

# Improved strategies for branching on general disjunctions

G. Cornuéjols · L. Liberti · G. Nannicini

Received: July 21, 2008 / Revised: May 5, 2009

**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, on instances that contain both integer and continuous variables the number of nodes in the enumeration tree is reduced by more than a factor of two, and 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 [9], 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 in the form  $x_j \leq k$  on one branch and  $x_j \geq k + 1$  on the other branch, where  $k$  is an integer. However, this need not be so: any disjunction 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 a general disjunction is considered impractical because of the large computational effort needed to find a suitable branching direction  $\pi$ . A recent paper [10] proposes branching on general disjunctions arising from Gomory Mixed-Integer Cuts

---

G. Cornuéjols

LIF, Faculté de Sciences de Luminy, Marseille, France and  
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

E-mail: gc0v@andrew.cmu.edu

Supported by NSF grant CMMI0653419, ONR grant N00014-03-1-0188 and ANR grant BLAN06-1-138894.

L. Liberti, G. Nannicini

LIX, École Polytechnique, 91128 Palaiseau, France

E-mail: {liberti, giacomon}@lix.polytechnique.fr

Supported by ANR grant 07-JCJC-0151.

(GMICs). GMICs can be viewed as intersection cuts [4]. At each node there is a choice of possible GMICs from which to derive the branching disjunction. The branching strategy suggested in [10] 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 after branching 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 for mixed-integer instances; instead of simply computing the disjunctions that define GMICs 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 propose a branching algorithm that combines branching on general disjunctions and on single variables, by trying to identify, on each instance, whether branching on general disjunctions is profitable and worth the computational overhead. On mixed-integer instances, we obtain a reduction of a factor 2.5 in the average number of enumerated nodes, and a slight advantage in computing time.

## 2 Preliminaries and notation

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

$$\left. \begin{array}{l} \min c^\top x \\ Ax = b \\ x \geq 0 \\ \forall j \in N_I \quad x_j \in \mathbb{Z}, \end{array} \right\} \mathcal{P} \quad (1)$$

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  $\tilde{\mathcal{P}}$ . The Branch-and-Bound algorithm makes an implicit use of the concept of disjunctions [5]: whenever the solution of the current relaxation is fractional, we divide the current problem  $\mathcal{P}$  into two subproblems  $\mathcal{P}_1$  and  $\mathcal{P}_2$  such that the union of the feasible regions of  $\mathcal{P}_1$  and  $\mathcal{P}_2$  contains all feasible solutions to  $\mathcal{P}$ . Usually, this is done by choosing a fractional component  $\bar{x}_i$  (for some  $i \in N_I$ ) of the optimal solution  $\bar{x}$  to the relaxation  $\tilde{\mathcal{P}}$ , and adding the constraints  $x_i \leq \lfloor \bar{x}_i \rfloor$  and  $x_i \geq \lceil \bar{x}_i \rceil$  to  $\mathcal{P}_1$  and  $\mathcal{P}_2$  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  $\mathcal{P}_1$  and  $\mathcal{P}_2$  respectively, as long as no integer feasible point is cut off. Owen and Mehrotra [11] generated branching directions  $\pi$  where  $\pi_j \in \{-1, 0, +1\}$  for all  $j \in N_I$  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 [10] proposed using disjunctions arising from GMICs generated directly from the rows of the optimal tableau. Given  $B \subset N$  an optimal basis of  $\tilde{\mathcal{P}}$ , and  $J = N \setminus B$ , i.e.  $J$  is the set of nonbasic variables, the corresponding simplex tableau is given by

$$x_i = \bar{x}_i - \sum_{j \in J} \bar{a}_{ij} x_j \quad \forall i \in B. \quad (2)$$

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

$$r_i^j = \begin{cases} -\bar{a}_{ij} & \text{if } i \in B \\ 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

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  $\bar{x}$ . Let  $D(\pi, \pi_0)$  define the *split disjunction*  $\pi^\top x \leq \pi_0 \vee \pi^\top x \geq \pi_0 + 1$ , where  $\pi \in \mathbb{Z}^n, \pi_0 \in \mathbb{Z}, \pi_j = 0$  for  $i \notin N_I, \pi_0 = \lfloor \pi^\top \bar{x} \rfloor$ . By integrality of  $(\pi, \pi_0)$ , any feasible solution of  $\mathcal{P}$  satisfies every split disjunction. Let  $\varepsilon(\pi, \pi_0) = \pi^\top \bar{x} - \pi_0$  be the violation by  $\bar{x}$  of the first term of  $D(\pi, \pi_0)$ . Assume that the disjunction  $D(\pi, \pi_0)$  is violated by  $\bar{x}$ , i.e.  $0 < \varepsilon(\pi, \pi_0) < 1$ . Balas [4] defines the *intersection cut* associated with a basis  $B$  and a split disjunction  $D(\pi, \pi_0)$  as

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

where for all  $j \in J$  we define

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

The Euclidean distance between  $\bar{x}$  and the cut defined above is (see [6]):

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

### 3 Branching on general disjunctions: a quadratic optimization approach

Assume that the optimal solution  $\bar{x}$  to the LP relaxation  $\tilde{\mathcal{P}}$  is not feasible to  $\mathcal{P}$ . We would like to generate a good branching disjunction  $D(\pi, \pi_0)$ . In [10] 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 from (6) that this means increasing the value of  $\alpha_j(\pi, \pi_0)$  for all  $j \in J$ , which in turn corresponds to decreasing the coefficient of the intersection cut (4). [10] considered intersection cuts (4) obtained directly from the optimal tableau (2) as GMICs for  $i \in B \cap N_I$  such that  $\bar{x}_i \notin \mathbb{Z}$ .

