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

Improved strategies for branching on general disjunctions

G. Cornuéjols · L. Liberti · G. Nannicini

July 21, 2008

Abstract Within the context of solving Mixed-Integer Linear Programs by a Branch-and-Cut algorithm, we propose a new strategy for branching. Computational experiments show that, on the majority of our test instances, this approach enumerates fewer nodes than traditional branching. On average, the number of nodes in the enumeration tree is reduced by a factor two, while computing time is comparable. On a few instances, the improvements are of several orders of magnitude in both number of nodes and computing time.

Keywords integer programming · branch and bound · split disjunctions

1 Introduction

Mixed Integer Linear Programs (MILPs) arise in several real-life applications, and are usually solved via a Branch-and-Cut algorithm such as that implemented by Cplex [7], where the node bound is obtained by solving a Linear Programming (LP) relaxation of the MILP. Usually, branching occurs on the domain of integer variables; however, this need not be so: any disjunction of the feasible region of the relaxed LP not excluding points that are feasible in the original MILP can be used for branching. We use the term *branching on general disjunctions* to mean a branching strategy where the disjunctions are two disjoint halfspaces of the form $\pi x \leq \beta_0, \pi x \geq \beta_1$ with $\beta_0 < \beta_1$. Branching on general disjunction is considered impractical because of the large computational effort needed to find a suitable general disjunction. A recent paper [8] proposes branching on general disjunctions arising from Mixed-Integer Gomory Cuts (MIGC). At each node there is a choice of possible MIGCs from which to derive the branching disjunction. The branching strategy suggested in [8] is based on the distance cut off by the corresponding intersection cut as a quality measure for the choice of disjunction. The improvement in objective function value that occurs

G. Cornuéjols
LIF, Faculté de Sciences de Luminy, Marseille, France and
Tepper School of Businness, Carnegie Mellon University, Pittsburgh, PA, USA
E-mail: `gc0v@andrew.cmu.edu`

L. Liberti, G. Nannicini
LIX, École Polytechnique, 91128 Palaiseau, France
E-mail: `{liberti,giacomon}@lix.polytechnique.fr`

after branching on a split disjunction is at least as large as the improvement obtained after adding the corresponding intersection cut. In this paper, we propose a modification in the class of disjunctions used for branching; instead of simply computing the disjunctions that define MIGC at the optimal basis, we try to generate a new set of disjunctions in order to increase the distance cut off by the corresponding intersection cut. Moreover, we combine branching on simple disjunctions and on general disjunctions in an effective branching algorithm, which shows an improvement over traditional branching rules on the majority of test instances.

2 Preliminaries and notation

In this paper we consider the Mixed Integer Linear Program in standard form:

$$\left.\begin{array}{rl} \min c^\top x & \\ Ax = b & \\ x \geq 0 & \\ \forall j \in N_I \quad x_j \in \mathbb{Z}, & \end{array}\right\} \mathscr{P} \tag{1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$ and $N_I \subset N = \{1,\dots,n\}$. The LP relaxation of (1) is the linear program obtained by dropping the integrality constraints, and is denoted by $\overline{\mathscr{P}}$. The Branch-and-Bound algorithm makes an implicit use of the concept of disjunctions [3]: whenever the solution of the current relaxation is fractional, we divide the current problem $\mathscr{P}$ into two subproblems $\mathscr{P}_1$ and $\mathscr{P}_2$ such that the union of the feasible regions of $\mathscr{P}_1$ and $\mathscr{P}_2$ contains all feasible solutions to $\mathscr{P}$. Usually, this is done by choosing a fractional component $\overline{x_i}$ (for some $i \in N_I$) of the optimal solution $\overline{x}$ to the relaxation $\overline{\mathscr{P}}$, and adding the constraints $x_i \leq \lfloor \overline{x_i} \rfloor$ and $x_i \geq \lceil \overline{x_i} \rceil$ to $\mathscr{P}_1$ and $\mathscr{P}_2$ respectively.

Within this paper, we take the more general approach whereby branching can occur with respect to a direction $\pi \in \mathbb{R}^n$ by adding the constraints $\pi x \leq \beta_0$, $\pi x \geq \beta_1$ with $\beta_0 < \beta_1$ to $\mathscr{P}_1$ and $\mathscr{P}_2$ respectively, as long as no integer feasible point is cut off. Owen and Mehrotra [9] generated branching directions π where $\pi_j \in \{-1, 0, +1\}\ \forall j \in N_I$ and showed that using such branching directions can decrease the size of the enumeration tree significantly. Aardal et al. [1] used basis reduction to find good branching directions for certain classes of difficult integer programs. Karamanov and Cornuéjols [8] proposed using the disjunctions defining MIGCs at the current basis. Given $B \subset N$ an optimal basis of $\overline{\mathscr{P}}$, and $J = N \setminus B$, i.e. J is the set of nonbasic variables, the corresponding simplex tableau is given by

$$x_i = \overline{x_i} - \sum_{j\in J} \overline{a}_{ij} x_j \quad \forall i \in B. \tag{2}$$

For $j \in J$, let $r^j \in \mathbb{R}^n$ be defined as

$$r^j_i = \begin{cases} -\overline{a}_{ij} & \text{if } i \in B \\ 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

These vectors are the extreme rays of the cone $\{x \in \mathbb{R}^n \mid Ax = b \wedge \forall j \in J\ (x_j \geq 0)\}$ with apex $\overline{x}$. Let $D(\pi,\pi_0)$ define the split disjunction $\pi^T x \leq \pi_0 \vee \pi^T x \geq \pi_0 + 1$, where $\pi \in \mathbb{Z}^n, \pi_0 \in \mathbb{Z}, \pi_j = 0$ for $i \notin N_I, \pi_0 = \lfloor \pi^T \overline{x} \rfloor$. By integrality of (π,π_0), any feasible solution of $\mathscr{P}$ satisfies every split disjunction. Let $\varepsilon(\pi,\pi_0) = \pi^T \overline{x} - \pi_0$ be the violation by $\overline{x}$ of the first term of

