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

Miroslav Karamanov · Gérard Cornuéjols

Branching on General Disjunctions

Received: date / Accepted: date / / Manuscript date: 19th July 2005

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 branching on general disjunctions is more efficient than branching on variables for a majority of the instances, based on solution time and tree size for the instances that could be solved to optimality within a given time limit, and integrality gap closed for the remaining instances.

Keywords branch and bound · branch and cut · branching · split disjunctions

Mathematics Subject Classification (2000) 90C10

1 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 DMI-0352885 and ONR grant N00014-97-1-0196.

Miroslav Karamanov
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213.
E-mail: miroslav@andrew.cmu.edu

Gérard Cornuéjols
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213,
and LIF, Faculté des Sciences de Luminy, 13288 Marseille, France.
E-mail: gc0v@andrew.cmu.edu

the last decade due to advances in hardware but mostly due to modifications in the algorithm. 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 basic solution. 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* [11].

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 practically impossible. 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 resonable 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 [14]. 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. 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 [20]. Our approach is explained in detail in Section 3.2 and a short introduction to intersection cuts is included in Section 3.1. A review of related earlier work is present in Section 2.

In Section 4, 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. We also test an algorithm combining branching on variables and general disjunctions.

There is no doubt that branching on general disjunctions can reduce the amount of enumeration compared to branching on single variables, provided that we know the right disjunctions to branch on. In this paper, we show that such disjunctions can be generated and applied without much computational overhead.

2 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 [16], Grötschel, Lovász, and Schrijver [15], and Lovász and Scarf [19]. Finding thin directions is done by lattice basis reduction based on the work of Lenstra, Lenstra, and Lovász [17]. 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 [12]. 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 [21].

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 [13], 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 [22] 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.

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 a computationally efficient algorithm that competes with branching on single variables not only in terms of tree size but in terms of solution time.

3 Branching on general disjunctions

3.1 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, Cornuejols, 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 $N_I \subseteq N := \{1,2,\dots,n\}$. 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. 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.

Define a *split disjunction* $D(\pi, \pi_0)$ to be a disjunction of the form $\pi^T x \leq \pi_0 \ \vee\ \pi^T x \geq \pi_0 + 1$, 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.

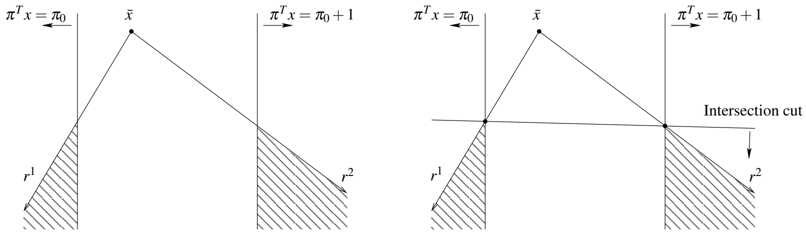

Fig. 1 Deriving the intersection cut

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*, can be used as a measure of the quality of the cut. To our knowledge, it was first used 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 $D(\hat{\pi}^i, \hat{\pi}^i_0)$:

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

This disjunction can also be obtained by strengthening the simple disjunction $D(\pi^i = e_i,\ \pi_0^i = \lfloor \bar{x}_i \rfloor)$ on the non-basic integer variables where the affected coefficients are modified so that the distance cut off is maximized.

3.2 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 optimal objective values computed at the children nodes that would result from branching on this variable. Specifically, let $\bar{x}_1$ and $\bar{x}_2$ be the optimal solutions for the first and second child, respectively, and let $z(\bar{x}_1) = c^T\bar{x}_1$ and $z(\bar{x}_2) = c^T\bar{x}_2$ be the corresponding objective values. Two typical functions are $\min(z(\bar{x}_1), z(\bar{x}_2))$ or $\frac{1}{2}[z(\bar{x}_1) + z(\bar{x}_2)]$. Experiments with other options have been reported in the literature. As an example, Linderoth and Savelsberg [18] show good results with the sum $\frac{1}{3}[\min(z(\bar{x}_1), z(\bar{x}_2)) + z(\bar{x}_1) + z(\bar{x}_2)]$, while Achterberg, Koch, and Martin [3] propose $\frac{1}{6}[4\min(z(\bar{x}_1), z(\bar{x}_2)) + z(\bar{x}_1) + z(\bar{x}_2)]$. 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 and 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 $\min(z(\bar{x}_1), z(\bar{x}_2))$, 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

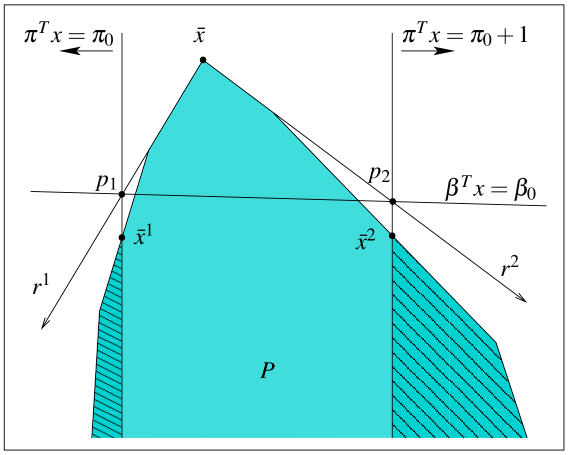

Fig. 2 The LP bound obtained by branching is different from the one obtained by cutting.