In this paper we consider split disjunctions arising from GMICs generated from linear combinations of the rows of the simplex tableau (2):

$$\sum_{i \in B} \lambda_i x_i = \bar{x} - \sum_{j \in J} \tilde{a}_j x_j, \quad (7)$$

where

$$\begin{aligned} \tilde{x} &= \sum_{i \in B} \lambda_i \bar{x}_i \\ \tilde{a}_j &= \sum_{i \in B} \lambda_i \tilde{a}_{ij} \text{ for } j \in J. \end{aligned} \quad (8)$$

Note that in order to generate a GMIC we need  $\tilde{x} \notin \mathbb{Z}$  and therefore  $\lambda \neq 0$ . For  $i \in B$ , it will be convenient to define  $\tilde{a}_i = \lambda_i$ . The GMIC associated with (7) is the inequality

$$\sum_{j \in N} \frac{x_j}{\alpha_j} \geq 1 \quad (9)$$

where

$$\alpha_j = \begin{cases} \max\left(\frac{\tilde{x} - \lfloor \tilde{x} \rfloor}{\tilde{a}_j - \lfloor \tilde{a}_j \rfloor}, \frac{\lceil \tilde{x} \rceil - \tilde{x}}{\lceil \tilde{a}_j \rceil - \tilde{a}_j}\right) & \text{if } j \in N_I \\ \max\left(\frac{\tilde{x} - \lfloor \tilde{x} \rfloor}{\tilde{a}_j}, \frac{\lceil \tilde{x} \rceil - \tilde{x}}{-\tilde{a}_j}\right) & \text{if } j \in N \setminus N_I. \end{cases} \quad (10)$$

By convention,  $\alpha_j$  is equal to  $+\infty$  when one of the denominators is zero in (10). Note that the GMIC (9) may not cut off  $\bar{x}$  when  $\lambda_i$  is not integer for  $i \in B \cap N_I$  or  $\lambda_i \neq 0$  for  $i \in B \setminus N_I$ . On the other hand, when  $\lambda$  is integral for  $i \in B \cap N_I$  and  $\lambda_i = 0$  for  $i \in B \setminus N_I$  the GMIC (9) is of the form (4) since all the basic variables have a zero coefficient  $1/\alpha_j$ . In this case, the distance cut off is given by (6). For this reason, we restrict ourselves to nonzero integral multipliers  $\lambda$ . In this case, the split disjunction  $D(\pi, \pi_0)$  that defines the GMIC associated to (7) can be computed as (see [4, 8]):

$$\pi_j = \begin{cases} \lfloor \tilde{a}_j \rfloor & \text{if } j \in N_I \cap J \text{ and } \tilde{a}_j - \lfloor \tilde{a}_j \rfloor \leq \tilde{x}_i - \lfloor \tilde{x} \rfloor \\ \lceil \tilde{a}_j \rceil & \text{if } j \in N_I \cap J \text{ and } \tilde{a}_j - \lfloor \tilde{a}_j \rfloor > \tilde{x}_i - \lfloor \tilde{x} \rfloor \\ \lambda_j & \text{if } j \in N_I \cap B \\ 0 & \text{otherwise,} \end{cases} \quad (11)$$

$$\pi_0 = \lfloor \pi^\top \bar{x} \rfloor.$$

It is easy to check that if we plug (11) into  $\alpha_j(\pi, \pi_0)$  as defined in (5) we get exactly  $\alpha_j$  as defined in (10) for  $j \in J$ . Since all the basic variables have a zero coefficient  $1/\alpha_j$  for  $j \in B$ , this shows that the GMIC (9) is an intersection cut.

In this paper we study a method for decreasing  $1/\alpha_j$  for  $j \in J$ . 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 choose an integral vector  $\lambda$  that defines (7), which we have seen to have an influence on both the numerators and the denominators of (10) through (8). It seems difficult to optimize  $\alpha_j$  for  $j \in J \cap N_I$  because both terms of the fraction are nonlinear. Furthermore, for  $j \in N_I$ ,  $1/\alpha_j$  is always between 0 and 1 independent of the choice of  $\lambda$ . For  $j \in J \setminus N_I$  the coefficient  $1/\alpha_j$  is not bounded, therefore we concentrate on these coefficients. From (10) we see that the denominator of  $\alpha_j$  for  $j \in J \setminus N_I$  is a linear function of  $\lambda$  through (8), whereas the numerator is a nonlinear function of  $\lambda$  and is always between 0 and 1. For this reason we attempt to minimize  $\tilde{a}_j$  for  $j \in J \setminus N_I$  over integral vectors  $\lambda$ . More specifically, we would like to minimize  $\|\tilde{d}\|$ , where

$$\tilde{d} = (\tilde{a}_j)_{j \in J \setminus N_I}. \quad (12)$$

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  $D \in \mathbb{R}^{|B_I| \times |J_C|}$ ,  $d_{ij} = \tilde{a}_{ij}$ . Minimizing  $\|\tilde{d}\|$  can be written as

$$\min_{\lambda \in \mathbb{Z}^{|B_I|} \setminus \{0\}} \left\| \sum_{i \in B_I} \lambda_i d_i \right\|. \quad (13)$$

