Branching on Split Disjunctions

G. NANNICINI [a,1], G. CORNUÉJOLS [a,2], M. KARAMANOV [b], L. LIBERTI [c,3]
[a] *Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA*
[b] *Capacity and Operations Planning, Bank of America, Charlotte, NC*
[c] *LIX, Ecole Polytechnique, Palaiseau, France*

Abstract. Branch-and-Cut is the most commonly used algorithm for solving Integer and Mixed-Integer Linear Programs. In order to reduce the number of nodes that have to be enumerated before optimality of a solution can be proven, branching on general disjunctions (i.e. split disjunctions involving more than one variable, as opposed to branching on simple disjunctions defined on one variable only) was shown to be very effective on particular classes of instances, but not much work has been done to study general purpose methods of this kind. In this paper, we survey known results related to this line of research, and we study the relationship between branching and cutting from a split disjunction.

Keywords. Integer programming, Branch-and-Bound, Split disjunctions

Introduction

Solving Mixed-Integer Linear Programs (MILPs) is of great practical use in a number of applications, and efficient software exists for this purpose. One key ingredient is the Branch-and-Bound algorithm [21]. Branch-and-Bound has two main components: dividing a problem into subproblems, which is known as branching, and computing bounds on the objective function value at the subproblems. The idea is to recursively subdivide the initial problem into smaller problems, until the subproblems can be easily solved, and use bounds to eliminate as many as possible. Subproblems are typically stored in a tree structure, hence they are called *nodes* in the literature. The bounding phase is carried out by considering the Linear Programming (LP) relaxation of each node; in this paper, we focus on the branching phase.

Whenever the solution $\bar{x}$ to the LP associated with a node is fractional on a variable x_i that is required to take on integer values, a natural way of branching is to create two subproblems imposing the constraint $x_i \leq \lfloor \bar{x}_i \rfloor$ on one subproblem and $x_i \geq \lceil \bar{x}_i \rceil$ on the other. In this paper, we take a different approach whereby branching can occur on a general hyperplane with integer components π by imposing $\pi^\top x \leq \pi_0$ on one child and $\pi^\top x \geq \pi_0 + 1$ on the other.

How do we choose π? We use the connections between branching and cutting from split disjunctions [12], Gomory Mixed-Integer (GMI) cuts [16] and intersection cuts [7].

[1]Corresponding Author: e-mail: nannicin@andrew.cmu.edu. Supported by an IBM fellowship.
[2]Supported by NSF grant CMMI1024554 and ONR grant N00014-09-1-0033.
[3]Supported by ANR grant 07-JCJC-0151.

GMI cuts arise as intersection cuts from a split disjunction, and provide a computationally inexpensive way of generating π. Therefore, we generate a pool of possible branching hyperplanes this way, and select one by using strong branching [6]. We also investigate the effect of modifying the hyperplanes by strenghtening the underlying GMI cut with a Reduce-and-Split like algorithm [4,15].

Computational experiments on MIPLIB instances show that this approach is effective in practice, and can significantly reduce the size of the enumeration tree; on average, the reduction in number of nodes is by more than a factor two on mixed-integer instances.

Extended versions of this work have appeared in [14,19]. In this paper, we give a unified treating of this topic. In Section 1, we give some preliminaries and survey the research carried out in this area. In Section 2 we introduce our notation and recall several known results that are useful for the subsequent parts of the paper. Section 3 studies the relationship between the integrality gap (i.e. the difference between the integer optimum and the relaxed optimum) closed by generating an intersection cut from a split disjunction, or branching on the same disjunction. In Section 4 we describe a branching scheme that is based on exploiting the disjunctions defining the GMI cuts read directly from an optimal simplex tableau; Section 5 modifies this scheme by adding a disjunction strenghtening step. In Section 6 we discuss the size of the coefficients of “good” disjunctions. Finally, Section 7 concludes the paper with a computational evaluation.

1. Preliminaries and Literature Review

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

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

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 $\mathcal{P}$ is the linear program obtained by dropping the integrality constraints, and is denoted by $\bar{\mathcal{P}}$. We denote by P the set of feasible solutions of $\bar{\mathcal{P}}$, which is a polyhedron. The Branch-and-Bound algorithm makes an implicit use of the concept of disjunctions [8]: whenever the solution to the current LP 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 $\bar{\mathcal{P}}$, 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. Choosing which variable should be branched on at each step is of fundamental importance for the performance of Branch-and-Bound. We refer to [2] for a recent survey on this topic.