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 the corresponding optimal basic solutions. Let $p_1 := \arg\min\{c^T x : x \in P(B) \text{ and } \pi^T x \le \pi_0\}$ and $p_2 := \arg\min\{c^T x : x \in P(B) \text{ and } \pi^T x \ge \pi_0 + 1\}$. 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 $\min\{c^T x : x \in P(B) \text{ and } \beta^T x \le \beta_0\}$, provides a lower bound for any measure of 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. One idea would be to come up with a heuristic search routine. Instead, we suggest to concentrate on a finite class of general disjunctions which we can enumerate — 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

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

formula (6). Third, as we mention in Section 3.1, 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 ($k = 10$). 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 roughly equivalent to applying strong branching to the k most fractional variables.

4 Experimental results

The test set for the experiments we report is the union of the Mixed Integer Linear Programming libraries MIPLIB 2.0 [23], 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 tested. We also exclude some very difficult instances — those for which less than 50 nodes can be processed in one hour. These two groups of instances are considered in the first experiment, where the gap closed at the root is compared 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 80. 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 a IBM IntellistationZ Pro computer with an Intel Xeon 3.2GHz CPU and 2GB RAM. The MILP solver used is COIN-OR BCP, where some user methods are modified for the purpose of our experiment. The LP solver is COIN-OR CLP. Mixed integer Gomory cuts were generated using the cut generator in the library COIN-OR CGL.

In our experiments, we compare branching on single variables to branching on general 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 lower bound at interruption and the lower bound at the root node, and the *relative* (*percentage*) *gap closed*: the absolute gap closed relative to the gap at the root.

In the first experiment, we study the gap closed after branching at the root node. In the second experiment, we study the gap closed and the number of active nodes left after branching for five levels. We apply pure branch and bound in these experiments in order to avoid the influence of adding different cutting planes. This ensures a clean comparison between the two branching procedures. Finally, we test our idea in a cut-and-branch framework and propose a combination of the two branching methods. We also study the increase in solution time per node caused by branching on general disjunctions.

4.1 Gap closed at the root

We first compare the gap closed at the root by ordinary branching on single variables and branching on MIG disjunctions. When branching on variables, we select the (up to) ten most fractional variables — those integer variables that have fractional parts closest to 0.5. Then, the variable to branch on is chosen by strong branching with objective to maximize the minimum gap closed in the children, i.e. the variable with maximum $\min(z(\bar{x}^1), z(\bar{x}^2))$ is selected. In this implementation, if branching on a variable creates an infeasible child, this variable is preferred to all others. For short, we call this setup for branching on simple disjunctions *SIMDI*.

When branching on general disjunctions, we select the (up to) ten MIG disjunctions with best distance cut off. Again, the object to branch on is chosen by strong branching following the same rules as before. This guarantees fair comparison for variables 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*.

The comparison of the absolute gap closed shows that GENDI performs better for 59 out of 108 instances, while SIMDI is better for 20 instances. For 90 instances the optimal objective value is known and this allows to compute the percentage gap closed. The average percentage gap closed by GENDI and SIMDI over this set of instances is 17.9% and 11.7%, respectively. If we consider only those 65 instances for which the gap closed by both methods differ, the average gap closed by GENDI and SIMDI is 16.8% and 8.1%, resp. Detailed results are given in Table 6.

A closer look at the cases when one method substantially dominates the other shows that for 11 instances GENDI closes a positive amount of gap while SIMDI cannot close any gap. In contrast, there are only 3 instances for which GENDI is unsuccessful while SIMDI manages to improve the lower bound. When we look at 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 12 instances, while SIMDI is an order of magnitude better for 2 instances. Some examples where GENDI is clearly more successful are: `10teams`, `blend2`, `fiber`, `gt2`, `mod010`, `momentum1`, 2, and 3, `msc98-ip`, `p0282`, `nw04`, and `swath`. Similarly, SIMDI dominates in: `arki001`, `bell3b`, `mzzv42z`, and `roll3000`.

Table 1 Comparison of SIMDI and GENDI after five levels of branching

	SIMDI	GENDI
Absolute gap closed		
Count better	19	48
Percentage gap closed		
(Instances with known optimal solution)		
Average	24.3	29.1
Count better	18	46
Active nodes at level 5		
Average	25.8	18.0
Count better	10	43
Gap closed and active nodes together		
Count better	6	33

These results are a strong indication that GENDI performs better than SIMDI. Next, we test whether these good results at the root proliferate throughout the tree by branching for five levels and by branching for up to two hours. Branching for five levels helps observe another good effect of branching on MIG disjunctions: a decrease in the number of active nodes.

4.2 Branching for five levels

In the second experiment, we branch at the top five 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 observed at the root as well. But now we observe an interesting secondary effect: branching on general 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 fifth level.

Detailed results of the experiment are shown in Table 7. Summary results are shown in Table 1. 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 gap closed, SIMDI dominates in 19 cases, GENDI in 48 cases out of 80. The average gap closed by SIMDI and GENDI is 24.3% and 29.1%, resp, over the set of 75 instances for which the optimal objective value is known. These results support our earlier observation that GENDI closes more gap.