$D(\pi,\pi_0)$. Assume that the disjunction $D(\pi,\pi_0)$ is violated by $\bar{x}$, i.e. $0<\varepsilon(\pi,\pi_0)<1$. The intersection cut associated with a basis B and a split disjunction $D(\pi,\pi_0)$ is

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

where $\forall j\in J$ we define

$$\alpha_j(\pi,\pi_0)=\begin{cases}-\frac{\varepsilon(\pi,\pi_0)}{\pi^T r^j} & \text{if } \pi^T r^j<0\\ \frac{1-\varepsilon(\pi,\pi_0)}{\pi^T r^j} & \text{if } \pi^T r^j>0\\ +\infty & \text{otherwise.}\end{cases} \tag{5}$$

The Euclidean distance between $\bar{x}$ and the cut defined above is (see [4]):

$$\delta(B,\pi,\pi_0)=\sqrt{\frac{1}{\sum_{j\in J}\frac{1}{\alpha_j(\pi,\pi_0)^2}}}. \tag{6}$$

3 Branching on general disjunctions: a quadratic optimization approach

We would like to generate good branching disjunctions $D(\pi,\pi_0)$. In [8] it is shown that the gap closed by branching on a disjunction $D(\pi,\pi_0)$ is at least as large as the improvement in the objective function obtained by the corresponding intersection cut. Thus, it makes sense to attempt to increase the value of the distance $\delta(B,\pi,\pi_0)$ as much as possible. It is easy to see that this means increasing the value of $\alpha_j(\pi,\pi_0)\ \forall j\in J$, which in turn corresponds to decreasing the coefficient of the intersection cut (4). [8] considered intersection cuts (4) obtained directly from the optimal tableau (2) as MIGCs for $i\in B\cap N_I$ such that $\bar{x}_i\notin\mathbb{Z}$. The split disjunction $D(\pi^i,\pi^i_0)$ that defines the MIGC associated to a row $\bar{a}_i$ where x_i is basic can be computed as:

$$\pi^i_j=\begin{cases}\lfloor\bar{a}_{ij}\rfloor & \text{if } j\in N_I\cap J \text{ and } \bar{a}_{ij}-\lfloor\bar{a}_{ij}\rfloor\leq\bar{x}_i-\lfloor\bar{x}_i\rfloor\\ \lceil\bar{a}_{ij}\rceil & \text{if } j\in N_I\cap J \text{ and } \bar{a}_{ij}-\lfloor\bar{a}_{ij}\rfloor>\bar{x}_i-\lfloor\bar{x}_i\rfloor\\ 1 & \text{if } i=j\\ 0 & \text{otherwise,}\end{cases} \tag{7}$$

$$\pi^i_0=\lfloor(\pi^i)^T\bar{x}\rfloor.$$

The corresponding $\alpha_j(\pi,\pi_0)$ is

$$\alpha_j(\pi,\pi_0)=\begin{cases}\max\left(\frac{\bar{x}_i-\lfloor\bar{x}_i\rfloor}{\bar{a}_{ij}-\lfloor\bar{a}_{ij}\rfloor},\frac{\lceil\bar{x}_i\rceil-\bar{x}_i}{\lceil\bar{a}_{ij}\rceil-\bar{a}_{ij}}\right) & \text{if } j\in J\cap N_I\\ \max\left(\frac{\bar{x}_i-\lfloor\bar{x}_i\rfloor}{\bar{a}_{ij}},\frac{\lceil\bar{x}_i\rceil-\bar{x}_i}{-\bar{a}_{ij}}\right) & \text{if } j\in J\setminus N_I.\end{cases} \tag{8}$$

By convention, $\alpha_j(\pi,\pi_0)$ is equal to $+\infty$ when one of the denominators is zero in (8).

In this paper we study a method for increasing $\alpha_j(\pi,\pi_0)$. By (6), this yields a disjunction with a larger value of $\delta(B,\pi,\pi_0)$, which is thus likely to close a larger gap. To achieve this goal, we modify the underlying disjunction $D(\pi,\pi_0)$, which has an influence on both the numerators and the denominators of (8). It seems difficult to optimize $\alpha_j(\pi,\pi_0)$ for $j\in J\cap N_I$ because both terms of the fraction are nonlinear. Therefore, we concentrate on $j\in J\setminus N_I$.

Let $B_I=B\cap N_I$, $J_C=J\setminus N_I$; apply a permutation to the simplex tableau in order to obtain $B_I=\{1,\dots,|B_I|\}, J_C=\{1,\dots,|J_C|\}$, and define the matrix $D\in\mathbb{R}^{|B_I|\times|J_C|}$, $d_{ij}=\bar{a}_{ij}$.

Andersen, Cornuéjols and Li [2] proposed a reduction algorithm which cycles through the rows d_i of D and, for each one, considers whether summing an integer multiple of some other row of D yields a reduction of $\|d_i\|$. The optimal integer multiplier can be computed in closed form (see [2]). The denominators of $\alpha_j(\pi,\pi_0)\,\forall j\in J\setminus N_I$ for the intersection cut associated with a row of the simplex tableau are exactly the elements of the corresponding row of D, thus a reduction of $\|d_i\|$ should yield an increase in the α_j's. However, this method only takes into account two rows at a time when computing the optimal integer coefficient.