Here, we take a 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 feasible point of $\mathcal{P}$ is cut off. A natural way of generating such directions is to consider split disjunctions $D(\pi, \pi_0)$ of the form:

$$\pi^\top x \leq \pi_0 \quad \bigvee \quad \pi^\top x \geq \pi_0 + 1 \tag{1}$$

with $\pi \in \mathbb{Z}^n$, $\pi_0 \in \mathbb{Z}$, $\pi_i = 0 \; \forall i \notin N_I$. By integrality, every feasible solution to $\mathcal{P}$ satisfies any split disjunction. In other words, a split disjunction is defined by two parallel hyperplanes that have no integer point in the interior of the “strip” between them. In the branching literature, disjunctions involving only one variable are labeled *simple* or *elementary*, whereas those involving more than one variable are called *general*.

There are mainly two different categories of approaches to branching on general disjunctions that have been proposed in the MILP literature. The first category contains methods that try to identify “thin” directions of P ; the second category focuses on improving as much as possible the LP bound at the children nodes. [25] discusses both problems, which are shown to be strongly **NP**-hard in [26].

1.1. Branching on thin directions

The concept of thin direction requires the notion of *width* of a full-dimensional polyhedron P along a direction u, which is defined as $\max_{x,y \in P}(ux - uy)$. Thus, for a *pure* integer program associated with P, the *integer width* is defined as

$$\min_{\pi \in \mathbb{Z}^n \setminus \{0\}} \max_{x,y \in P} (\pi x - \pi y).$$

This definition naturally extends to the mixed integer case by considering integer directions $\pi \in \mathbb{Z}^n \setminus \{0\}$ with $\pi_j = 0$ for $j \notin N_I$.

The work of Lenstra [23] on solving integer programs in fixed dimension in polynomial time (see also [17,24]) is at the origin of the idea of branching on thin directions of P. The method works as follows. First, some thin directions of P are computed, using the lattice basis reduction algorithm by Lenstra, Lenstra and Lovász [22]. Then, the space is transformed so that these directions correspond to unit vectors, and the problem is solved by Branch-and-Bound in the new space. Thus, branching on single variables in the transformed space translates back to branching on general disjunctions in the original space. This method has proven successful for some particular instances where standard Branch-and-Bound fails because of the huge size of the enumeration tree, such as the Market Split instances [13], whose solution is discussed in [1]. Other examples are given in [20,27].

1.2. Branching for maximum bound improvement

Another line of research which has been pursued is that of selecting a good general disjunction for branching at each node of the Branch-and-Bound tree, in order to improve as much as possible the bound at the children nodes. Owen and Mehrotra [28] proposed branching on split disjunctions with coefficients in $\{-1, 0, 1\}$ on the integer variables with fractional values at the current node. They generate all possible such disjunctions, and evaluate them using strong branching (i.e. solving the LPs associated with the children nodes to optimality), in order to select the one that gives the largest improvement of the dual bound. [14,19] follow this idea of generating disjunctions for maximum bound improvement, and try to do so by exploiting the relationship between split cuts [12] and split disjunctions for branching.

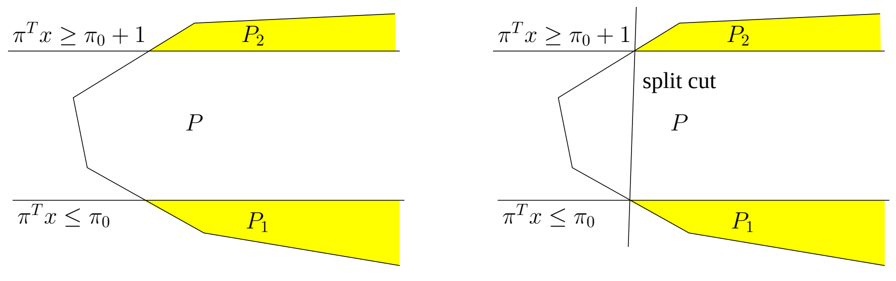


Figure 1. Deriving a split cut.

2. Split Disjunctions: Cutting and Branching

Given a split disjunction $D(\pi, \pi_0)$ of the form (1), a *split cut* for P is a cut which is valid (i.e. does not cut off any integral feasible solution) for both $P_1 = P \cap \{x : \pi^\top x \leq \pi_0\}$ and $P_2 = P \cap \{x : \pi^\top x \geq \pi_0 + 1\}$. Since $P_1 \cup P_2$ contains all integral feasible points of P, such a cut is valid for P as well. Split cuts were introduced in [12]. It is intuitive to see that there should be some kind of relationship between a split cut derived from $D(\pi, \pi_0)$ and the "strength" of the two polyhedra P_1, P_2; since the aim of branching is exactly that of creating two strong (but smaller) subproblems $\mathcal{P}_1, \mathcal{P}_2$ associated with P_1, P_2, we want to study this relationship. To do so, we first investigate some useful properties of split cuts that will lead to the formulas used in the rest of this paper.