It is interesting to observe that GENDI also produces a smaller number of active nodes at the fifth level, on average. Out of the maximum possible 32 nodes, SIMDI generates 25.8 while GENDI generates 18. On this criterion, SIMDI performs better in 10 cases while GENDI does this in 43 cases. This effect is important not by itself but in combination with improvement in the gap. Combining both criteria, SIMDI results in a larger gap closed and a smaller number of active nodes in only six cases, while GENDI achieves this in 33 cases.

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

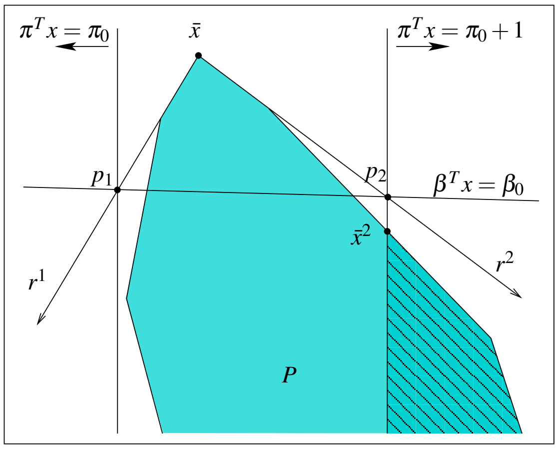

Fig. 4 Disjunction with only one feasible child

happens at every level of the branching tree, resulting in a single node at level five. Sometimes, this is combined with an impressive improvement of the gap closed over SIMDI, e.g. `manna81`, `p0033`, `ste1al`, `set1cl`, `seymour`, and `swath`. See also `lseu`, `roll3000`, `set1ch`, and `sp97ar`.

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 and only the fact that we apply pure branch and bound leaves us such good opportunities. This could be true in some cases but in others the disjunction inequality is stronger than the corresponding MIG cut. Figure 4 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 close more gap than the corresponding cut, as we showed in Section 3.2.

We do not consider branching on general disjunctions a substitute of cutting planes. Our procedure comes into play when branch and cut decides to start branching. And then, a routine that tends to avoid branching (and duplicating the problem) when some part of the gap can be closed by a cut could be very useful.

4.3 Cut and branch

Now, we test the performance of our approach in a cut-and-branch framework. We generate ten rounds of mixed integer Gomory cuts before proceeding to branching. Branching is done as before: at each node, up to ten branching objects are selected for strong branching. We make two minor modifications in the algorithm for branching on general disjunctions. The first aims at avoiding 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 solve the instances and compare the tree size and the running time. For those 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.

Branching on general disjunctions performs better than branching on variables for a majority of the instances but not for all ot them. This suggests that a combined approach can build upon the advantages of both "pure" algorithms. We combine them in the following way: at every node of the branching tree, we select five most fractional variables and five disjunctions with maximum distance cut off. Then we apply strong branching to all these branching objects. We call this algorithm *COMBI* and compare it to the other two. Detailed results from this experiment are presented in Table 8.

Out of 80 test instances, SIMDI solved 35 in the alloted time, GENDI solved 50, and COMBI solved 50. GENDI solved all instances solved by SIMDI except one, `stein45`. COMBI solved all instances solved by SIMDI and could not solve only one instance solved by GENDI, `bell4`.

We ranked the methods according to the amount of gap closed for each instance (see Table 2). SIMDI ranked first 46 times and GENDI 67 times. GENDI closed more gap than SIMDI in 33 cases while SIMDI dominated in 8 cases. In addition, we compute the ratios of the absolute gap closed by GENDI and SIMDI for all instances and look at the geometric mean. (Values of infinity, corresponding to zero gap closed by SIMDI, are excluded.) Since part of this gap is closed by the cuts added at the root, a better measure for comparing the different branching methods is the ratio between the gap closed only by branching. These ratios are shown in the secong column of Table 3. GENDI closes 15% more gap than SIMDI on average, and if we consider only the gap closed by branching, the improvement is 26%. These results indicate that GENDI performs better than SIMDI on this test set.

The performance of the combined approach, COMBI, tends to be close to the better of the other two methods. It equaled the best gap closed by SIMDI and GENDI for 53 (out of 80) instances and even improved it in six cases. It ranked last only three times demonstrating more consistent behavior. Comparing the ratios of

Table 2 Ranks according to gap closed.

Rank	SIMDI	GENDI	COMBI
1	46	67	59
2	2	7	18
3	32	6	3

Table 3 Geometric mean of the ratios of three parameters over all 80 instances.

Ratio	$\frac{\text{GENDI}}{\text{SIMDI}}$	$\frac{\text{COMBI}}{\text{SIMDI}}$	$\frac{\text{COMBI}}{\text{GENDI}}$
Absolute gap closed	1.15	1.14	0.99
Absolute gap closed by branching	1.26	1.23	0.97
Average solution time per node	1.24	1.29	1.04

the gap closed (Table 3), we conclude that COMBI performs a few percent worse than GENDI on average but still outperforms SIMDI significantly.