Our idea is to use, for each row d_k of D, a subset $R_k\subset B_I$ of the rows of the simplex tableau with $d_k\in R_k$, in order to reduce $\|d_k\|$ as much as possible with a linear combination with integer coefficients of d_k and $d_j\ \forall j\in R_k\setminus\{k\}$. This is done by defining, for each row d_k that we want to reduce, the convex minimization problem:

$$\min_{\lambda^k\in\mathbb{R}^{|R_k|},\lambda^k_k=1}\|\sum_{j\in R_k}\lambda^k_j d_j\|, \tag{9}$$

and then rounding the coefficients λ^k_j to the nearest integer $\left\lfloor\lambda^k_j\right\rceil$. The purpose of imposing $\lambda^k_k=1$ is to get different optimization problems for $k=1,\dots,|B_I|$, thus increasing the chance of obtaining different branching directions. With basic algebraic calculations it can be shown that this problem can be solved via an $|R_k|\times|R_k|$ linear system. We formalize our problem: for $k=1,\dots,|B_I|$ we solve the linear system

$$A^k\lambda^k=b^k,$$

where $A^k\in\mathbb{R}^{|R_k|\times|R_k|}$ and $b^k\in\mathbb{R}^{|R_k|}$ are defined as follows:

$$\begin{aligned}A^k_{ij}&=\begin{cases}1 & \text{if } i=j=k\\ 0 & \text{if } i=k \text{ or } j=k \text{ but not both}\\ \sum_{h=1}^{|J^C|}d_{ih}d_{jh} & \text{otherwise,}\end{cases}\\ b^k_i&=\begin{cases}1 & \text{if } i=k\\ -\sum_{h=1}^{|J^C|}d_{ih}d_{kh} & \text{otherwise.}\end{cases}\end{aligned} \tag{10}$$

The form of the linear system guarantees $\lambda^k_k=1$ in the solution.

Once these linear systems are solved and we have the optimal coefficients $\lambda^k\in\mathbb{R}^{|R_k|}\ \forall k\in\{1,\dots,|B_I|\}$, we round them to the nearest integer, and consider the norm of $\sum_{j\in R_k}\left\lfloor\lambda^k_j\right\rceil d_j$. If $\|\sum_{j\in R_k}\left\lfloor\lambda^k_j\right\rceil d_j\|<\|d_k\|$, then we use the row $\sum_{j\in R_k}\left\lfloor\lambda^k_j\right\rceil\bar{a}_j$ instead of row $\bar{a}_k$ of the simplex tableau in order to compute the split disjunction that defines the associated MIGC, and consider the possibly improved disjunction for branching.

Example 1 Suppose we have the following matrix D:

$$D=\begin{bmatrix}3&1&8&2&3&2&3\\1&-2&0&12&-2&-4&-5\\0&-1&4&1&4&5&-1\\1&1&1&1&0&0&2\end{bmatrix}$$

and we apply the reduction algorithm to the first row d_1. Following (10), we obtain the linear system:

$$\begin{bmatrix}1&0&0&0\\0&194&-9&1\\0&-9&60&2\\0&1&2&8\end{bmatrix}\begin{bmatrix}\lambda^1_1\\\lambda^1_2\\\lambda^1_3\\\lambda^1_4\end{bmatrix}\begin{bmatrix}1\\4\\-52\\-20\end{bmatrix},$$

whose solution is $\lambda^1 = (\lambda_1^1, \lambda_2^1, \lambda_3^1, \lambda_4^1)^\top = (1, -0.0042, -0.7906, -2.3018)^\top$. Rounding each component to the nearest integer and computing $\lfloor \lambda^1 \rceil^\top D$ we obtain the row:

$$\begin{bmatrix} 1 \ 0 \ 2 \ -1 \ -1 \ -3 \ 0 \end{bmatrix},$$

whose L_2 norm is 4, as opposed to the initial norm of d_1, which is 10. Thus, we compute the corresponding row of the simplex tableau with the same coefficients λ^1, and consider the possibly improved disjunction for branching.

On the other hand, if we apply the reduction algorithm to the second row of D, we obtain the linear system:

$$\begin{bmatrix} 100 & 0 & 52 & 20 \\ 0 & 1 & 0 & 1 \\ 52 & 0 & 60 & 2 \\ 20 & 0 & 2 & 8 \end{bmatrix} \begin{bmatrix} \lambda_1^2 \\ \lambda_2^2 \\ \lambda_3^2 \\ \lambda_4^2 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 9 \\ -1 \end{bmatrix},$$

whose solution is $\lambda^2 = (\lambda_1^2, \lambda_2^2, \lambda_3^2, \lambda_4^2)^\top = (-0.0627, 1, 0.2050, -0.0196)^\top$. Rounding each component to the nearest integer and computing $\lfloor \lambda^2 \rceil^\top D$ gives the original row d_2, hence the reduction algorithm did not yield an improvement.

The choice of each set $R_k \subset B_I \ \forall k$ has an effect on the performance of the norm reduction algorithm. Although using $R_k = B_I$ is possible, in that case two problems arise: first, the size of the linear systems may become unmanageable in practice, and second, if we add up too many rows then the coefficients on the variables with indices $\in J \cap N_I$ may deteriorate. In particular, we may get more nonzero coefficients. Thus, we do the following. We fix a maximum cardinality $M_{|R_k|}$; if $M_{|R_k|} \geq |B_I|$, we set $R_k = B_I$. Otherwise, for each row k that we want to reduce, we sort the remaining rows by ascending number of nonzero coefficients on the variables with indices in $\{i \in J \cap N_I | \bar{a}_{ki} = 0\}$, and select the first $M_{|R_k|}$ indices as those in R_k. The reason for this choice is that, although the value of the coefficients on the variables with indices $\in J \cap N_I$ is bounded, we would like those that are zero in row $\bar{a}_k$ to be left unmodified when we compute $\sum_{j \in R_k} \lfloor \lambda_j^k \rceil \bar{a}_j$. As zero coefficients on those variables yield a stronger cut, we argue that this will yield a stronger split disjunction as well.

4 Computational experiments

To assess the usefulness of our approach, we implemented within the Cplex 11.0 Branch-and-Bound framework the following branching methods:

- branching on single variables (Simple Disjunctions, SD),
- branching on split disjunctions after the reduction step that we proposed in Section 3 (Improved General Disjunctions, IGD),
- branching on the split disjunctions that define the MIGCs at the current basis (General Disjunctions, GD), in order to compare with the work of Karamanov and Cornuéjols [8],
- branching on split disjunctions after the application of the Reduce-and-Split reduction algorithm (Reduce-and-Split, RS), in order to compare with the work of Andersen, Cornuéjols and Li [2],
- a combination of the SD and IGD method (Combined General Disjunction, CGD), which is described in Section 4.2.