Split cuts are disjunctive cuts [8], i.e. cutting planes which are valid for $conv(P_1 \cup P_2)$. A pictorial representation of such a cut is given in Figure 1. Is there an easier way of deriving split cuts without having to resort to disjunctive programming? [5] establishes a correspondence between split cuts and intersection cuts [7], showing that each split cut for P can be derived as an intersection cut from a split disjunction and a suitable basis of $\bar{\mathcal{P}}$. This allows for a geometric understanding of their derivation, and for closed form formulas.

We need some definitions. A *basis* for $\bar{\mathcal{P}}$ is an m-subset B of N such that the column submatrix of A induced by B is an invertible submatrix of A. Let $J := N \setminus B$ denote the index set of the nonbasic variables, $B_I = B \cap N_I$ the set of integer basic variables, $J_I = J \cap N_I$ the set of integer nonbasic variables, $J_C = J \setminus N_I$ the set of continuous nonbasic variables. Additionally, we denote by $\langle x \rangle$ the fractional part of x, i.e. $\langle x \rangle = x - \lfloor x \rfloor$. A further relaxation of the set P with respect to a basis B is obtained by removing the non-negativity constraints on the basic variables. We denote it by $P(B)$:

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

This set is a translate of a polyhedral cone: $P(B) = C + \bar{x}$, where $C = \{x \in \mathbb{R}^n : Ax = 0 \text{ and } x_j \geq 0 \text{ for } j \in J\}$ and $\bar{x}$ solves $\{x \in \mathbb{R}^n : Ax = b \text{ and } x_j = 0 \text{ for } j \in J\}$, i.e. $\bar{x}$ is the *basic solution* corresponding to the basis B. Typically, B will be the optimal basis of an LP relaxation of MILP, but not necessarily so (see e.g. [11]). In this paper, B will be optimal for $\bar{\mathcal{P}}$. The cone C can be expressed also in terms of its extreme rays, r^j for $j \in J$: $P(B) = Cone(\{r^j\}_{j \in J}) + \bar{x}$, where $Cone(\{r^j\})$ denotes the polyhedral cone generated by vectors $\{r^j\}$. Looking at the simplex tableau associated with B written in the usual form:

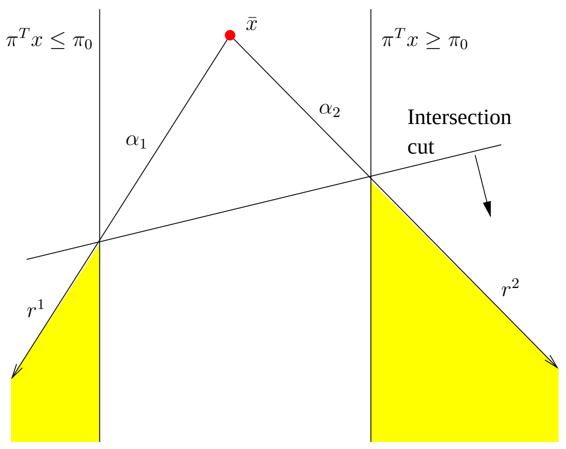


Figure 2. Deriving the intersection cut

$$x_i = \bar{x}_i - \sum_{j \in J} \bar{a}_{ij} x_j \quad \forall i \in B, \tag{3}$$

the extreme rays of $P(B)$ can be read directly as:

$$r_i^j = \begin{cases} -\bar{a}_{ij} & \text{if } i \in B \\ 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Observe that our interest is in split disjunctions that are violated by the current fractional solution $\bar{x}$. This is because we do not want $\bar{x}$ to be feasible for either P_1 or P_2, so that the solution to the LP relaxation is forced to “move” after branching. The same is true for cutting planes: the most interesting ones are typically those that cut off $\bar{x}$. How do we generate a split cut as an intersection cut? Given any disjunction $D(\pi, \pi_0)$ violated by $\bar{x}$, we can generate a split cut using $D(\pi, \pi_0)$ and $P(B)$ as exemplified in Figure 2. In particular, the intersection cut is a half-space bounded by the hyperplane passing through the intersection points of $D(\pi, \pi_0)$ with the extreme rays of $P(B)$.

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

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

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

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

$$\sum_{j\in J} \frac{x_j}{\alpha_j(\pi,\pi_0)} \geq 1. \tag{6}$$

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

$$d(B,\pi,\pi_0) := \sqrt{\frac{1}{\sum_{j\in J} \frac{1}{(\alpha_j(\pi,\pi_0))^2}}} \tag{7}$$

This quantity, called *distance cut off* or *depth*, was used as a measure of cut quality in [9].