Table 3 shows the ratio of the average solution time per node of the three branching procedures as well. We observe that GENDI requires 24% more time to solve a node than SIMDI (including the time for strong branching). The corresponding statistic for COMBI is 29%. Careful analysis shows that the increased solution time per node caused by branching on general disjunctions is offset by the reduced amount of enumeration and the increased amount of integrality gap closed.

To make a more detailed analysis, we split the test set into three sets: instances solved by all methods, those not solved by any method, and the remaining. This analysis is presented below. In summary, the two algorithms using branching on general disjunctions solved 43% more instances within the two hour time limit compared to branching on variables. For those instances solved by all algorithms, GENDI and COMBI are more efficient in terms of solution time and tree size, on average. For the instances not solved by any method, GENDI and COMBI closed more gap for the same time. These observations are a strong indication that the proposed procedures for branching on general disjunctions perform better than branching on variables.

Instances solved by all methods

Thirty five instances were solved by the three algorithms which allows to compare the size of the branching trees and the total solution times. One instance, `manna81`, was solved solely by cutting planes and is excluded from this analysis.

Table 4 contains the geometric mean of the ratios of solution time and tree size. It shows that GENDI decreases the solution time by 28% compared to SIMDI. COMBI is more efficient and solves these instances twice as fast as SIMDI. The decrease in the tree size is even more significant. GENDI and COMBI explore 65% and 42% of the nodes explored by SIMDI, respectively. At the two extremes, GENDI and COMBI are at least five times more efficient in solving `dcmulti`,

Table 4 Geometric mean of the ratios of solution time and tree size over 35 instances solved by all methods

Ratio	GENDI / SIMDI	COMBI / SIMDI	COMBI / GENDI
Solution time	0.72	0.50	0.70
Tree size	0.65	0.42	0.65

`flugpl`, `gesa3`, `gesa3_o`, `lseu`, and `p0033`, while SIMDI is about five times faster with `air05` and `qnet1_o`. COMBI is faster than the other two algorithms in most cases. As an extreme example, it solves `nw04` in 48 seconds while SIMDI and GENDI require 60 and 113 minutes, respectively.

In conclusion, we observe that branching on general disjunctions causes significant reduction in the total solution time compared to branching on variables. The combined approach performs notably better than the other two algorithms on this test set.

Instances solved by some but not all methods

Fifteen instances were solved by some but not all algorithms. These are relatively difficult instances on which branching on general disjunctions performs significantly better and in many cases orders of magnitude better. Thirteen of them were solved by GENDI and COMBI but not solved by SIMDI within two hours. The geometric mean of the solution times is 228 seconds by GENDI and 301 seconds by COMBI. Some cases of extreme reduction of the solution time are: `bell5` solved in 9.9 s and 2055 processed nodes by GENDI; `fiber` solved in 18 s and 203 nodes by GENDI, and in 11.3 s and 172 nodes by COMBI; `gt2` solved in 1.1 s and 120 nodes by GENDI, and in 0.5 s and 41 nodes by COMBI. All these instances could not be solved in two hours and hundreds of thousands of nodes when branching on variables. Another example is `10teams` where SIMDI closed only 28.6% of the gap in two hours, while GENDI solved the problem in 75 minutes and COMBI solved it in four minutes. A significant difference in the gap closed is observed for `rout`, `p0282`, and `fiber`, as well.

One instance, `bell4`, was solved only by GENDI. It was solved in 51.2 s and 3490 nodes while the other methods could not solve it in two hours and more than 250,000 nodes. SIMDI managed to close 96.6% of the gap and COMBI closed 99.6%. The last instance from this group, `stein45`, was solved by SIMDI in 103 minutes and by COMBI in 80 minutes but not solved by GENDI.

These observations clearly reveal the properties of branching on general disjunctions to close larger amounts of the integrality gap and to decrease the amount of enumeration.

Instances not solved by any method

Twenty eight instances were not solved by any method within the time limit. We compare the gap closed by the three algorithms in the two-hour interval of time. None of the algorithms closed any gap by branching for three instances: `liu`,

Table 5 Geometric mean of the ratios of gap closed over 23 instances not solved by any method

Ratio	$\frac{\text{GENDI}}{\text{SIMDI}}$	$\frac{\text{COMBI}}{\text{SIMDI}}$	$\frac{\text{COMBI}}{\text{GENDI}}$
Absolute gap closed	1.30	1.25	0.97
Absolute gap closed by branching	1.59	1.43	0.90

`markshare1`, and 2. For two instances, `opt1217` and `p2756`, SIMDI closed no gap while GENDI and COMBI closed positive amounts of gap. These five instances are excluded from the summary statistics presented in Table 5. For the remaining 23 instances, GENDI and COMBI closed 30% and 25% more gap than SIMDI, respectively. If we consider only the gap closed in the branching phase of cut and branch, the improvement in the gap is 59% and 43%, respectively. These results show that branching on general disjunctions closes significantly more gap than branching on variables for the same time interval, on average.

5 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 is used for pre-selection of the most promising disjunctions before strong branching.

We test these ideas in experiments with 80 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) smaller tree size and solution time for the instances solved to optimality, and (ii) larger amount of gap closed for the other instances. The test results clearly 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.

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. More work can be done on criteria for evaluating disjunctions as well.

Acknowledgements

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

Table 6: Comparison of the gap closed at the root node by branching on a single variable and on a MIG disjunction.