In each set of experiments we applied only the methods that were meaningful for that particular experiment. We applied strong branching in order to choose the best branching decision. When not otherwise stated, the best branching decision is considered to be the one that generates the smallest number of feasible children, or, in the case of a tie, the one that closes more gap, computed as $\min\{c^\top \bar{x}^1, c^\top \bar{x}^2\}$ where $\bar{x}^1, \bar{x}^2$ are the optimal solutions of the LP relaxations of the children nodes. If a branching decision generates only one feasible child at the current node, one side of the disjunction (i.e. the feasible one) can be considered as a cutting plane; when several disjunctions of this kind are discovered, we add all these cutting planes. This leads to only one feasible child, but with possibly a larger closed gap with respect to the case where we add only one disjunction as branching constraint. Unless otherwise stated, our testbed is the union of `miplib2.0`, `miplib3` and `miplib2003`, after the removal of all instances that can be solved to optimality in less than 10 nodes by the SD algorithm[1], and the instances where one node cannot be processed in less than 30 minutes by the SD algorithm[2]. In total, the set consists in 96 heterogeneous instances. The node selection strategy was set to *best bound*, and the value of the optimal solution was given as a cutoff value for all those instances where the optimum is known[3]. These choices were meant to reduce as much as possible the size of the enumeration tree, and to minimize the effect of heuristics and of other uncontrollable factors (such as the time to find the first integer solution) in order to get a more stable indication of the algorithm performance on branching.

The rest of this section is divided as follows. In Section 4.1 we consider the different branching algorithms separately, and compare them in several respects. In Section 4.2 we capitalize on the insight gained with the different experiments of the previous section, and we combine the methods into a single branching algorithm, which we test in a Branch-and-Cut framework. All averages reported in the following are geometric averages.

4.1 Comparison of the different methods

The first set of experiments involves branching at root node in order to evaluate the amount of integrality gap closed; we compute the relative closed integrality gap as $\frac{\text{closed gap}}{\text{initial gap}}$ for those instances where the optimal solution is known, while for other instances we simply compare the absolute closed gap. In this set of experiments we evaluated all possible branching decisions via strong branching: that is, for SD we branched on all fractional integer variables, for GD we branched on the split disjunctions computed from the rows of the simplex tableau where the associated basic variable is a fractional integer variable, and for IGD we branched on all the split disjunctions obtained after the reduction step described in Section 3 applied to all rows of the simplex tableau where the associated basic variable is a fractional integer variable. For IGD, the maximum number of rows considered in each reduction step was set to 50 (i.e. $M_{|R_k|} = 50\,\forall k$). The experiments were made in a bare Branch-and-Bound setting; that is, presolving, cutting planes and heuristics were disabled. In these experiments, we chose the branching decision that closed the largest gap, regardless of the number of feasible children. In Table 1 we give a summary of the results for this experiment. We report the average relative integrality gap closed by each method, the number of instances where each method closes at least the same absolute gap as the other two methods, and the number of instances where the disjunction that closes the largest gap generates only one feasible child.

[1] The instances are: `air01, air02, air03, air06, misc04, stein09`.

[2] The instances are: `atlanta-ip, ds, momentum1, momentum2, momentum3, msc98-ip, mzzv11, mzzv42z, net12, rd-rplusc-21, stp3d`.

[3] See the `miplib2003` web site: `http://miplib.zib.de/miplib2003.php`.

Average closed gap (on instances with known optimum)	
Simple disjunctions (SD):	4.27%
General disjunctions (GD):	6.71%
Improved general disjunctions (IGD):	6.56%

Number of instances with largest closed gap	
Simple disjunctions (SD):	58
General disjunctions (GD):	64
Improved general disjunctions (IGD):	70

Number of instances with one child	
Simple disjunctions (SD):	10
General disjunctions (GD):	29
Improved general disjunctions (IGD):	27

Table 1 Results after branching at root node

For the first criterion we only considered instances where the optimal solution is known, so that we could compute the relative amount of integrality gap close; for the remaining criteria, we also considered the instances with unknown optimum. We immediately observe that branching on general disjunctions instead of single variables yields a significantly larger amount of closed integrality gap. In our experiments, the GD method closes more gap on average than the other two methods, and both GD and IGD clearly outperform SD under this criterion. Not only GD and IGD close more gap, but they also more frequently generate only one feasible child node; the number of children was not taken into account when choosing the branching decision in this set of experiments, but it is interesting to note that with GD and IGD we often have disjunctions that close a large amount of gap and also do not increase the size of the enumeration tree. Although the GD method closes slightly more gap on average than IGD on the instances with known optimum, if we compare the number of instances where each method closes at least the same amount of gap as the other two methods then IGD ranks first with 70 instances over 96 in total.

For many reasons, applying strong branching on all possible branching decisions is impractical, as it requires a very large computational effort which is not counterbalanced by the reduction of the size of the enumeration tree. In the remaining experiments we evaluated the performance of the branching algorithms in a framework where strong branching is applied only to the most promising branching decisions. The number of branching decisions evaluated with strong branching was set to 10. In the case of SD, we picked the 10 integer variables with the largest fractionary part (i.e. closer to 0.5). For GD and IGD, we picked the 10 split disjunctions associated with the 10 MIGCs that have the largest cut off distance (equation (6), see [8]), where for IGD the distance was computed after the reduction step.

