

SIMDI while processing only 18% of the nodes SIMDI processes. On instance `arki001`, GENDI closes 8.2 times more gap in 60% of the nodes. Overall, GENDI closes at least twice as much gap as SIMDI for six instances while SIMDI is never twice as good as GENDI. The best case relative performance of SIMDI are instances `danoint` and `harp2`, for which it closes 1.85 and 1.3 times more gap than GENDI, respectively. It is important to note that in the same time it processes 3.6 and 2.6 times more nodes, resp.

The results of the last experiment show that branching on MIG disjunctions convincingly outperforms branching on variables in a cut-and-branch framework that combines aggressive cut generation and strong branching on ten branching objects. In particular, when applied to the most difficult instances in the test set, branching on general disjunctions closes significantly more gap than branching on variables for the same time interval. It also closes much more gap per processed node than branching on variables does. These conclusions confirm the large potential of branching on MIG disjunctions that we observed in the other experiments.

We designed our experiments in a way that allows a direct comparison between branching on general disjunctions and branching on variables. The algorithms we tested do not pretend to compete with the best current algorithms for solving MILP problems. Using cut-and-branch with aggressive strong branching at every node is not the most efficient algorithm for general MILP. This is evident from the comparison with other research and commercial codes shown in Table 10. The source of data for this table is a comparison of optimization software on a set of benchmark instances provided by Mittelmann [26] and published on 1/23/2006 – approximately the time when our version of BCP was downloaded and our experiments conducted. We used Cplex 9.0 while Table 10 contains Cplex 10.0 solution times. The hardware platform on which those tests were performed is comparable to ours: 3.2 GHz Intel Pentium 4 CPU, with 4GB RAM, running Linux OS. The tested solvers were run for two hours with default settings.

4 Conclusion

In this paper, we propose a procedure for branching on general disjunctions as part of a branch-and-cut algorithm for solving Mixed Integer Linear Programming problems. The procedure is independent of the instance characteristics and can be applied to any MILP.

We discuss the relation between branching disjunctions and intersection cuts and show that branching on general disjunctions can close more gap than adding the corresponding intersection cut, implying that branching cannot be substituted by cutting planes. We propose to use the distance cut off by the intersection cut as a measure of quality of the disjunction. This measure can be used for pre-selection of the most promising disjunctions before strong branching.

We test these ideas in experiments with 84 test instances from the literature, comparing branching on variables with branching on general disjunctions. We observe

that on average the effect of branching on general disjunctions is: (i) a smaller tree size for the instances solved to optimality, and (ii) a larger amount of gap closed for the other instances. The test results indicate that the proposed procedure outperforms regular branching on variables for most instances.

We also observe that, in addition to the larger amount of gap closed, branching on general disjunctions results in only one feasible child more often than branching on variables does (with the same branching rules applied). This is an interesting side effect that decreases the tree size further. In our implementation, we select ten disjunctions for strong branching and sometimes more than one produces an infeasible child. One could add all these disjunctive inequalities, which are valid for P_I, as cuts instead of adding just one through branching. This will decrease the search space at the child node. We have not implemented this idea.

Another interesting observation about our implementation of branching on general disjunctions is that it tends to produce LP subproblems with fewer integer infeasibilities than those produced by branching on variables.

We obtain an efficient algorithm by considering only a specific class of disjunctions—those defining mixed integer Gomory cuts—instead of searching the whole set of split disjunctions. This approach can be extended to disjunctions defining other classes of split cuts, such as lift-and-project, reduce-and-split, and mixed integer rounding cuts. The advantages of branching on variables and on general disjunctions can be combined in one algorithm. One idea is to branch on general disjunctions close to the root and branch on variables at the deeper levels. Another approach would be to generate disjunctions and identify fractional variables simultaneously, and let strong branching choose the branching object among them. Some research in this direction has been initiated [14]. More studies would be fruitful on criteria for evaluating disjunctions.

Acknowledgments

We are indebted to François Margot for his help with the software packages in COIN-OR.

Table 5: Comparison of the gap closed at the root node by branching on a single variable and on a MIG disjunction.