Instance	Simple disjunction Absolute gap closed	Simple disjunction Relative gap closed [%]	MIG disjunction Absolute gap closed	MIG disjunction Relative gap closed [%]
10teams	0	0	4	57.14
a1c1s1	42.45	-	9.87	-
aflow30a	6.09	3.49	11.22	6.42
aflow40b	6.77	4.17	4.49	2.76
air01	53	100	53	100
air02	170	100	170	100
air03	1295.75	100	1295.75	100
air04	43.67	7.26	53.73	8.93
air05	72.54	14.61	31.11	6.27
air06	21.97	67.32	25.78	78.99
arki001	42.2	-	0	-
atlanta-ip	0	-	0.18	-
bell3a	3174.32	20.03	1857.8	11.72
bell3b	316638.07	82.89	13855.95	3.63
bell4	360675.73	64.79	122948.47	22.08
bell5	298008.96	83.25	298008.96	83.25
blend2	0.01	0.84	0.13	19.72
bm23	0.91	6.8	0.29	2.13
cap6000	69.83	44.38	85.59	54.4
dano3mip	0.09	-	0.02	-
danoint	0.03	1	0.03	1.01
dcmulti	921.07	21.9	647.89	15.4
ds	0.12	-	0.13	-
egout	5.77	1.38	16	3.82
fast0507	0.06	3.03	0.05	2.43
fiber	2648.8	1.06	40055.4	16.03
fixnet3	84.62	0.75	365.02	3.24
fixnet4	29.43	0.63	95.5	2.04
fixnet6	22.44	0.81	23.79	0.85
flugpl	689.44	2.01	1874.28	5.46
gen	9.23	5.03	65.21	35.57
gesa2	3636.77	1.2	8160.05	2.69
gesa2_o	3636.77	1.2	8160.05	2.69
gesa3	5348.27	3.4	10051.01	6.39
gesa3_o	5348.27	3.4	10051.01	6.39
glass4	0	-	0	-
gt2	359.47	4.66	4816.57	62.51
harp2	4413.48	0.97	21196.43	4.67
khb05250	1670000	15.15	670982	6.09
l152lav	5.83	8.89	9.79	14.91
liu	214	-	214	-
lp4l	9.11	37.16	20.44	83.41
lseu	12	4.21	98.06	34.37
manna81	0.5	0.38	0.5	0.38
markshare1	0	0	0	0
markshare2	0	0	0	0
mas74	40.84	3.1	75.5	5.73
mas76	20.59	1.85	62.27	5.6
misc01	0	0	13.24	2.61
misc02	0	0	25	3.68
misc03	0	0	92.5	6.38
misc04	5.65	54.92	5.65	54.92
misc05	1.2	2.24	14.4	26.87
misc06	2.08	22.73	2.08	22.73
misc07	0	0	0	0
mkc	0	0	0	0
mod008	0.24	1.48	0.02	0.14
mod010	3.59	22.53	13.52	84.92
mod011	415328.41	5.49	709329.48	9.38
mod013	1.36	5.45	6.13	24.59

continued on next page

Table 6: *continued*

Instance	Simple disjunction Absolute gap closed	Simple disjunction Relative gap closed [%]	MIG disjunction Absolute gap closed	MIG disjunction Relative gap closed [%]
modglob	8969.42	2.9	8969.42	2.9
momentum1	10.41	-	3199.99	-
momentum2	0.16	-	690.39	-
momentum3	0.26	-	4.9	-
msc98-ip	0	-	27360	-
mzzv11	0	0	8.49	0.69
mzzv42z	48.88	4.51	0	0
net12	3.65	1.85	1.87	0.95
nsrand-ipx	0	0	110	4.74
nw04	22	3.99	259.33	47.04
opt1217	0	0	0.27	6.78
p0033	29.93	5.27	205.76	36.2
p0040	10.49	4.55	160.91	69.83
p0201	0	0	310	41.89
p0282	181.18	0.22	32843.91	40.28
p0291	1031.33	29.31	1031.33	29.31
p0548	12.68	0.15	12.68	0.15
p2756	0	0	0	0
pipex	0.07	0.48	4.08	28.1
pk1	0	0	0	0
pp08a	142.14	3.09	80	1.74
pp08aCUTS	68.03	3.64	81.44	4.36
protfold	0.36	-	0.64	-
qiu	0	0	0	0
qnet1	42.99	2.45	213.47	12.16
qnet1_o	474.01	12.05	474.01	12.05
rd-rplusc-21	0	-	0	-
rgn	0	0	0.8	2.4
roll3000	0.94	-	0	-
rout	2.34	2.44	2.34	2.44
sample2	15	11.72	15	11.72
sentoy	7.44	11.06	10.79	16.04
set1al	15.64	0.33	37.93	0.8
set1ch	95.87	0.43	722.86	3.21
set1cl	15.64	0.33	37.93	0.79
seymour	0.36	1.86	0.49	2.57
sp97ar	54195.09	-	249175.81	-
stein9	0	0	0	0
stein15	0	0	0	0
stein27	0	0	0	0
stein45	0	0	0	0
swath	0.13	-	4.12	-
t1717	146.93	-	86.64	-
timtab1	24290	3.3	24290	3.3
timtab2	23443	-	5462	-
tr12-30	367.03	0.32	300	0.26
vpm1	0.25	5.39	0.66	14.48
vpm2	0.06	1.68	0.1	2.7

Table 7: Comparison of the gap closed and the number of active nodes after five levels of branching.