In the next two experiments, we solved up to 1000 nodes in the enumeration tree. We reverted back to the original branching decision selection method that favours those disjunctions which generate a smaller number of feasible children. Having a smaller number of children is a considerable advantage as we are able to progress further in depth of the enumeration tree, thus possibly leading to a larger closed gap. The evaluation criterion was the percentage of the integrality gap closed after 1000 nodes, or, if the instance was solved in less than 1000 nodes, the number of nodes required to solve to optimality. For this set of experiments, 7 instances[4] were excluded from the testbed, as solving 1000 nodes required more than 12 hours. To choose the number of rows $M_{|R_k|}$ that defines the size of the linear system at each iteration of the reduction step for the IGD method, we compared three different values: 10, 20 and 50; we included in the comparison the Reduce-and-Split (RS) coefficient improvement method introduced by Andersen, Cornuéjols and Li [2], in order to test whether their algorithm to generate good cutting planes was also suitable for branching. For fairness, we used for RS the same procedure as for the IGD methods: we picked the 10 split disjunctions associated with the 10 MIGCs that have the largest cut off distance after the reduction step, and applied strong branching. We followed the implementation of the RS reduction algorithm given in the CGL library [6]. A summary of the results is given in Table 2. It can be seen that IGD using 20 or 50 rows for the reduction step yields very

Number of solved instances	
RS:	37
IGD with $M_{|R_k|} = 10$ (IGD10):	43
IGD with $M_{|R_k|} = 20$ (IGD20):	42
IGD with $M_{|R_k|} = 50$ (IGD50):	42

Average number of nodes on instances solved by all methods	
RS:	57.8
IGD with $M_{|R_k|} = 10$ (IGD10):	41.8
IGD with $M_{|R_k|} = 20$ (IGD20):	44.0
IGD with $M_{|R_k|} = 50$ (IGD50):	39.7

Average gap closed on instances not solved by any method	
RS:	12.80%
IGD with $M_{|R_k|} = 10$ (IGD10):	14.12%
IGD with $M_{|R_k|} = 20$ (IGD20):	15.06%
IGD with $M_{|R_k|} = 50$ (IGD50):	14.85%

Number of instances with largest closed gap	
RS:	53
IGD with $M_{|R_k|} = 10$ (IGD10):	63
IGD with $M_{|R_k|} = 20$ (IGD20):	62
IGD with $M_{|R_k|} = 50$ (IGD50):	66

Table 2 Results after 1000 solved nodes

similar results in terms of average closed gap on instances not solved by any method, and both choices close more gap than IGD with $M_{|R_k|} = 10$ or RS on the unsolved instances. The average number of nodes is smaller for $M_{|R_k|} = 50$. In particular IGD outperforms RS for branching. We give the following possible reason. One of the advantages of RS for cut generation is that the reduction algorithm generates several split disjunctions, trying to increase the distance cut off by each one of the associated cutting planes. As several cuts are generated at each round, this approach is effective [2]. However, at each node of a Branch-and-Bound tree only one disjunction is chosen for branching, so a method which tries to

[4] The instances are: `dano3mip`, `fast0507`, `manna81`, `mitre`, `protfold`, `sp97ar`, `t1717`.

compute only one strong disjunction is more fruitful than one that generates a set of several possibly weaker ones. This may explain why the reduction algorithm described in Section 3 seems to be more effective than RS for branching. We decided to use IGD with $M_{|R_k|} = 50$ in all following experiments. We did not test larger values of the parameter, since solving the linear system would take too much time in practice.

A summary of the comparison between SD, GD and IGD with $M_{|R_k|} = 50$ can be found in Table 3. The increase in the gap per node that can be closed by branching on general

Number of solved instances	
Simple disjunctions (SD):	35
General disjunctions (GD):	42
Improved general disjunctions (IGD):	42

Average number of nodes on instances solved by all methods	
Simple disjunctions (SD):	92.7
General disjunctions (GD):	52.9
Improved general disjunctions (IGD):	43.2

Average gap closed on instances not solved by any method	
Simple disjunctions (SD):	9.36%
General disjunctions (GD):	13.78%
Improved general disjunctions (IGD):	14.15%

Number of instances with largest closed gap	
Simple disjunctions (SD):	55
General disjunctions (GD):	56
Improved general disjunctions (IGD):	63

Table 3 Results after 1000 solved nodes

disjunctions with respect to branching on single variables is large. This is confirmed by the results in [8]. Besides, the IGD method seems to be on average superior in all respects to the other two methods, as it closes more gap for the unsolved instances under 1000 nodes, and requires less nodes for the solved instances. This is also evident if we compare the number of instances where each method closes at least the same absolute gap as the other two methods: IGD ranks first with 63 instances over the 89 instances in the testset. However, there are two instances where branching on simple disjunctions is more profitable than branching on general disjunctions: the `air04` and `air05` instances are solved by the SD method in 225 and 139 nodes respectively, while GD and IGD do not manage to solve them in 1000 nodes. All other instances which are solved by SD are also solved by GD and IGD.

4.2 Combination of several methods

Results in Table 3 suggest that IGD is indeed capable of closing more gap per node on a large number of instances; however, a more detailed analysis of the results shows that there are a few instances where branching on general disjunctions is not profitable, and thus both GD and IGD perform poorly. This may also happen, for example, in zero gap instances,

Algorithm 1 CGD branching algorithm

```
Initialization: $ActiveGDCounter \leftarrow 3, FailedActivation \leftarrow 0, NodeCounter \leftarrow 0$
while branching do
    if root node then
        $NumGD \leftarrow 20, NumSD \leftarrow 20$
    else
        if $ActiveGDCounter > 0$ then
            $NumGD \leftarrow 7, NumSD \leftarrow 3$
        else
            $NumGD \leftarrow 0, NumSD \leftarrow 10$
        end if
    end if
    generate $NumGD$ general disjunctions
    generate $NumSD$ simple disjunctions
    for all branching decisions do
        apply strong branching
    end for
    choose a disjunction $D(\pi, \pi_0)$
    if $ActiveGDCounter > 0$ then
        if $D(\pi, \pi_0)$ has support $> 1$ then
            $ActiveGDCounter \leftarrow 10$
            $FailedActivation \leftarrow 0$
        else
            $ActiveGDCounter \leftarrow ActiveGDCounter - 1$
            if $ActiveGDCounter = 0$ then
                $FailedActivation \leftarrow FailedActivation + 1$
            end if
        end if
    else
        $NodeCounter \leftarrow NodeCounter + 1$
    end if
    if $FailedActivation < 10 \wedge NodeCounter = 100$ then
        $ActiveGDCounter \leftarrow 1$
        $NodeCounter \leftarrow 0$
    end if
end while
```

where the enumeration of nodes with SD is usually more effective. Thus, we decided to combine both the SD and the IGD method into a single branching algorithm which tries to decide, for each instance, if it is more effective to branch on simple disjunctions or on general disjunctions. First we describe the ideas and the practical considerations behind the algorithm, and then we will describe how we implemented it.