This is a shortest vector problem in the additive group generated by the rows of  $D$ . If these rows are linearly independent, the group defines a lattice, and we have the classical shortest vector problem in a lattice, which is NP-hard under randomized reductions [2].

Andersen, Cornuéjols and Li [3] proposed a heuristic for (13) based on a reduction algorithm which cycles through the rows of  $D$  and, for each such row  $d_k$ , considers whether summing an integer multiple of some other row yields a reduction of  $\|d_k\|$ . If this is the case, the matrix  $D$  is updated by replacing  $d_k$  with the shorter vector. Note, however, that this method only considers two rows at a time.

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  $d_i$  for all  $i \in R_k \setminus \{k\}$ . This is done by defining, for each row  $d_k$  that we want to reduce, the convex minimization problem:

$$\min_{\lambda^k \in \mathbb{R}^{|R_k|}, \lambda_k^k = 1} \left\| \sum_{i \in R_k} \lambda_i^k d_i \right\|, \quad (14)$$

and then rounding the coefficients  $\lambda_i^k$  to the nearest integer  $\lceil \lambda_i^k \rceil$ . There are several reasons for imposing  $\lambda_k^k = 1$ . One reason is that not only do we want to find a short vector, but it is also important to find a vector  $\lambda^k$  with small norm. We will come back to this issue in Section 3.1. Another reason is that we must avoid the zero vector as a solution. Yet another is to get different optimization problems for  $k = 1, \dots, |B_I|$ , thus increasing the chance of obtaining different branching directions. Vanishing the partial derivatives of  $\|\sum_{i \in R_k} \lambda_i^k d_i\|$  with respect to  $\lambda_i^k$  for all  $i$ , we obtain an  $|R_k| \times |R_k|$  linear system that yields the optimal (continuous) solution. 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:

$$A_{ij}^k = \begin{cases} 1 & \text{if } i = j = k \\ 0 & \text{if } i = k \text{ or } j = k \text{ but not both} \\ \sum_{h=1}^{|J^C|} d_{ih} d_{jh} & \text{otherwise,} \end{cases} \quad (15)$$

$$b_i^k = \begin{cases} 1 & \text{if } i = k \\ -\sum_{h=1}^{|J^C|} d_{ih} d_{kh} & \text{otherwise.} \end{cases}$$

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  $\lambda^k \in \mathbb{R}^{|R_k|}$  for all  $k \in \{1, \dots, |B_I|\}$ , we round them to the nearest integer. Then, we consider the norm of  $\sum_{i \in R_k} \lceil \lambda_i^k \rceil d_i$ . If  $\|\sum_{i \in R_k} \lceil \lambda_i^k \rceil d_i\| < \|d_k\|$ , then we have an improvement with respect to the original row of the simplex tableau; in this case, we use

$$\sum_{i \in R_k} \lambda_i^k x_i = \sum_{i \in R_k} \lambda_i^k \bar{x}_i - \sum_{j \in J} \sum_{i \in R_k} \lambda_i^k \bar{a}_{ij} x_j, \quad (16)$$

instead of row  $\bar{a}_k$  in order to compute a GMIC, 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 (15), we obtain the linear system:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 194 & -9 & 1 \\ 0 & -9 & 60 & 2 \\ 0 & 1 & 2 & 8 \end{bmatrix} \begin{bmatrix} \lambda_1^1 \\ \lambda_2^1 \\ \lambda_3^1 \\ \lambda_4^1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ -52 \\ -20 \end{bmatrix},$$

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  $\lfloor \lambda^1 \rfloor^\top D$  we obtain the row:

$$[1 \ 0 \ 2 \ -1 \ -1 \ -3 \ 0],$$

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  $\lambda^1$ , 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  $\lfloor \lambda^2 \rfloor^\top D$  gives the original row  $d_2$ , hence the reduction algorithm did not yield an improvement.

### 3.1 The importance of the norm of $\lambda$

Although solving the shortest vector problem (13) is important for finding a deep cut, it is not the only consideration when trying to find a good branching direction. In the space  $B \cap N_I$ , the distance between the two hyperplanes that define a split disjunction  $D(\pi, \pi_0)$  is related to the norm of  $\lambda$ , and is in fact equal to  $1/\|\lambda\|$  as can be seen from (11). Therefore, in this space disjunctions that cut off a larger volume will have a small  $\|\lambda\|$ . We illustrate this with an example.