Instance	Absolute gap closed		Relative gap closed [%]	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.
10teams	0	4	0	57.14
a1c1s1	121.2	121.2	1.15	1.15
aflow30a	11.3	11.2	6.49	6.42
aflow40b	5.7	4.5	3.52	2.77
air04	14.9	20.6	2.48	3.43
air05	0.39	12.2	0.08	2.45
arki001	42.2	42.2	3.48	3.48
bell3a	3174	1857	20.03	11.72
bell3b	316638	13855	82.89	3.63
bell4	360675	122948	64.79	22.08
bell5	298008	298008	83.25	83.25

continued on the next page

Table 5: *continued*

Instance	Absolute gap closed		Relative gap closed [%]	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.
blend2	0.01	0.13	1.47	19.15
bm23	0.91	0.29	6.78	2.16
cap6000	69.8	85.6	43.55	53.38
dano3mip	0	0.01	—	—
danoint	0.05	0.05	1.65	1.65
dcmulti	648	648	15.4	15.4
egout	29	29	6.92	6.92
fast0507	0	0.02	0	1.08
fiber	1293	43547	0.52	17.43
fixnet3	60.7	60.7	0.54	0.54
fixnet4	66.5	108	1.42	2.31
fixnet6	124	2.02	4.48	0.07
flugpl	689	1874	2.01	5.46
gen	65.2	65.2	35.64	35.64
gesa2	14015	14015	4.62	4.62
gesa2_o	14015	14015	4.62	4.62
gesa3	4819	21.5	3.06	0.01
gesa3_o	4819	11452	3.06	7.28
gt2	4750	4816	61.65	62.51
harp2	2.03	11627	0	2.56
khb05250	1670000	1670000	15.15	15.15
l152lav	0.82	9.6	1.25	14.63
liu	214	214	—	—
lp4l	16.57	10.66	67.63	43.51
lseu	12	98.06	4.21	34.37
manna81	0.5	0.5	0.38	0.38
markshare1	0	0	0	0
markshare2	0	0	0	0
mas74	40.8	75.5	3.10	5.73
mas76	1.76	62.3	0.16	5.60
misc01	0	0	0	0
misc02	0	25	0	3.68
misc03	0	0	0	0
misc04	5.64	5.64	54.87	54.87
misc05	1.2	0	2.24	0
misc06	2.09	2.09	22.79	22.79
misc07	0	10	0	0.72
mkc	0	0	0	0
mod008	0.17	0.02	1.06	0.12
mod010	3.58	0.92	22.49	5.78
mod011	415328	896043	5.49	11.85
mod013	1.36	0.63	5.46	2.53
modglob	3984	4225	1.29	1.36
momentum1	3200	4263	8.8	11.73
net12	10.4	10.4	5.29	5.29
nsrand-ipx	0	36.7	0	1.58
nw04	0.67	343	0.12	62.27
opt1217	0	0.27	0	6.72
p0033	2.47	205	0.43	36.2
p0040	4.84	161	2.1	69.83
p0201	0	191	0	25.78
p0282	32844	32844	40.28	40.28
p0291	101	1031	2.86	29.31
p0548	12.7	12.7	0.15	0.15
p2756	9.8	1.42	2.25	0.33
pipex	0.05	1.62	0.34	11.16
pk1	0	0	0	0
pp08a	233	233	5.05	5.05
pp08aCUTS	61.3	81.4	3.28	4.36
qiu	0	0	0	0
qnet1	3.98	226	0.23	12.85
qnet1_o	245	351	6.22	8.91

continued on the next page

Table 5: *continued*

Instance	Absolute gap closed		Relative gap closed [%]	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.
rd-rplusc-21	0	0	0	0
rgn	0	2.2	0	6.59
roll3000	0	2.85	0	0.16
rout	2.34	2.34	2.45	2.45
sample2	15	0	11.72	0
sentoy	6.5	10.8	9.66	16.04
set1al	37.9	37.9	0.80	0.80
set1ch	757	757	3.36	3.36
set1cl	37.9	37.9	0.79	0.79
seymour	0.33	0.5	1.72	2.61
sp97ar	2537	234316	0.03	2.88
stein15	0	0	0	0
stein27	0	0	0	0
stein45	0	0	0	0
stein9	0	0	0	0
swath	0.02	6.45	0.02	4.85
timtab1	0	23872	0	3.24
timtab2	0	24000	0	2.37
tr12-30	929	929	0.80	0.80
vpm1	0.25	0.67	5.46	14.63
vpm2	0.11	0.11	2.85	2.85

Table 6: Comparison of the gap closed and number of active nodes after eight levels of branching on single variables and on MIG disjunctions. (Branch-and-bound)