The most evident drawback of branching on general disjunctions is that it is slower than using simple disjunctions. It is slower in several respects: the first reason is that the computations at each node take longer. This is because we have to compute the distance cut off by the MIGC associated with each row of the simplex tableau, and the reduction step that we propose involves the solution of an $M_{|R_k|} \times M_{|R_k|}$ linear system for each row which is improved, where we chose $M_{|R_k|} = 50$. All these computations are carried out several times, thus the overhead per node with respect to branching on simple disjunctions is significant. The second reason is that, by branching on general disjunction, we add one (or more) rows to the formulation of children nodes, which may result in a slowdown of the LP solution process. On the other hand, branching on simple disjunctions involves only a change in the bounds of some variables, thus the size of the LP does not increase. This suggests that branching on general disjunctions should be used only if it is truly profitable, which in turn requires a measure of profit. We decided to use the amount of closed gap as a measure of profit. Besides, since the computational overhead per node is significant when considering general disjunctions for branching, we would like to consider them only if it brings an improvement in the solution time. Thus, if on a given instance general disjunctions are never used because simple disjunctions are more profitable, we would like to disable their computation as soon as possible in the enumeration tree. As the polyhedron underlying a problem may significantly change in different parts of the branching tree, it may be a good idea to test branching on general disjunctions periodically even if it has been disabled, in order to verify whether it has become profitable.

We implemented a branching algorithm based on the above considerations in the following way: at each node, branching on general disjunctions can be active or not. If it is active, we test 10 possible branching decisions with strong branching: 7 general disjunctions, and 3 simple disjunctions. General disjunctions are picked only if they generate a smaller amount of children nodes, or (in case of a tie) if the amount of closed gap is at least 50% larger. As a consequence, at all nodes where we do not manage to close any gap we always prefer simple disjunctions if they generate the same number of children as general disjunctions. At the beginning of the enumeration tree, branching on general disjunctions is active for the first 3 nodes; moreover, we put an increased effort at root node, where we consider up to 20 simple disjunctions and 20 general disjunctions. Whenever a general disjunction is chosen for branching, then branching on general disjunctions is activated for the following 10 nodes. Otherwise, when it is disactivated (because of simple disjunctions being preferred to general disjunctions for a given number of consecutive nodes, i.e. 3 at the beginning of the enumeration tree, 10 otherwise), it is reactivated again after 100 nodes, but only for one node, in order to test whether in that part of the enumeration tree general disjunctions are worthwhile. If a general disjunction is chosen, then branching on general disjunctions is reactivated for the following 10 nodes. After 10 consecutive unfruitful activations, i.e. general disjunctions are not chosen after being activated for 10 consecutive times, branching on general disjunctions is permanently disabled. When performing the reduction step described in Section 3, in order to save time we do not consider all rows for reduction, but only the most promising ones. We do this by looking at the MIGC associated with each row where the basic integer variable is fractional, and sorting them by the corresponding distance cut off (6). The 10 rows (20 at root node) with the largest distance are modified with the reduction step of Section 3. Since only 7 have to be selected for strong branching, we recompute the distances and pick the 7 largest ones. We give a description of this algorithm in Algorithm 1.

To assess the practical usefulness of this approach, we compared this branching algorithm, which we will call Combined General Disjunctions (CGD), with SD. In order to evaluate the same number of branching decisions via strong branching with both methods at each node, we modified SD in order to consider, at root node, the branching decisions corresponding to the 40 integer variables with largest fractional part, and then reverting back to the usual 10 for the following nodes. We let Cplex 11.0 apply cutting planes at root node with the default parameters, and then branched for two hours. Again, preprocessing and heuristics were disabled. In Table 4 we compare the number of solved instances within the two hours limit, the average closed gap on instances not solved by either method, the average number of nodes and average CPU time on instances solved by both methods. The results clearly indicate that the CGD is able to combine the potential of the IGD method to close more gap with the rapidity of branching on simple disjunctions when general disjunctions are not worth the additional required time. Not only CGD solves all instances solved by SD, but it solves 3 more: `10teams` in 273.46 seconds of CPU time, `gesa2_o` in 2616.2 seconds, and `rout` in 2540.74 seconds. On the instances which have not been solved by either of the

Number of solved instances	
Simple disjunctions (SD):	67
Combined general disjunctions (CGD):	70

Average number of nodes on instances solved by both methods	
Simple disjunctions (SD):	195.1
Combined general disjunctions (CGD):	98.0

Average number of nodes on instances not solved by either method	
Simple disjunctions (SD):	35796.0
Combined general disjunctions (CGD):	15075.7

Average gap closed on instances not solved by either method	
Simple disjunctions (SD):	5.35%
Combined general disjunctions (CGD):	7.03%

Average CPU time [sec] on instances solved by both methods	
Simple disjunctions (SD):	3.03
Combined general disjunctions (CGD):	3.35

Table 4 Results in a Branch-and-Cut framework with a two hours time limit

two methods, the average integrality gap closed by CGD is 31% larger in relative terms than the one closed by SD. This result is even more important if we consider that CGD is slower: in the 2 hours limit CGD solved only half as many nodes as SD on average, thus the gap closed per node is significantly larger for CGD. These average values only take into account the instances with known optimum value.