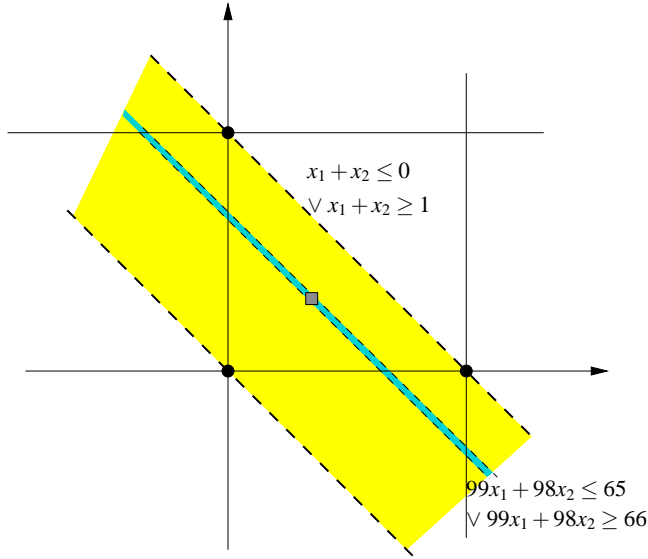
*Example 2* Consider the following tableau, where  $x_1, x_2$  are binary variables and  $y_1, y_2$  are continuous:

$$\begin{cases} x_1 = 1/3 + 98y_1 + y_2 \\ x_2 = 1/3 - 99y_1 - 1.01y_2 \end{cases} \quad (17)$$

The solution to the shortest vector problem (13) is given by the integer multipliers  $\lambda_1 = 99, \lambda_2 = 98$  which yield the shortest vector in the lattice  $(0, 0.02)$  and the disjunction  $99x_1 + 98x_2 \leq 65 \vee 99x_1 + 98x_2 \geq 66$ . Our heuristic method computes the continuous multipliers  $\lambda_1 = 1, \lambda_2 = 98/99$  which are rounded to  $\lambda_1 = 1, \lambda_2 = 1$ , that correspond to the disjunction  $x_1 + x_2 \leq 0 \vee x_1 + x_2 \geq 1$ . It is easy to verify that the distance between these two hyperplanes is roughly ten times larger than in the first case. Therefore, in the unit square, the disjunction obtained through our heuristic method dominates the one computed through the exact solution of the shortest vector problem. Figure 1 gives a picture of this.

### 3.2 Choosing the set $R_k$

The choice of each set  $R_k \subset B_I$  for all  $k$  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



**Fig. 1** Representation of the disjunctions discussed in Example 2.

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  $\{i \in J \cap N_I \mid \bar{a}_{ki} = 0\}$ , 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  $\sum_{j \in R_k} \lambda_j^k \bar{a}_j$ . As zero coefficients on those variables yield a stronger cut, we argue that this will yield a stronger split disjunction as well. During the sorting operation, to keep computational times to a minimum we use the row number as a tie breaker.

### 3.3 The depth of the cut is not always a good measure

As stated earlier, the improvement in the objective function given by a GMIC is a lower bound on the improvement obtained by branching on the corresponding disjunction. We give an example showing that this lower bound may not be tight and in fact the difference between these two values can be arbitrarily large.

*Example 3* Consider the integer program:

$$\left. \begin{array}{ll}
 \min & -x_1 - x_2 \\
 & x_1 \leq 1.5 \\
 & x_2 \leq 1 \\
 & x_1/m - x_2 \geq 1.5/m - 1.25 \\
 & mx_1 - x_2 \leq 1.5m - 0.75 \\
 x_1, x_2 & \in \mathbb{Z},
 \end{array} \right\} \mathcal{P} \quad (18)$$

where  $m > 1$  is a given parameter close to 1. The solution to the LP relaxation is  $(1.5, 1)$ , with an objective value of  $-2.5$ . The intersection cut obtained from the disjunction  $x_1 - x_2 \leq 0 \vee x_1 - x_2 \geq 1$  is  $x_1 + x_2 \leq 2$ , which gives an objective value of  $-2$ . Now suppose we branch on  $x_1 - x_2 \leq 0 \vee x_1 - x_2 \geq 1$ . We obtain two children  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , which are both feasible. One can verify that optimal solution to the LP relaxation of  $\mathcal{P}_1$  is  $(\frac{1.5-1.25m}{1-m}, \frac{1.5-1.25m}{1-m})$  with objective value  $-2\frac{1.5-1.25m}{1-m}$ , and the optimal solution to the LP relaxation of  $\mathcal{P}_2$  is  $(\frac{1.5m-1.75}{m-1}, \frac{0.5m-0.75}{m-1})$  with objective value  $-\frac{2m-2.5}{m-1}$ . Therefore, the gap closed by branching is:

$$\max\left\{-2\frac{1.5-1.25m}{1-m}, -\frac{2m-2.5}{m-1}\right\} - 2.5,$$

which can be made arbitrarily large when  $m$  tends to 1 from above. At the same time, the intersection cut associated with the same disjunction closes a gap of 0.5 regardless of  $m$ . We give a picture of the situation for  $m = 1.1$  in Figure 2.