Instance	Absolute gap closed		Relative gap closed [%]		Nodes at level 9	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
10teams	0	7	0	100	256	79
a1c1s1	1051	1051	10	10	145	107
aflow30a	32.8	48.7	18.74	27.86	252	30
aflow40b	27.6	39.6	17.03	24.37	242	92
arki001	83.6	209	6.89	17.23	172	107
bell3a	10785	7775	68.04	49.05	37	4
bell3b	336827	338368	88.17	88.57	2	3
bell4	494120	504243	88.76	90.58	32	23
bell5	300135	310221	83.84	86.66	2	1
blend2	0.21	0.38	30.93	55.97	107	193
bm23	5.31	2.86	39.54	21.3	18	8
cap6000	109	122	67.75	75.88	256	15
danoint	0.05	0.05	1.65	1.65	194	178
dcmulti	2250	3659	53.49	86.99	22	88
egout	118	134	28.15	32.03	1	1
fiber	6276	198990	2.51	79.64	256	39
fixnet3	2066	2066	18.36	18.36	188	188
fixnet4	877	725	18.74	15.5	190	220
fixnet6	306	26.5	11	0.95	192	232
flugpl	6624	26393	19.3	76.92	35	8
gen	98.6	126	53.89	69.04	2	2
gesa2	60346	160548	19.89	52.92	109	56
gesa2_o	60346	160548	19.89	52.92	109	53
gesa3	50987	44266	32.39	28.12	115	86
gesa3_o	50987	47248	32.39	30.02	115	67
gt2	6234	6142	80.9	79.7	256	48
harp2	11627	211932	2.56	46.73	1	51
khb05250	6764925	6764925	61.38	61.38	256	256
l152lav	17.9	40.0	27.24	60.86	11	7
lp4l	17.9	24.5	73.22	100	10	0

continued on the next page

Table 6: *continued*

Instance	Absolute gap closed		Relative gap closed [%]		Nodes at level 9	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
lseu	123	187	43.07	65.47	224	7
manna81	4	4	3.01	3.01	256	1
markshare1	0	0	0	0	256	89
markshare2	0	0	0	0	256	162
mas74	135	163	10.22	12.38	256	223
mas76	89.9	198	8.09	17.82	256	255
misc01	30.7	55.2	6.06	10.9	11	1
misc02	214	138	31.47	20.22	1	3
misc03	100	255	6.9	17.59	7	1
misc04	10.3	10.3	100	100	0	0
misc05	21.6	21.6	40.3	40.3	8	6
misc06	9.17	9.17	99.99	99.99	0	0
misc07	16.3	10	1.16	0.72	62	1
mkc	16.1	0	33.48	0	11	1
mod008	2.4	10.5	14.93	65.46	197	45
mod010	13.8	15.9	86.37	100	1	0
mod011	1584442	4100446	20.95	54.21	256	256
mod013	11.5	12.6	45.93	50.38	30	15
modglob	93178	46851	30.1	15.13	256	250
nsrand-ipx	0	190	0	8.19	256	1
nw04	139	551	25.15	100	1	0
opt1217	0	2.02	0	50.25	256	87
p0033	400	454	70.32	79.86	31	2
p0040	90.6	230	39.31	100	3	0
p0201	240	554	32.43	74.83	23	24
p0282	77619	77490	95.19	95.03	148	73
p0291	3286	3510	93.4	99.74	4	6
p0548	167	182	2	2.18	1	1
p2756	10.2	13.9	2.34	3.2	1	1
pipex	3.38	12.5	23.29	86.06	183	3
pk1	0	0	0	0	256	256
pp08a	1378	1404	29.96	30.51	256	250
pp08aCUTS	447	582	23.89	31.14	256	256
qiu	416	379	52.1	47.39	135	117
qnet1	734	806	41.79	45.91	112	239
qnet1_o	1500	2454	38.13	62.38	177	180
rgn	2.2	21.4	6.59	64.07	41	84
roll3000	0.8	13.6	0.04	0.76	6	1
rout	20.7	12.6	21.66	13.15	158	72
sample2	108	83	84.38	64.84	115	144
sentoy	26.2	48.1	38.97	71.48	256	72
set1al	262	262	5.55	5.55	1	1
set1ch	4065	4065	18.04	18.04	8	8
set1cl	262	262	5.44	5.44	1	1
stein15	2	0	100	0	0	1
stein27	1.33	0	26.6	0	219	1
stein45	0	0	0	0	256	1
stein9	1	0	100	0	0	1
swath	0.29	42.2	0.22	31.75	126	1
timtab1	58221	80887	7.91	10.99	3	4
timtab2	63992	49536	6.32	4.89	4	1
tr12-30	3914	3914	3.36	3.36	1	1
vpm1	0.67	2.83	14.63	61.79	243	9
vpm2	1	0.51	25.91	13.21	24	3

Table 7: Comparison of the gap closed and number of active nodes after eight levels of branching on single variables and on MIG disjunctions. (Cut-and-branch)