Instance	Simple disjunctions			MIG disjunctions		
	Nodes at level 5	Absolute gap closed	Relative gap closed [%]	Nodes at level 5	Absolute gap closed	Relative gap closed [%]
10teams	32	0.0	0.0	10	4.0	57.1
a1c1s1	32	336.3	-	32	57.3	-
aflow30a	28	26.1	15.0	4	32.8	18.7
aflow40b	19	12.3	7.6	16	32.8	20.2
air04	23	212.4	35.3	31	230.1	38.3
air05	32	246.9	49.7	32	192.9	38.9
arki001	32	148.6	-	32	0.0	-
bell3a	14	9645.2	60.8	8	9128.8	57.6
bell3b	2	338929.2	88.7	8	258394.2	67.6
bell4	5	494769.1	88.9	17	416779.6	74.9
bell5	4	299455.8	83.6	1	307175.6	85.8
blend2	16	0.2	28.2	27	0.2	36.5
bm23	32	3.5	25.7	16	6.2	46.5
cap6000	32	106.8	67.9	32	106.8	67.9
danoint	32	0.0	1.7	32	0.0	1.7
dcmulti	31	2048.0	48.7	15	821.9	19.5
egout	1	29.3	7.0	1	61.0	14.6
fast0507	12	0.3	17.3	6	0.2	11.5
fiber	32	6974.0	2.8	31	72697.2	29.1
fixnet3	32	370.5	3.3	32	1545.9	13.7
fixnet4	32	145.0	3.1	32	415.0	8.9
fixnet6	32	60.5	2.2	32	95.5	3.4
flugpl	14	5097.6	14.9	11	6615.9	19.3
gen	6	33.0	18.0	1	117.4	64.0
gesa2	32	14251.6	4.7	21	39091.6	12.9
gesa2_o	32	12133.5	4.0	19	39091.6	12.9
gesa3	32	32480.0	20.6	13	43313.2	27.5
gesa3_o	32	32480.0	20.6	13	43313.2	27.5
glass4	12	0.0	-	10	0.0	-
gt2	32	6182.9	80.2	12	4940.9	64.1
harp2	32	24867.7	5.5	2	32823.5	7.2
khb05250	32	5183898.0	47.0	32	5097557.0	46.3
l152lav	28	30.7	46.7	28	30.7	46.7
liu	32	214.0	-	32	214.0	-
lseu	28	25.0	8.7	5	160.9	56.4
manna81	32	1.0	0.8	1	2.5	1.9
markshare1	32	0.0	0.0	32	0.0	0.0
markshare2	32	0.0	0.0	32	0.0	0.0
mas74	32	147.2	11.2	32	159.4	12.1
mas76	32	99.6	9.0	32	111.9	10.1
misc01	7	36.5	7.2	6	40.0	7.9
misc03	8	1450.0	100.0	16	67.5	4.7
misc05	20	17.4	32.5	26	16.5	30.8
misc07	24	15.3	1.1	18	11.3	0.8
mkc	28	1.7	3.5	1	0.0	0.0
mod008	25	16.1	100.0	25	16.1	100.0
mod011	32	1230186.8	16.3	32	2746460.5	36.3
mod013	32	6.9	27.5	16	13.6	54.4
modglob	32	25488.7	8.2	32	32258.2	10.4
net12	28	9.4	4.8	15	15.0	7.6
nsrand-ipx	32	0.0	0.0	28	415.0	17.9
nw04	8	138.0	25.0	8	138.0	25.0
opt1217	32	0.0	0.0	32	1.0	25.4
p0033	12	179.4	31.6	1	388.9	68.4
p0201	30	120.0	16.2	31	480.0	64.9
p0282	26	934.3	1.1	27	77433.0	95.0
p0548	1	64.3	0.8	1	123.8	1.5
p2756	7	0.0	0.0	1	1.0	0.2
pipex	32	1.5	10.1	23	5.6	38.5
pk1	32	0.0	0.0	32	0.0	0.0
pp08a	32	673.9	14.6	32	380.0	8.3

continued on next page

Table 7: *continued*

Instance	Simple disjunctions Nodes at level 5	Absolute gap closed	Relative gap closed [%]	MIG disjunctions Nodes at level 5	Absolute gap closed	Relative gap closed [%]
pp08aCUTS	32	240.9	12.9	32	374.7	20.0
qiu	31	272.7	34.1	29	290.7	36.4
qnet1	32	665.7	37.9	28	683.8	38.9
qnet1_o	24	1905.4	48.4	24	1884.7	47.9
rd-rplusc-21	1	0.0	-	1	0.0	-
rgn	24	33.4	100.0	32	8.8	26.3
roll3000	20	3.4	-	3	226.4	-
rout	32	14.9	15.6	23	20.7	21.7
sample2	26	86.7	67.7	32	63.0	49.2
sentoy	32	27.0	40.2	32	25.2	37.4
set1al	32	76.0	1.6	1	173.5	3.7
set1ch	32	502.1	2.2	2	2740.4	12.2
set1cl	32	76.0	1.6	1	173.5	3.6
seymour	32	1.0	5.2	1	1.9	10.2
sp97ar	32	424694.5	-	2	827703.8	-
stein15	29	2.0	100.0	2	2.0	100.0
stein27	32	0.0	0.0	6	0.0	0.0
stein45	32	0.0	0.0	5	0.0	0.0
swath	24	1.4	-	1	13.7	-
timtab1	24	70664.0	9.6	13	65088.0	8.8
timtab2	21	52373.0	-	11	62952.0	-
tr12-30	32	1833.2	1.6	32	1063.7	0.9
vpm1	32	0.5	10.5	5	1.9	40.9
vpm2	32	0.5	12.7	6	0.5	13.7