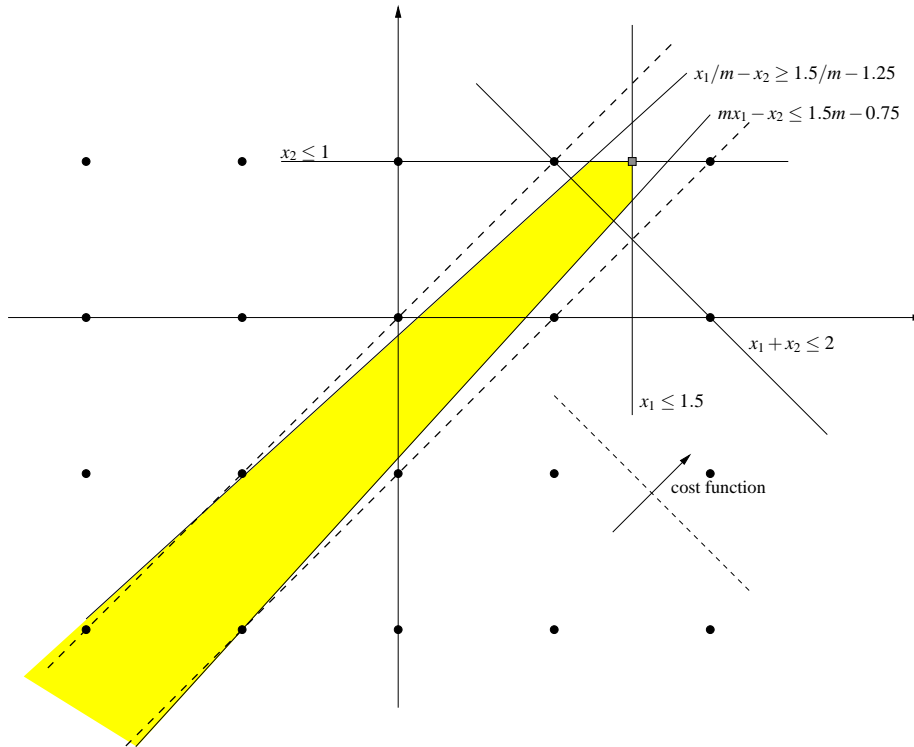


Fig. 2 Representation of Example 3 for  $m = 1.1$ .

Example 3 suggests that we should not choose the branching direction by relying solely on the distance cut off by the corresponding intersection cut, even though this distance gives an indication of the strength of a disjunction and can guide us towards generating better ones. Therefore, in our computational experiments (see Section 4) we employed strong branching to select a disjunction among those computed with the procedure described so far.

## 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 GMICs at the current basis (General Disjunctions, GD), in order to compare with the work of Karamanov and Cornuéjols [10],
- 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 [3],
- 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. 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. Note that these solutions are computed by employing the `CPXdualopt()` function of the Cplex's callable library, because the only available strong branching function, `CPXstrongbranch()`, does not support branching on general disjunctions. 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. The node selection strategy was the Cplex default setting (*best bound*), and the value of the optimal solution was given as a cutoff value for all those instances where the optimum is known<sup>1</sup>. Finally, we set the `mipgap` parameter to zero. 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.

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 all the algorithms<sup>2</sup>, and the instances where the root node, including strong branching, cannot be processed in less than 30 minutes by all the algorithms<sup>3</sup>. In total, the set consists in 96 heterogeneous instances. We divided this set into two disjoint subsets: the first one, which we call `MixedInteger`, contains only the 57 instances with more than one continuous variable. The second subset, which we call `PureInteger`, contains pure integer problems and mixed-integer problems with only one continuous variable. We are mainly interested in `MixedInteger`; however, at the end of this section we will discuss some experiments performed on `PureInteger` employing the combined branching algorithm CGD (Section 4.2). The list of instances in the two test sets can be given in Table 1 and Table 2. Note that

<sup>1</sup> See the `miplib2003` web site: <http://miplib.zib.de/miplib2003.php>.

<sup>2</sup> The instances are: `air01`, `air02`, `air03`, `air06`, `misc04`, `stein09`.

<sup>3</sup> The instances are: `atlanta-ip`, `ds`, `momentum1`, `momentum2`, `momentum3`, `msc98-ip`, `mzzv11`, `mzzv42z`, `net12`, `rd-rplusc-21`, `stp3d`.

10teams	a1c1s1	aflow30a	aflow40b	arki001	bell3a
bell13b	bell14	bell15	blend2	dano3mip	dano1nt
dcmulti	dsbmip	egout	fiber	fixnet3	fixnet4
fixnet6	flugpl	gen	gesa2	gesa2_o	gesa3
gesa3_o	glass4	khb05250	liu	markshare1	markshare2
misc05	misc06	mkc	mod011	mod013	modglob
noswot	pk1	pp08aCUTS	pp08a	qiu	qnet1
qnet1_o	rentacar	rgn	roll3000	rout	sample2
set1a1	set1ch	set1cl	swath	timtab1	timtab2
tri2-30	vpm1	vpm2			

**Table 1** List of instances in the `MixedInteger` test set.

air04	air05	bm23	cap6000	fast0507	gt2
harp2	l152lav	lp41	lseu	manna81	mas74
mas76	misc01	misc02	misc03	misc07	mitre
mod008	mod010	nsrand-ix	nw04	opt1217	p0033
p0040	p0201	p0282	p0291	p0548	p2756
pipex	protfold	sentoy	seymour	sp97ar	stein15
stein27	stein45	t1717			

**Table 2** List of instances in the `PureInteger` test set.

the norm reduction algorithm proposed in this paper can be applied only to instances with continuous variables; if continuous variables are not present in the original formulation of the problem, they are introduced as artificial variables when reformulating the problem into standard form or adding cutting planes. Therefore, we were able to apply the proposed procedure on the `PureInteger` instances.