Instance	Relative gap closed by cuts [%]	Relative gap closed by branching [%]		Nodes at level 9	
		Simple disj.	MIG disj.	Simple disj.	MIG disj.
10teams	100	0	0	0	0
a1c1s1	54.40	6.64	6.64	66	88
aflow30a	50.50	0.18	5.15	87	256
aflow40b	42.97	0.10	3.24	200	256
arki001	52.59	0	3.70	203	195
bell3a	70.31	0.43	12.07	114	32
bell3b	77.67	14.32	12.42	26	1
bell4	92.65	2.89	2.98	78	15
bell5	90.79	2.96	4.21	52	38
blend2	29.46	7.36	23.56	125	190
bm23	33.36	3.20	27.18	195	13
cap6000	62.98	4.77	3.56	256	18
danoint	0.99	0.66	0.99	201	178
dcmulti	69.49	2.19	27.96	15	97
egout	99.99	0.01	0.01	0	0
fiber	94.65	0.11	1.64	2	48
fixnet3	99.87	0.13	0.13	1	20
fixnet4	89.65	4.84	1.84	192	197
fixnet6	84.57	1.99	3.21	192	202
flugpl	14.69	12.09	52.96	21	6
gen	100	0	0	0	0
gesa2	98.16	1.24	1.19	232	225
gesa2_o	93.14	0.29	2.13	172	169
gesa3	77.5	13.31	10.93	150	180
gesa3_o	74.77	8.69	15.14	109	176
gt2	99.87	0.13	0.13	256	4
harp2	63.54	0.59	3.57	39	227
khb05250	99.11	0.82	0.89	1	0
l152lav	34.19	9.72	17.87	6	3
lp4l	100	0	0	0	0
lseu	78.43	0.46	7.28	256	13
manna81	100	0	0	0	0
markshare1	0	0	0	256	38
markshare2	0	0	0	256	207
mas74	8.89	0.14	2.61	256	255
mas76	13.22	1.51	8.13	256	238
misc01	4.18	0.12	5.39	13	2
misc02	16.56	3.29	13.25	1	1
misc03	17.62	0.04	0.90	14	2
misc04	83.67	16.34	2.24	0	1
misc05	53.06	32.59	41.04	3	4
misc06	74.59	2.84	19.08	256	256
misc07	0.72	0.90	1.79	56	2
mkc	56.56	29.81	1.19	54	2
mod008	86.68	2.49	13.32	104	0
mod010	100	0	0	0	0
mod011	46.93	4	7.44	256	256
mod013	74.25	10.07	21.02	3	6
modglob	70.11	10.61	8.54	256	254
nsrand-ipx	66.20	0	5.22	256	57
nw04	97.81	2.19	2.19	0	0
opt1217	50.50	0	2.24	256	12
p0033	99.54	0.46	0.46	0	0
p0040	100	0	0	0	0
p0201	61.32	0.44	16.59	17	25
p0282	97.53	0.99	1.18	173	60
p0291	99.82	0.16	0.18	13	0
p0548	94.37	2.60	2.89	2	5
p2756	97.83	0.11	0.04	8	2
pipex	58.15	2.20	17.23	12	2

continued on the next page

Table 7: *continued*

Instance	Relative gap closed by cuts [%]	Relative gap closed by branching [%]		Nodes at level 9	
		Simple disj.	MIG disj.	Simple disj.	MIG disj.
pk1	0	0	0	256	255
pp08a	90.50	1.29	1.47	256	252
pp08aCUTS	80.21	2.77	3.05	256	256
qiu	7.80	42.98	42.98	185	185
qnet1	72.43	2.08	21.83	238	248
qnet1_o	83.33	8.44	9.57	31	221
rgn	75.15	0.21	3.29	30	34
roll3000	75.95	0.06	1.09	9	6
rout	4.11	6.90	15.31	70	21
sample2	41.71	8.16	11.41	14	18
sentoy	27.17	13.27	28.72	256	113
set1al	99.98	0.02	0.02	1	0
set1ch	91.05	0.12	0.19	256	238
set1cl	100	0	0	0	0
stein15	0	100	0	0	3
stein27	0	37.40	0	219	1
stein45	0	0	0	255	1
stein9	100	0	0	0	0
swath	33.68	0.24	1.22	105	10
timtab1	41.76	0.01	0.75	5	3
timtab2	34.11	0.06	2.29	12	3
tr12-30	93.31	1.07	1.07	256	256
vpm1	90.61	1.31	9.39	255	62
vpm2	77.72	0.26	2.07	113	209

Table 8: Comparison of the gap closed, the number of processed nodes, the solution time and the average integer infeasibilities per node when branching on single variables and on MIG disjunctions. (Branch-and-bound. Node limit of 2000 processed nodes.)