Table 8: Branching on simple disjunctions, MIG disjunctions, and combined in a cut-and-branch framework with time limit of two hours.

Instance	Relative gap closed by cuts [%]	Simple disjunctions: Solution time [s]	Simple disjunctions: Nodes processed	Simple disjunctions: Rel. gap closed [%]	MIG disjunctions: Solution time [s]	MIG disjunctions: Nodes processed	MIG disjunctions: Rel. gap closed [%]	Combined: Solution time [s]	Combined: Nodes processed	Combined: Rel. gap closed [%]
10teams	0	7200	4550	28.6	4526.8	1255	100	240.5	99	100
a1c1s1	2518.2 *	7200	13437	3516.8 *	7200	14710	3033.4 *	7200	12329	2863.7 *
aflow30a	5.8	7200	60379	85	7200	33485	82.7	7200	31054	86.1
aflow40b	0	7200	11795	43	7200	5180	50.4	7200	5679	48.5
air04	0	7200	631	96.4	7200	432	82.9	7200	410	82.5
air05	0	1029.9	274	100	5081.2	1089	100	1981.1	413	100
bell3a	58.6	129	28754	100	123.8	28643	100	126.2	28643	100
bell3b	79.3	7200	260898	96.6	105.3	8811	100	1687.6	60447	100
bell4	45	7200	268780	96.6	51.2	3490	100	7200	288096	99.6
bell5	89.1	7200	1517347	99.5	9.9	2055	100	4964.6	400625	100
blend2	0	41.7	1915	100	11.2	519	100	22	747	100.1
bm23	28.7	0.8	210	100	0.4	123	100	0.6	126	100
cap6000	0	7200	34572	99.9	6918.6	34719	100	6931.4	34718	100
danoint	0	7200	7614	23.8	7200	4800	16.9	7200	6335	22.5
dcmulti	10.4	273.2	5475	100	4.9	113	100	8.2	187	100
egout	51.8	3.7	422	100	2.9	223	100	1.6	120	100
fast0507	0	7200	98	25.4	7200	72	15.7	7200	55	18.4
fiber	35.4	7200	122832	66.5	18	203	100	11.3	172	100
fixnet3	2.1	7200	173254	20.8	7200	160591	49.7	7200	164895	46.4
fixnet4	2.5	7200	168267	19.6	7200	142080	32.6	7200	147986	32.8
fixnet6	1.1	7200	154719	13.7	7200	130749	22.9	7200	137584	19.4
flugpl	14.5	2.6	1050	100	0.1	38	100	0.1	34	100
gen	50.4	20.1	246	100	17.3	119	100	13	107	100
gesa2	83.4	7200	52912	92.6	270.2	1867	100	413.4	2904	100
gesa2_o	82.8	7200	56691	93.8	475.8	3268	100	128	949	100
gesa3	48.1	127.5	843	100	19.3	140	100	19.2	142	100
gesa3_o	57.4	579.5	4246	100	22.8	168	100	15.7	121	100
glass4	0 *	7200	274535	2200 *	7200	251180	1863.1 *	7200	260218	1967.3 *
gt2	67.2	7200	624148	83.9	1.1	120	100	0.5	41	100
harp2	0	7200	80904	38.5	7200	63573	39.7	7200	56013	43.8
khb05250	0	39.6	1175	100	35.7	1071	100	39.6	1172	100
l152lav	0	32.6	196	100	34.9	217	100	12.9	87	100
liu	214 *	7200	39865	214 *	7200	37436	214 *	7200	44918	214 *
lseu	47.8	79.2	12297	100	14.3	2546	100	8.7	1483	100