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; to compute the geometric average of a set of values not necessarily greater than zero, we added 1 to all values before computing the mean, and subtracted 1 from the result. Note that this rule was also applied for the computation of average CPU times. Section 4.3 concludes the paper.

#### 4.1 Comparison of the different methods on mixed-integer instances

The first set of experiments that we discuss aims at choosing the number of rows  $M_{|R_k|}$  that should be considered for the norm reduction procedure describes in Section 3. In this section we only consider the mixed-integer instances, i.e. the `MixedInteger` test set. First, we analyze whether the rows of the matrix  $D$  are linearly independent or not. To do so, we computed the matrix  $D$  for each instance from the optimal tableau at root node, and solved the minimum norm problem (13). If the norm of the resulting linear combination of rows was found to be smaller than  $10^{-5}$ , the rows of  $D$  were considered linearly dependent. Over the whole test set `MixedInteger`, no instance satisfied this linear dependence criterion. Therefore, the problem (13) is indeed a shortest vector problem in a lattice for all instances. Note, however, that our heuristic procedure to compute the coefficients  $\lambda$ 's could be applied even if the rows of  $D$  were linearly dependent.

Section 3.3 motivates the use of strong branching to choose among the branching decisions when branching on general disjunctions. However, 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 following, 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 fractional part (i.e. closer to 0.5). For GD and IGD, we picked the 10 split disjunctions associated with the 10 GMICs that have the largest cut off distance (equation (6), see [10]), where for IGD the distance was computed after the reduction step. We remark that in this case the norm reduction procedure is applied to all rows before computing the distance cut off.

In the next two experiments, we solved up to 1000 nodes in the enumeration tree. The experiments were made in a bare Branch-and-Bound setting; that is, presolving, cutting planes and heuristics were disabled. The branching decision selection method 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. 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. For this set of experiments only, the `dano3mip` instance was 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 [3], 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 GMICs that have the largest cut off distance after the reduction step, and applied strong branching. We used our own implementation of the RS reduction algorithm. A summary of the results is given in Table 3. It can be seen that IGD using 20 or 50 rows for the reduction step yields very 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 significantly smaller for  $M_{|R_k|} = 50$ . In particular IGD outperforms RS for branching. To investigate the reason for this, we recorded the average norm of the disjunction chosen for branching at each node of the enumeration tree; recall (Section 3.1) that the norm depends on the coefficients  $\lambda$  generated by the disjunction improvement algorithm. The average norms (computed through a geometric mean over the whole set of instances) are as follows:

- Reduce-and-Split: 8.55;
- IGD with  $M_{|R_k|} = 10$ : 6.15;
- IGD with  $M_{|R_k|} = 20$ : 6.09;
- IGD with  $M_{|R_k|} = 50$ : 6.24.

Therefore, our heuristic procedure generates smaller  $\lambda$ 's with respect to RS, which results in disjunctions that cut off a larger volume. This has a positive effect on the size of the enumeration tree. We give another possible reason for the superiority of IGD with respect to

Number of solved instances	
RS:	17
IGD with $M_{ R_k } = 10$ (IGD10):	21
IGD with $M_{ R_k } = 20$ (IGD20):	20
IGD with $M_{ R_k } = 50$ (IGD50):	20
Average number of nodes on instances solved by all methods	
RS:	77.7
IGD with $M_{ R_k } = 10$ (IGD10):	73.4
IGD with $M_{ R_k } = 20$ (IGD20):	72.3
IGD with $M_{ R_k } = 50$ (IGD50):	58.8
Average gap closed on instances not solved by any method	
RS:	10.74%
IGD with $M_{ R_k } = 10$ (IGD10):	12.37%
IGD with $M_{ R_k } = 20$ (IGD20):	13.39%
IGD with $M_{ R_k } = 50$ (IGD50):	13.03%

**Table 3** Results on the `MixedInteger` test set after 1000 solved nodes

RS for branching. 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 [3]. However, at each node of a Branch-and-Bound tree only one disjunction is chosen for branching, so a method which tries to compute only one strong disjunction is more fruitful than one that generates a set of several possibly weaker ones. This, combined with smaller coefficient  $\lambda$ 's at the end of the reduction procedure, may explain why the 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 4. The increase in the gap per node that can be closed by branching on general disjunctions with respect to branching on single variables is large. This is confirmed by the results in [10]. 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 36 instances over the 57 instances in `MixedInteger`.