Instance	Relative gap closed by branching [%]		Nodes processed		Solution time		Average Integer Infeasibilities per Node	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
10teams	100	100	119	65	2485.7	1346.4	143	71.5
a1c1s1	18.31	19.23	2000	2000	6358.8	6642.7	162.1	81.1
aflow30a	68.4	68.7	2000	2000	472.9	478.2	25.2	12.6
aflow40b	54.5	49	2000	2000	3357.5	3128.2	37.1	18.6
arki001	69.03	95.58	2000	2000	1843.1	1451.3	42	21
bell3a	87.1	87.1	2000	2000	79.3	75	1.1	0.6
bell3b	99.7	99.9	2000	2000	84.4	73	5.8	2.9
bell4	99.9	100	2000	939	89.7	23.3	12.4	6.2
bell5	98	100	2000	613	77.8	21.6	3.1	1.6
blend2	100	100	1171	591	66	30	4.5	2.3
bm23	100	100	137	113	5.2	2.2	3.9	2
cap6000	78.1	78.1	2000	2000	681.9	724.9	3	1.5
danoint	24.4	16.5	2000	2000	3964.9	5640.5	30.5	15.3
dcmulti	100	100	626	627	41.8	21.5	6.3	3.2
egout	99.3	100	2000	202	79.5	1.5	5.2	2.6
fiber	76	100	2000	179	341.1	28.8	35.7	17.9
fixnet3	42.9	42.9	2000	2000	357.6	318.9	58.7	29.4
fixnet4	39.6	38.3	2000	2000	383	366.2	61.9	31
fixnet6	42.3	43.8	2000	2000	353.9	360.1	51.4	25.7
flugpl	100	100	1152	55	43.5	0.2	3.6	1.8
gen	100	100	169	37	14.6	4	9.9	5
gesa2	87.3	99.7	2000	2000	790.9	522.2	49.8	24.9
gesa2_o	87.3	99	2000	2000	818.4	566.5	61.8	30.9
gesa3	100	100	1085	858	310.2	280.2	31.6	15.8
gesa3_o	100	100	1122	1124	396.4	288.7	46.8	23.4
gt2	100	100	2000	47	84.9	0.6	8.7	4.4
harp2	56.9	41.7	2000	2000	466.3	449.2	25.6	12.8
khb05250	100	100	829	823	57.5	26.3	9.1	4.6
l152lav	100	100	123	76	54.8	33.4	40.2	20.1
lp41	100	100	46	46	8.1	8.7	23.3	11.7
lseu	85.3	100	2000	1961	77.2	68.4	6	3
manna81	4.5	100	2000	273	53475	1315.6	804.8	402.4

continued on the next page

Table 8: *continued*

Instance	Relative gap closed by branching [%]		Nodes processed		Solution time		Average Integer Infeasibilities per Node	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
markshare1	0	0	2000	2000	75.4	69.9	6	3
markshare2	0	0	2000	2000	77.2	72.1	7	3.5
mas74	35.7	36	2000	2000	95.3	92	11.9	6
mas76	38.1	39.6	2000	2000	91.1	78.5	10.6	5.3
misc01	100	100	153	169	8.2	5.6	12.4	6.2
misc02	100	100	17	15	0.8	0.3	8.5	4.3
misc03	100	100	185	313	14	13.9	14	7
misc04	100	100	6	6	2.8	2.6	3	1.5
misc05	100	100	78	88	5.1	3.8	7.9	4
misc06	100	100	21	20	3.9	3.1	5	2.5
misc07	70.1	53.6	2000	2000	372	465	21.3	10.7
mkc	83.1	97.7	2000	2000	5519	40373.8	87.5	43.8
mod008	78	100	2000	525	82.3	10.6	5	2.5
mod010	100	100	14	13	4.7	4.6	24	12
mod011	78.3	97.4	2000	2000	5743.9	8403.7	14.5	7.3
mod013	100	100	208	111	9.1	1.1	3.4	1.7
modglob	63.3	66.3	2000	2000	159.2	113.3	19.8	9.9
nsrand-ipx	37.4	54.4	2000	2000	6229.3	3731.1	64.7	32.4
nw04	100	100	66	3	161.6	13.6	10.9	5.5
opt1217	5.2	25.4	2000	2000	217.3	169.1	28.7	14.4
p0033	100	100	226	51	9.5	0.7	2.7	1.4
p0040	100	100	22	2	0.9	0	2.9	1.5
p0201	100	100	112	90	9.8	4.9	19.3	9.7
p0282	100	100	91	156	4.8	4	9.8	4.9
p0291	100	100	34	14	1.5	0.2	3.9	2
p0548	100	100	2000	221	160.6	11.7	25.7	12.9
p2756	66.8	89.2	2000	2000	1225.1	1104.2	89.6	44.8
pipex	100	100	682	154	26.6	4.1	4.4	2.2
pk1	28.1	25	2000	2000	94.1	89.6	14	7
pp08a	46.7	48.8	2000	2000	149.5	99.9	39.5	19.8
pp08aCUTS	60.6	63.7	2000	2000	253.1	218.1	33.1	16.6
qiu	88.4	89.5	2000	2000	2460.7	2615.9	21.7	10.9
qnet1	100	100	36	412	21.8	208.5	29.3	14.7
qnet1_o	100	100	177	460	41.3	119	8.4	4.2
rgn	100	100	1290	1286	55.4	41.7	4.1	2.1