continued on next page

Table 8: *continued*

Instance	Relative gap closed by cuts [%]	Simple disjunctions Solution time [s]	Nodes processed	Rel. gap closed [%]	MIG disjunctions Solution time [s]	Nodes processed	Rel. gap closed [%]	Combined Solution time [s]	Nodes processed	Rel. gap closed [%]
manna81	100	0.4	1	100	0.3	1	100	0.3	1	100
markshare1	0	7200	2150514	0	7200	1349880	0	7200	461430	0
markshare2	0	7200	1633304	0	7200	1071762	0	7200	358188	0
mas74	0	7200	510347	75.8	7200	521762	78.3	7200	356091	75.1
mas76	0	4078	347759	100	3146.7	263187	100	5640.9	253624	100
misc01	6.5	3.5	207	100	2.8	180	100	3.2	163	100
misc03	6.4	7.3	279	100	17.4	707	100	10.5	369	100
misc05	23.7	8.2	207	100	4.8	141	100	2.3	75	100
misc07	0	634.4	9768	100	926.9	16161	100	528.2	8864	100
mkc	31.1	7200	3314	31.1	7200	201	60.3	7200	786	55.7
mod008	0	78.4	12498	100	80.3	12213	100	7.8	1131	100
mod011	0	7200	3701	65.3	7200	2424	83.2	7200	3272	72.8
mod013	35	0.8	140	100	1.2	167	100	0.6	72	100
modglob	18.9	7200	193834	59.1	7200	164859	77.4	7200	186402	76
net12	7.2	7200	64	7.8	7200	5	12.3	7200	8	8.3
nsrand-ipx	8.2	7200	7426	15.2	7200	6264	55.5	7200	6740	50.1
nw04	0	3584.8	474	100	6795.9	699	100	48.3	6	100
opt1217	0	7200	181560	0	7200	128001	25.4	7200	153013	43.3
p0033	12.6	0.7	349	100	0.1	22	100	0.1	40	100
p0201	45.6	4.5	126	100	3.8	131	100	3	82	100
p0282	5.4	7200	271116	12.5	3.6	140	100	6.1	186	100
p0548	4.5	7200	263866	86.1	7200	232534	99	7200	249489	97.4
p2756	0	7200	71574	0	7200	80020	2.4	7200	83052	2.4
pipex	46	2	422	100	1.6	273	100	1.4	219	100
pk1	0	3483.6	229371	100	4675.1	297376	100	5320.1	262141	100
pp08a	86.5	7200	161212	98.8	1068.4	18022	100	1297.8	24570	100
pp08aCUTS	49.7	7200	149337	93.6	4797.4	83338	100	7165	134868	100
qiu	0	3394.4	9786	100	4194.6	10741	100	2838.3	7185	100
qnet1	0	138.3	1219	100	309.1	1140	100	126.9	574	100
qnet1_o	20.6	35.4	214	100	204.5	826	100	234.1	828	100
rgn	26.3	9.2	1383	100	8.3	1367	100	6.7	1125	100
rout	0	7200	105270	53.5	2775.3	35932	100	655.3	10152	100
sample2	26.7	0.5	100	100	0.7	117	100	0.8	107	100
sentoy	0	1.2	188	100	1.4	211	100	1.2	180	100
set1al	96.2	0.4	19	100	0.5	19	100	0.6	19	100

continued on next page

Table 8: *continued*

Instance	Relative gap closed by cuts [%]	Simple disjunctions Solution time [s]	Simple disjunctions Nodes processed	Simple disjunctions Rel. gap closed [%]	MIG disjunctions Solution time [s]	MIG disjunctions Nodes processed	MIG disjunctions Rel. gap closed [%]	Combined Solution time [s]	Combined Nodes processed	Combined Rel. gap closed [%]
set1ch	85.3	7200	49392	87.7	7200	37705	91.7	7200	44872	91.2
set1cl	94.3	0.2	11	100	0.3	11	100	0.3	11	100
seymour	12.1	7200	589	35.8	7200	267	34.5	7200	417	35.7
sp97ar	234315 *	7200	1385	1397498 *	7200	808	3461368 *	7200	701	3344473 *
stein15	0	1.3	91	100	1.4	110	100	1.4	86	100
stein27	0	126.4	2771	100	180	3983	100	143.3	2889	100
stein45	0	6216.9	44577	100	7200	42340	93.8	4824.9	30572	100
swath	0 *	7200	22623	28.9 *	7200	16514	55.8 *	7200	15119	55.2 *
tr12-30	73.5	7200	41266	74.5	7200	34603	79.1	7200	33826	78.9
vpm1	39.7	7200	334508	96.3	1071.5	27509	100	1064.7	29306	100
vpm2	41.7	7200	197220	81.9	7200	121610	94	7200	144094	94.6

* Absolute gap closed reported.

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.
5. K. Andersen, G. Cornuéjols, and Y. Li. Reduce-and-split cuts: Improving the performance of mixed integer Gomory cuts. *to appear in Management Science*, 2005.
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. W. Cook, R. Kannan, and A Schrijver. Chvátal closures for mixed integer programs. *Mathematical Programming*, 47:155–174, 1990.
12. 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.
13. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 2003.
14. R. E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, Rand Corporation, 1960.
15. M. Grötschel, L. Lovász, and A. Schrijver. *Progress in combinatorial optimizarion*, chapter Geometric methods in combinatorial optimization, pages 167–183. Academic Press, Toronto, 1984.
16. H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
17. A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 1982.
18. J. Linderoth and M. Savelsbergh. A computational study of search strategies for mixed integer programming. Report LEC-97-12, Georgia Institute of Technology, 1997.
19. L. Lovász and H. E. Scarf. The generalized basis reduction algorithm. *Mathematics of Operations Research*, 17:751–764, 1992.
20. H. Marchand and L.A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49:363–371, 2001.
21. S. Mehrotra and Z. Li. On generalized branching methods for mixed integer programming. Technical report, Northwestern University, Evanston, Illinois 60208, 2004.
22. 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.
23. MIPLIB website of Zuse Institute Berlin.
    URL: http://miplib.zib.de/miplib3/miplib_prev.html.