We report a full table of results on the instance that have not been solved in less than two hours by the SD method in Table 5. If we consider the 5 instances for which the optimal solution value is not known, then on the `liu` instance both methods close the same absolute gap, on `dano3mip` CGD closes more gap, and on the remaining 3 instances (`sp97ar, t1717, timtab2`) SD closes more gap. However, on all 5 instances CGD solves a smaller amount of nodes since it is slower, and the relative difference (i.e. $\frac{\text{absolute gap SD}}{\text{absolute gap CGD}} - 1$) in closed gap on the 3 instances where SD closes more gap is small: on `timtab2`, the difference is only 0.13%, but CGD requires 4 times fewer nodes; on `sp97ar` the difference is 4.95% in favour of SD, but CGD requires 13 times fewer nodes. The difference increases on the `t1717` instance: SD closes in relative terms 12.99% more gap than CGD, solving twice as many nodes in the two hours limit.

On a few instances, CGD performs strikingly better than SD. Examples are the `arki001` and `opt1217` instances, which are difficult instances of `miplib2003`. For `arki001`, branching with CGD closes 45% of the gap, whereas branching with SD only closes 6.83%. Similarly, for `opt1217` CGD closes 33.2%, versus 0% for SD. The `arki001` instance was first solved to optimality only recently by Balas and Saxena [5]: they invest a large computational effort in order to generate rank-1 split cuts that close 83.05% of the integrality gap, and then use Cplex's Branch-and-Bound algorithm to close the remaining gap (16.95%) in 643425 nodes. We report that, if we run CGD on `arki001` without time limits, 28.27% of the inte-

	SD algorithm			CGD algorithm			Gap
	Closed Gap			Closed Gap			Closed
Instance	Abs.	Rel.	Nodes	Abs.	Rel.	Nodes	By Cuts
10teams*	0	0%	11775	2	28.5%	398	71.3%
a1c1s1	337.58	3.21%	5340	371.423	3.54%	2578	62.29%
aflow40b	36.854	22.7%	20398	25.8243	15.9%	5477	57.3%
arki001	88.0556	6.83%	3612	580.27	45%	4000	28.27%
dano3mip	0.322586	-	8	0.374207	-	6	0%
danoint	0.310476	10.2%	5547	0.286139	9.44%	4790	2%
fast0507	0.262111	14.1%	587	0.0561795	3.03%	96	0%
gesa2_o*	84644.7	27.9%	195797	147352	48.5%	13181	51.4%
glass4	3293.85	0%	84369	3104.73	0%	79050	0%
harp2	199205	43.9%	74255	215937	47.5%	12565	32.6%
liu	214	-	108162	214	-	100347	0%
markshare1	0	0%	11027872	0	0%	2540405	0%
markshare2	0	0%	8606987	0	0%	2431791	0%
mas74	859.296	65.2%	2405902	641.509	48.7%	800207	4.6%
mkc	2.92749	6.1%	14486	6.52824	13.6%	8663	5.7%
noswot	0	0%	3192040	0	0%	1598812	0%
nsrand-ipx	158.293	6.82%	3932	222.726	9.6%	1431	49.08%
opt1217	0	0%	409010	1.33599	33.2%	316821	17%
protfold	2.32009	21.2%	140	2.14092	19.5%	150	3.6%
roll3000	127.615	7.12%	3083	293.192	16.4%	1406	40.68%
rout*	55.1337	57.6%	189312	94.9211	99.2%	28137	0.8%
set1ch	977.236	4.34%	120033	1355.82	6.02%	41034	86.06%
seymour	1.44368	7.54%	1251	1.09335	5.71%	688	41.66%
sp97ar	1.48955e+06	-	4231	1.41919e+06	-	318	0%
swath	28.3223	21.3%	20831	15.7973	11.9%	4724	34.9%
t1717	785.581	-	76	695.249	-	31	0%
timtab1	108754	14.8%	130014	103832	14.1%	35760	62.2%
timtab2	531157	-	50595	530454	-	12461	0%
tr12-30	183.374	0.158%	17852	691.388	0.594%	6883	99.142%

Table 5 Results in a Branch-and-Cut framework on the instances unsolved in two hours by the SD method. Instances with a * have been solved by the CGD method.

grality gap is closed by Cplex's cutting planes with default parameters, while the remaining 71.73% is closed by our branching algorithm in 925738 nodes. Note that Balas and Saxena used the preprocessed problem as input, while in this paper we always work with the original instances (i.e. without preprocessing). `10teams`, `gesa2_o`, `harp2`, `rout` and `tr12-30` are five other instances where CGD greatly outperforms SD. Among examples that were solved by both algorithms (see Table 6), `bell3a` required 15955 nodes using SD versus only 20 using CGD, `bell5` required 773432 nodes using SD versus 24 using CGD, and `gesa2` required 38539 nodes using SD versus 140 using CGD. There is also an improvement in computing time by several orders of magnitude on these three instances.

On those instances which are solved by both methods, CGD requires on average only half the nodes needed by SD, and the average CPU time is very close for both methods (with a slight advantage for SD). Full results are reported in Table 6.

Summarizing, in our experiments the combination between SD and IGD, which we have called CGD, seems clearly superior to the traditional branching strategy that is represented by branching on single variables. Moreover, as Cplex's callable library is not optimized for branching on general disjunctions, the implementation of CGD could be made faster.