continued on the next page

Table 8: *continued*

Instance	Relative gap closed by branching [%]		Nodes processed		Solution time		Average Integer Infeasibilities per Node	
	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
roll3000	45.79	71.22	2000	2000	11901.8	12317.4	149.4	74.7
rout	60.3	73.7	2000	2000	506.9	368.8	31.1	15.6
sample2	100	100	96	106	3.9	0.6	4.5	2.3
sentoy	100	100	183	175	7.3	6.8	4.9	2.5
set1al	26.3	26.3	2000	2000	868.8	908.5	183.3	91.7
set1ch	33.7	33.9	2000	2000	702.3	727.3	117.6	58.8
set1cl	26.2	26.2	2000	2000	871.8	903.5	186.4	93.2
stein15	100	100	75	112	3	1.3	6.4	3.2
stein27	100	80	1966	2000	91.1	53.5	9	4.5
stein45	62.5	50	2000	2000	246.6	316.9	30.3	15.2
stein9	100	100	13	11	0.5	0	3.6	1.8
swath	10.83	37.3	2000	2000	4184.3	6819	47.4	23.7
timtab1	36.4	42.9	2000	2000	324.9	381.6	107.4	53.7
timtab2	23.81	24.23	2000	2000	837.7	1278.8	202.5	101.3
tr12-30	10.1	10.1	2000	2000	2208.2	2261.1	329.7	164.9
vpm1	47.8	100	2000	381	99.4	10.6	12.7	6.4
vpm2	60.1	76.4	2000	2000	149.9	127.5	20.9	10.5

Table 9: Comparison of the gap closed, the number of processed nodes, and the solution time when branching on single variables and on MIG disjunctions. (Cut-and-branch. Two hours limit on the solution time.)

Instance	Relative gap closed by cuts [%]	Relative gap closed by branching [%]		Nodes processed		Solution time	
		Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
10teams	100	0	0	16	19	7200	7200
a1c1s1	54.40	3.27	2.76	15257	14823	7200	7200
aflow30a	50	50	36.2	19741	60993	2370.6	7200
aflow40b	42.5	10.8	11.3	7819	7210	7200	7200
arki001	52.59	1.36	11.14	47562	27953	7200	7200
bell3a	70.3	29.7	29.7	10220	9928	17.5	17.6
bell3b	77.8	22.2	22.2	8192	5780	28.3	15
bell4	92.6	7.4	7.4	329481	2591	2392.8	9.6
bell5	90.6	9.4	9.4	594806	1366	895.3	2.5
blend2	29.3	70.7	70.7	1159	1174	14.3	17.5
bm23	33.4	66.6	66.6	149	104	0.3	0.3
cap6000	63	37	19.9	7901	8225	929.4	7200
danoint	1	20.2	10.9	18292	5098	7200	7200
dcmulti	76.5	23.5	23.5	58	41	4.4	3.3
egout	100	0	0	7	4	0.2	0.2
fiber	95.3	4.7	4.7	644	171	31.7	12.8
fixnet3	99.9	0.1	0.1	9	9	1.4	1.5
fixnet4	90.5	9.5	9.5	123	517	39.4	81.2
fixnet6	83.8	16.2	16.2	562	1639	89.8	215.5
flugpl	14.4	85.6	85.6	1122	26	1	0.1
gen	100	0	0	1	1	0.2	0.2
gesa2	98	2	2	189	110	9.1	8.2
gesa2_o	92.9	7.1	7.1	28315	14092	1717.4	1547.1
gesa3	77.5	22.5	22.5	58	62	6	7.6
gesa3_o	69.7	30.3	30.3	130	230	14	31.4
gt2	99.9	0.1	0.1	23	58	0.4	0.6
harp2	63.5	11.1	8.6	98893	37860	7200	7200
khb05250	99.6	0.4	0.4	7	7	0.9	0.9
l152lav	37.5	62.5	62.5	711	308	44.6	39.9
lp4l	100	0	0	2	2	0.7	0.7
lseu	75.8	24.2	24.2	422	415	1.3	1.4
manna81	100	0	0	1	1	0.5	0.4