On the instances solved by all methods, SD is roughly two times faster than GD and IGD. Moreover, if we consider only the instances not solved by any method (i.e. all branching algorithms solve 1000 nodes without reaching optimality) we obtain the following average times:

- SD: 32.59 seconds;
- GD: 150.61 seconds;
- IGD: 176.78 seconds.

This suggests that branching on general disjunctions introduces a significant computational overhead at each node. The average time spent per node by the three methods, recorded as

Number of solved instances	
Simple disjunctions (SD):	15
General disjunctions (GD):	20
Improved general disjunctions (IGD):	20
Average number of nodes on instances solved by all methods	
Simple disjunctions (SD):	125.6
General disjunctions (GD):	98.1
Improved general disjunctions (IGD):	75.3
Average CPU time [sec] on instances solved by all methods	
Simple disjunctions (SD):	2.53
General disjunctions (GD):	5.23
Improved general disjunctions (IGD):	4.79
Average gap closed on instances not solved by any method	
Simple disjunctions (SD):	9.02%
General disjunctions (GD):	12.99%
Improved general disjunctions (IGD):	13.30%

**Table 4** Results on the `MixedInteger` test set after 1000 solved nodes

the geometric mean of the average time spent per node over all the instances in `MixedInteger`, is as follows:

- SD: 0.02 seconds;
- GD: 0.08 seconds;
- IGD: 0.10 seconds.

Therefore, 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 GMIC 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 disjunctions, 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. Note, however, that in terms of average time per node IGD is only  $\approx 25\%$  slower than GD, which suggests that the norm reduction routine is not the most computationally expensive step that is carried out at each node.

Moreover, although IGD performs on average better than SD, there are some instances where branching on simple disjunctions achieves significantly better results. We give two such examples: on the `a1c1s1` instance, after 1000 nodes SD closes 4.33% of the integrality gap employing 217.1 seconds of CPU time, whereas IGD closes 1.66% and takes 6806.4 seconds; on the `dcmulti` instance, SD closes all the integrality in gap in 453 nodes and