	Gap Closed			SD algorithm		CGD algorithm	
	By Cuts	By Branching			Time		Time
Instance		Abs.	Rel.	Nodes	[sec]	Nodes	[sec]
aflow30a	65.9%	59.6358	34.1%	1813	77.886	1725	99.839
air04	17.9%	494.084	82.1%	181	164.972	203	683.874
air05	15.1%	421.787	84.9%	209	105.902	241	133.679
bell3a	70.8%	4638.26	29.2%	15955	11.822	20	0.047
bell3b	89.6%	39855.3	10.4%	1206	2.177	526	5.512
bell4	91.93%	44957.8	8.07%	9091	24.177	3636	24.242
bell5	85.6%	51456.8	14.4%	773432	553.703	24	0.128
blend2	23.2%	0.524858	76.8%	539	5.321	454	9.920
bm23	24.8%	10.0974	75.2%	119	0.272	78	0.364
cap6000	37.6%	113.47	62.4%	289	30.176	236	111.553
dcmulti	68.5%	1323.83	31.5%	41	1.050	56	2.853
dsbmip	100%	0	0%	15	1.666	23	2.754
egout	35.7%	568.101	64.3%	1	0.009	1	0.011
fiber	91.83%	20400.8	8.17%	153	3.944	28	4.025
fixnet3	97.98%	227.43	2.02%	5	0.300	5	0.421
fixnet4	87.7%	573.738	12.3%	33	1.438	52	8.526
fixnet6	83.4%	461.791	16.6%	1087	20.417	1365	52.859
flugpl	11.8%	30286.3	88.2%	199	0.074	16	0.021
gen	100%	112313	0%	0	0.021	0	0.026
gesa2	74.9%	76271.3	25.1%	38539	1232.150	140	28.490
gesa3	69.3%	48425.7	30.7%	51	2.149	63	4.016
gesa3_o	70.9%	45960.5	29.1%	89	3.934	34	9.955
gt2	91.65%	643.634	8.35%	236	0.412	43	0.139
khb05250	99.9336%	7317.49	0.0664%	5	0.106	2	0.105
l152lav	30.1%	65.4949	69.9%	552	15.614	149	16.251
lp4l	76%	5.875	24%	3	0.059	3	0.160
lseu	68.1%	91.0289	31.9%	61	0.181	46	0.349
manna81	100%	0	0%	0	0.143	0	0.147
mas76	4.2%	1065.02	95.8%	309659	651.702	377398	2608.890
misc01	44.5%	281.057	55.5%	251	3.836	274	7.151
misc02	56.6%	295.312	43.4%	19	0.148	10	0.191
misc03	9.8%	1308.17	90.2%	255	3.73	496	11.875
misc05	45.2%	29.3913	54.8%	103	1.658	33	0.823
misc06	26.5%	6.83269	73.5%	17	1.110	17	2.365
misc07	5.8%	1313.75	94.2%	12139	462.649	25940	1536.48
mitre	100%	0	0%	15	4.394	15	10.759
mod008	21.9%	12.5493	78.1%	345	0.937	13	0.095
mod010	28%	11.5	72%	25	0.567	2	0.440
mod011	68.2%	2.40503e+06	31.8%	707	2633.340	250	3539.000
mod013	30.1%	17.4348	69.9%	115	0.317	107	0.422
modglob	73.7%	81583	26.3%	1879	48.692	2387	75.699
nw04	9.1%	501.358	90.9%	83	75.879	48	109.610
p0033	99.9159%	0.478261	0.0841%	3	0.005	3	0.007
p0040	100%	62027	0%	0	0.002	0	0.001
p0201	46%	400	54%	69	1.147	50	1.635
p0282	96.99%	2458.44	3.01%	23	0.218	12	0.261
p0291	48.5%	5223.75	51.5%	0	0.017	0	0.018
p0548	99.9274%	6.08471	0.0726%	9	0.076	6	0.157
p2756	98.49%	6.56956	1.51%	7	0.364	13	1.205
pipex	63.5%	5.30334	36.5%	19	0.041	12	0.050
pk1	0%	11	100%	243317	956.355	189740	1468.170
pp08aCUTS	87.1%	240.666	12.9%	711	12.363	658	18.583
pp08a	94.38%	258.537	5.62%	392	4.633	372	4.481
qiu	0%	798.766	100%	19399	2780.000	19399	2901.890
qnet1	71%	509.709	29%	53	3.156	74	26.939
qnet1_o	85.1%	585.272	14.9%	17	1.267	13	3.826
rentacar	51%	759381	49%	11	12.047	11	14.973
rgn	15.9%	28.0903	84.1%	2089	2.143	1703	3.826
sample2	46.5%	68.4556	53.5%	35	0.092	33	0.103
sentoy	24.9%	50.6089	75.1%	52	0.175	53	0.266
set1al	99.9521%	2.2619	0.0479%	5	0.056	6	0.145
set1cl	34.7%	6484.25	65.3%	0	0.021	0	0.023
stein15	0%	2	100%	42	0.058	44	0.068
stein27	0%	5	100%	1628	3.785	1537	3.721
stein45	0%	8	100%	29676	218.862	28882	215.015
vpm1	89.1%	0.5	10.9%	17	0.092	17	0.107
vpm2	77%	0.888645	23%	1299	15.646	477	5.723

Table 6 Results in a Branch-and-Cut framework on the instances solved by both the SD and the CGD method.

References

1. Aardal, K., Bixby, R.E., Hurkens, C.A.J., Lenstra, A.K., Smeltink, J.W.: Market split and basis reduction: Towards a solution of the Cornuéjols-Dawande instances. INFORMS Journal on Computing **12**(3), 192–202 (2000). DOI http://dx.doi.org/10.1287/ijoc.12.3.192.12635
2. Andersen, K., Cornuéjols, G., Li, Y.: Reduce-and-split cuts: Improving the performance of mixed integer Gomory cuts. Management Science **51**(11), 1720–1732 (2005)
3. Balas, E.: Disjunctive programming. Annals of Discrete Mathematics **5**, 3–51 (1979)
4. Balas, E., Ceria, S., Cornuéjols, G.: Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. Management Science **42**(9), 1229–1246 (1996)
5. Balas, E., Saxena, A.: Optimizing over the split closure. Mathematical Programming **113**(2), 219–240 (2008)
6. Coin-or cut generation library. URL `https://projects.coin-or.org/Cgl`
7. ILOG: ILOG CPLEX 11.0 User's Manual. ILOG S.A., Gentilly, France (2007)
8. Karamanov, M., Cornuéjols, G.: Branching on general disjunctions. Tech. rep., Carnegie Mellon University (2005). URL `http://integer.tepper.cmu.edu`
9. Owen, J., Mehrotra, S.: Experimental results on using general disjunctions in branch-and-bound for general-integer linear program. Computational Optimization and Applications **20**, 159–170 (2001)