continued on the next page

Table 9: *continued*

Instance	Relative gap closed by cuts [%]	Relative gap closed by branching [%]		Nodes processed		Solution time	
		Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
markshare1	0	0	0	1080390	615114	7200	7200
markshare2	0	0	0	1066636	619146	7200	7200
mas74	8.9	80.2	74.4	1348963	1282589	7200	7200
mas76	13.2	86.8	86.8	136806	195978	709.4	1144.2
misc01	5.7	94.3	94.3	162	167	1.1	1
misc02	20	80	80	26	23	0.3	0.2
misc03	16.4	83.6	83.6	203	349	2.6	4.2
misc04	83.8	16.2	16.2	6	7	1.8	2
misc05	51.6	48.4	48.4	61	84	2.3	2.6
misc06	74.6	25.4	25.4	7	13	0.7	1.2
misc07	0.7	99.3	99.3	12610	18605	1450.7	2421.6
mkc	43.9	1	11.2	46257	8531	7200	7200
mod008	86.7	13.3	13.3	614	272	6.5	4.4
mod010	100	0	0	1	1	1.6	1.7
mod011	48.3	34.3	34.9	2810	2843	7200	7200
mod013	73.2	26.8	26.8	90	119	0.9	1
modglob	67.2	32.8	32.8	48578	51301	1664.4	2093.7
nsrand-ipx	70.8	2.5	5	18196	11663	7200	7200
nw04	98.6	1.4	1.4	7	3	185.8	184.2
opt1217	50.2	0	0	349705	349689	7200	7200
p0033	100	0	0	1	1	0.1	0.1
p0040	100	0	0	1	1	0	0
p0201	58.4	41.6	41.6	103	100	4	3.7
p0282	97.5	2.5	2.5	115	41	1.6	1.1
p0291	99.6	0.4	0.4	13	9	0.4	0.3
p0548	94.3	4.1	5.7	436999	50	7200	2
p2756	97.8	2.2	2.2	8643	204	420.1	12.5
pipex	57	43	43	515	241	1	0.6
pk1	0	100	100	138520	221848	4094.3	5999.1
pp08a	90.5	9.5	9.5	1707	2259	52.7	96.3
pp08aCUTS	80.2	19.8	19.8	1187	1994	41.7	94.6
qiu	7.8	92.2	92.2	10031	11768	3671.8	4297.9
qnet1	67	33	33	376	274	648.1	806.9
qnet1_o	75.1	24.9	24.9	633	763	572.7	1033.3
rgn	75	25	25	816	902	3.4	4.1

continued on the next page

Table 9: *continued*

Instance	Relative gap closed by cuts [%]	Relative gap closed by branching [%]		Nodes processed		Solution time	
		Simple disj.	MIG disj.	Simple disj.	MIG disj.	Simple disj.	MIG disj.
roll3000	75.69	2.10	5.56	17355	8503	7200	7200
rout	9.2	75	90.8	163940	17967	7200	1082.5
sample2	40.8	59.2	59.2	93	71	0.4	0.5
sentoy	27.2	72.8	72.8	203	129	0.4	0.4
set1al	100	0	0	9	9	0.5	0.5
set1ch	90.9	1.7	4	75867	63463	7200	7200
set1cl	100	0	0	1	1	0.1	0.1
stein15	0	100	100	76	115	0.5	1
stein27	0	100	100	2088	3855	33.5	89.6
stein45	0	100	100	25034	50926	1350.7	4112.5
stein9	100	0	0	1	1	0	0
swath	32.28	3.02	3.15	21186	16298	7200	7200
timtab1	41.8	10	9.7	446549	201471	7200	7200
timtab2	33.69	2.92	4.10	138659	55279	7200	7200
tr12-30	93.3	0.7	1.9	40481	30145	7200	7200
vpm1	91.3	8.7	8.7	1021	701	21.1	19.5
vpm2	77.2	22.8	22.8	5437	20196	120.6	591.9

Table 10 Comparison of Solution Time (in seconds) by SIMDI, GENDI, and Other MILP Solvers.

Instance	SIMDI	GENDI	CBC	GLPK	MINTO	SYMPHONY	SCIP-S	SCIP-L	SCIP-C	CPLEX
10teams	7200*	7200*	375	7200*	7200*	7200*	1307	5048	39	5
cap6000	929	7200*	126	7200*	780	7200*	60	479	42	22
markshare1	7200*	7200*	7200*	227	7200*	7200*	903	1334	11	14
markshare2	7200*	7200*	7200*	7200*	7200*	7200*	832	7200*	805	39
mas74	7200*	7200*	7200*	7200*	7200*	7200*	2105	7200*	1530	1522
mas76	709	1144	1603	7200*	7200*	4878	220	604	185	149
misc07	1451	2422	657	98	424	861	66	103	71	11
mod011	7200*	7200*	802	7200*	3750	1218	1383	1882	300	94
nw04	186	184	76	717	7200*	845	4093	3232	326	26
pk1	4094	5999	3607	7200*	7200*	3140	158	820	197	86
qiu	3672	4298	4374	7200*	800	5779	424	1800	197	37

* The solver could not solve the instance in the alloted two-hour time.