**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
  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

```

---

2.3 seconds, whereas IGD closes only 48.8% of the integrality gap in 1000 nodes and 54.6 seconds.

#### 4.2 Combination of several methods

Results in Table 4 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, 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.

Since branching on general disjunctions is slower than branching on simple disjunctions, it 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 the 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 deactivated (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 GMIC 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 with the default parameters, and branched for two hours. Again, preprocessing and heuristics were disabled.

#### 4.2.1 Results on the mixed-integer instances

We focus on the `MixedInteger` instances first. In Table 5 we compare the number of solved instances within the two hours limit, the average number of nodes and average CPU time on instances solved by both methods, and the average number of nodes and average closed gap on instances not solved by either method. For the computation of the average closed gap on instances not solved by either method, only instances where at least one method closed a nonzero integrality gap were taken into account. The results clearly indicate that

Number of solved instances	
Simple disjunctions (SD):	37
Combined general disjunctions (CGD):	40
Average number of nodes on instances solved by both methods	
Simple disjunctions (SD):	268.9
Combined general disjunctions (CGD):	106.2
Average CPU time [sec] on instances solved by both methods	
Simple disjunctions (SD):	4.46
Combined general disjunctions (CGD):	3.91
Average number of nodes on instances not solved by either method	
Simple disjunctions (SD):	62683.4
Combined general disjunctions (CGD):	27222.7
Average gap closed on instances not solved by either method	
Simple disjunctions (SD):	5.63%
Combined general disjunctions (CGD):	7.65%

**Table 5** Results on the `MixedInteger` test set in a Branch-and-Cut framework with a two hours time limit

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 two methods, the average integrality gap closed by CGD is 35% 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 6. If we consider the 3 instances of this set 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 instance (`timtab2`) SD closes more gap. However, on `timtab2` 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 is small: only 0.13%, but CGD requires 4 times fewer nodes with respect to SD.

On a few instances, CGD performs strikingly better than SD. One example is the `arki001` instance, which is a difficult instances of `mip1ib2003`: branching with CGD closes 45% of the gap, whereas branching with SD only closes 6.83%. The `arki001` instance was first solved to optimality only recently by Balas and Saxena [7]: 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

INSTANCE	SD ALGORITHM			CGD ALGORITHM			GAP CLOSED BY CUTS
	CLOSED GAP		NODES	CLOSED GAP		NODES	
	ABS.	REL.		ABS.	REL.		
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%
danooint	0.310476	10.2%	5547	0.286139	9.44%	4790	2%
gesa2.o*	84644.7	27.9%	195797	147352	48.5%	13181	51.4%
glass4	3293.85	0%	84369	3104.73	0%	79050	0%
liu	214	-	108162	214	-	100347	0%
markshare1	0	0%	11027872	0	0%	2540405	0%
markshare2	0	0%	8606987	0	0%	2431791	0%
mkc	2.92749	6.1%	14486	6.52824	13.6%	8663	5.7%
noswot	0	0%	3192040	0	0%	1598812	0%
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%
swath	28.3223	21.3%	20831	15.7973	11.9%	4724	34.9%
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 6** Results in a Branch-and-Cut framework on the `MixedInteger` instances unsolved in two hours by the SD method. Instances with a \* have been solved by the CGD method.

the integrality 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`, `rout` and `tr12-30` are four other instances where CGD greatly outperforms SD. Among examples that were solved by both algorithms (see Table 7), `be113a` required 15955 nodes using SD versus only 20 using CGD, `be115` 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 less than half the nodes needed by SD: the reduction in the size of the enumeration tree is of a factor 2.5. The average CPU time is very close for both methods (with a slight advantage for CGD). Full results are reported in Table 7.

#### 4.2.2 Results on the pure integer instances

As stated earlier, the combined branching algorithm CGD can be employed on pure integer instances: as continuous variables are introduced when the problem is reformulated in standard form, the disjunction improvement procedure described in Section 3 can be carried out. However, as in this case we are considering combinations of rows by looking at the coefficients on the slack variables only, we do not expect to obtain an improvement as large as in the mixed-integer case. Here we briefly analyze the performance of Algorithm 1, i.e. CGD on the `PureInteger` instances. In Table 8 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. Both

INSTANCE	GAP CLOSED			SD ALGORITHM		CGD ALGORITHM	
	BY CUTS	BY BRANCHING		NODES	TIME [SEC]	NODES	TIME [SEC]
		ABS.	REL.				
aflow30a	65.9%	59.6358	34.1%	1813	77.886	1725	99.839
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
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
khh05250	99.9336%	7317.49	0.0664%	5	0.106	2	0.105
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
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
modg1ob	73.7%	81583	26.3%	1879	48.692	2387	75.699
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
set1a1	99.9521%	2.2619	0.0479%	5	0.056	6	0.145
set1c1	34.7%	6484.25	65.3%	0	0.021	0	0.023
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 7** Results in a Branch-and-Cut framework on the `MixedInteger` instances solved by both the SD and the CGD method.

SD and CGD solve to optimality the same 30 instances over the 39 comprised in the test set. We clearly see that, on average, CGD closes more gap on the unsolved instances ( $\approx 35\%$  more gap than SD in relative terms), and enumerates fewer nodes on the instances that are solved by both methods. However, in this case the reduction in the size of the branching tree is of a factor 1.45, which is not as large as the one obtained on the `MixedInteger` instances (factor 2.5). This suggests that the norm reduction procedure proposed in this paper achieves better results whenever it can be applied on the coefficients of the continuous variables defined in the original problem, instead of acting on artificial variables only. Moreover, the average computing time on `PureInteger` instances is shorter for the SD method, whereas on `MixedInteger` instances the CGD is faster. Overall, the combined branching algorithm shows good performance on `PureInteger` instances: on average, it closes more gap on difficult instances and requires fewer nodes on easy instances; however, the gain is not as large

Number of solved instances	
Simple disjunctions (SD):	30
Combined general disjunctions (CGD):	30
Average number of nodes on instances solved by both methods	
Simple disjunctions (SD):	128.7
Combined general disjunctions (CGD):	88.2
Average CPU time [sec] on instances solved by both methods	
Simple disjunctions (SD):	1.83
Combined general disjunctions (CGD):	2.73
Average number of nodes on instances not solved by either method	
Simple disjunctions (SD):	11673.4
Combined general disjunctions (CGD):	4623.4
Average gap closed on instances not solved by either method	
Simple disjunctions (SD):	12.10%
Combined general disjunctions (CGD):	16.40%

**Table 8** Results on the PureInteger test set in a Branch-and-Cut framework with a two hours time limit

as on MixedInteger instances, therefore the reduction in computing time is not sufficient to offset the rapidity of SD.

#### 4.3 Conclusion

Summarizing, in our experiments the combination between SD and IGD, which we have called CGD, seems clearly superior to the traditional branching strategy (represented by branching on single variables) on mixed-integer instances. On pure integer instances, the combination between branching on general and on simple disjunction closes on average more gap per node, but this does not translate into a reduction in terms of computing time because our disjunction improvement procedure only looks at coefficients on the artificial variables, which is not as effective as working with the original variables. Finally, as Cplex's callable library is not optimized for branching on general disjunctions, the implementation of CGD could be made faster.

#### 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. Ajtai, M.: The shortest vector problem in  $l_2$  is NP-hard for randomized reductions. In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*. Dallas, TX (1998)
3. 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)

4. Balas, E.: Intersection cuts - a new type of cutting planes for integer programming. *Operations Research* **19**(1), 19–39 (1971)
5. Balas, E.: Disjunctive programming. *Annals of Discrete Mathematics* **5**, 3–51 (1979)
6. 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)
7. Balas, E., Saxena, A.: Optimizing over the split closure. *Mathematical Programming* **113**(2), 219–240 (2008)
8. Gomory, R.E.: An algorithm for integer solutions to linear programs. In: P. Wolfe (ed.) *Recent Advances in Mathematical Programming*, pp. 269–302. McGraw-Hill, New York (1963)
9. ILOG: ILOG CPLEX 11.0 User's Manual. ILOG S.A., Gentilly, France (2007)
10. Karamanov, M., Cornuéjols, G.: Branching on general disjunctions. Tech. rep., Carnegie Mellon University (2005). URL <http://integer.tepper.cmu.edu>
11. 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)