** The following MILP solvers have been used:
- CBC 1.00: http://www.coin-or.org/projects/Cbc.xml
- GLPK 4.9: http://www.gnu.org/software/glpk/glpk.html
- MINTO 3.1: http://coral.ie.lehigh.edu/ minto/
- SYMPHONY 5.1a: http://www.coin-or.org/projects/SYMPHONY.xml
- SCIP 0.81: http://scip.zib.de/
- CPLEX 10.0: http://www.ilog.com/products/cplex/ (Parameters: mipgap=0, absmipgap=1e-9)

CBC, SYMPHONY, and MINTO use Clp as an LP solver. SCIP uses Soplex (SCIP-S), Clp (SCIP-L), or Cplex (SCIP-C) as an LP solver.

*** Source of data: [26]

References

1. K. Aardal, R. E. Bixby, C. A. J. Hurkens, A. K. Lenstra, and J. W. Smeltink. Market split and basis reduction: Towards a solution of the Cornuéjols-Dawande instances. *Informs Journal on Computing*, 12(3):192–202, 2000.
2. K. Aardal, C. A. J. Hurkens, and A. K. Lenstra. Solving a system of linear diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research*, 25(3):427–442, 2000.
3. T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operation Research Letters*, 33:42–54, 2005.
4. T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. Technical Report 05-28, Zuse Institute Berlin, Takustr. 7, Berlin, 2005. Available from http://miplib.zib.de.
5. K. Andersen, G. Cornuéjols, and Y. Li. Reduce-and-split cuts: Improving the performance of mixed integer Gomory cuts. *Management Science*, 51:1720–1732, 2006.
6. E. Balas. Intersection cuts - a new type of cutting planes for integer programming. *Operations Research*, 19:19–39, 1971.
7. E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.
8. E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.
9. E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J. Lawrence, editor, *OR 69: Proc. Fifth Int. Conf. Oper. Res.*, pages 447–454, London, 1970. Tavistock Publications.
10. R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.
11. R.E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mixed integer programming: A progress report. In M. Grötschel, editor, *The Sharpest Cut: The impact of Manfred Padberg and his work*, MPS/SIAM Series in Optimization, pages 309–326. SIAM, 2004.
12. W. Cook, R. Kannan, and A Schrijver. Chvátal closures for mixed integer programs. *Mathematical Programming*, 47:155–174, 1990.
13. G. Cornuéjols and M. Dawande. A class of hard small 0-1 programs. In R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, editors, *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference*, Lecture notes in Computer Science 1412, pages 284–293. Springer-Verlag, Berlin, 1998.
14. G. Cornuéjols, L. Liberti, and G. Nannicini. Improved strategies for branching on general disjunctions. Technical report, Working paper, July 2008, revised April 2009.
15. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
16. R. E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, Rand Corporation, 1960.
17. M. Grötschel, L. Lovász, and A. Schrijver. *Progress in combinatorial optimizarion*, chapter Geometric methods in combinatorial optimization, pages 167–183. Academic Press, Toronto, 1984.
18. H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
19. R. Laundy, M. Perregaard, G. Tavares, H. Tipi, and A. Vazacopoulos. Solving hard mixed integer programming problems with Xpress-MP: A MIPLIB 2003 case study. Technical report, 2009.
20. A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 1982.
21. J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11:172–187, 1999.
22. L. Lovász and H. E. Scarf. The generalized basis reduction algorithm. *Mathematics of Operations Research*, 17:751–764, 1992.
23. A. Mahajan and T. K. Ralphs. *Operations Research and Cyber-Infrastructure*, chapter Experiments with Branshing using General Disjunctions, pages 101–118. Springer, 2009.
24. H. Marchand and L.A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49:363–371, 2001.
25. S. Mehrotra and Z. Li. On generalized branching methods for mixed integer programming. Technical report, Northwestern University, Evanston, Illinois 60208, 2004.

26. H. Mittelman. Benchmarks for optimization software. http://plato.la.asu.edu/bench.html, 23 January 2006.
27. J. Owen and S. Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear program. *Computational Optimization and Applications*, 20:159–170, 2001.
28. MIPLIB website of Zuse Institute Berlin.
URL: http://miplib.zib.de/miplib3/miplib_prev.html.