The well known GMI cuts from a basis B, which are included in virtually every Branch-and-Cut based software for solving MILPs, can be viewed as intersection cuts from a particular split disjunction [7]. They can be obtained as follows. We start from a disjunction on the integer basic variables:

$$\sum_{i\in B_I} \hat{\pi}_i x_i \leq \left\lfloor \sum_{i\in B_I} \hat{\pi}_i \bar{x}_i \right\rfloor \quad \bigvee \quad \sum_{i\in B_I} \hat{\pi}_i x_i \geq \left\lfloor \sum_{i\in B_I} \hat{\pi}_i \bar{x}_i \right\rfloor + 1, \tag{8}$$

with $\hat{\pi}_i \in \mathbb{Z}\ \forall i \in B_I$ and $\sum_{i\in B_I} \hat{\pi}_i \bar{x}_i \notin \mathbb{Z}$. (8) is then strengthened on the nonbasic integer variables where the affected α_j are modified so that the distance cut off (7) is maximized. We obtain the following disjunction:

$$\pi_j = \begin{cases} \left\lfloor \sum_{i\in B_I} \hat{\pi}_i \bar{a}_{ij} \right\rfloor & \text{if } j \in J_I \text{ and } \left\langle \sum_{i\in B_I} \hat{\pi}_i \bar{a}_{ij} \right\rangle \leq \left\langle \sum_{i\in B_I} \hat{\pi}_i \bar{x}_i \right\rangle \\ \left\lceil \sum_{i\in B_I} \hat{\pi}_i \bar{a}_{ij} \right\rceil & \text{if } j \in J_I \text{ and } \left\langle \sum_{i\in B_I} \hat{\pi}_i \bar{a}_{ij} \right\rangle > \left\langle \sum_{i\in B_I} \hat{\pi}_i \bar{x}_i \right\rangle \\ \hat{\pi}_j & \text{if } j \in B_I \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

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

Plugging (9) into (5) gives a GMI cut that cuts off $\bar{x}$. In the original cutting plane procedure of Gomory [16], (8) is an elementary disjunction. Notice that we have a closed form formula for this disjunction, therefore we can compute it very efficiently. This is the class of split disjunctions that will be employed in the remainder.

3. Gap Closed by Cutting and by Branching

A violated split disjunction can be used for generating an intersection cut but it can be used for branching as well. A good intersection cut cuts deeply into P and improves the LP bound at the children nodes. Our suggestion is that a split disjunction defining a deep cut is good for branching too.

Indeed, the improvement in the lower bound caused by branching on a split disjunction is no less than the improvement by the corresponding intersection cut. Let $\bar{x}_1$ and $\bar{x}_2$ be the LP relaxation optima of $\mathcal{P}_1$ and $\mathcal{P}_2$ (if $\mathcal{P}_k$ is infeasible then $\bar{x}_k = \infty$ for $k \in \{1,2\}$).

Proposition 3.1. $\min_{j\in J} c^\top x^j \leq \min(c^\top \bar{x}_1, c^\top \bar{x}_2)$.

Proof. We have $\bar{x}_1 = \arg\min\{c^\top x : c \in P_1\}$, $\bar{x}_2 = \arg\min\{c^\top x : c \in P_2\}$. Let $P_1^r = P(B) \cap \{x \in \mathbb{R}^n : \pi^\top x \leq \pi_0\}$ and $P_2^r = P(B) \cap \{x \in \mathbb{R}^n : \pi^\top x \geq \pi_0 + 1\}$. By the definition of the points $x^j \; \forall j \in J$, $\min_{j \in J}\{c^\top x^j\} = \min\{c^\top x : x \in P_1^r \cup P_2^r\}$. Since $P \subseteq P(B)$, $\min\{c^\top x : x \in P_1^r \cup P_2^r\} \leq \min\{c^\top x : x \in P_1 \cup P_2\} = \min(c^\top \bar{x}_1, c^\top \bar{x}_2)$, which completes the proof. □

Therefore, in order to generate children nodes that have tight LP relaxations, it makes sense to try to maximize the lower bound given in Proposition 3.1; as this quantity is bounded from below by the gap closed by the corresponding intersection cut, this explains our intuition. However, it can be shown with a small example that the bound on the gap closed by branching given by the corresponding intersection cut can be arbitrarily far from the real value.

Example 3.2. Consider the integer program:

$$\left.\begin{array}{rr} \min & -x_1 - x_2 \\ & x_1 \leq 1.5 \\ & x_2 \leq 1 \\ & x_1/m - x_2 \geq 1.5/m - 1.25 \\ & mx_1 - x_2 \leq 1.5m - 0.75 \\ x_1, x_2 & \in \mathbb{Z}, \end{array}\right\} \mathcal{P} \tag{10}$$

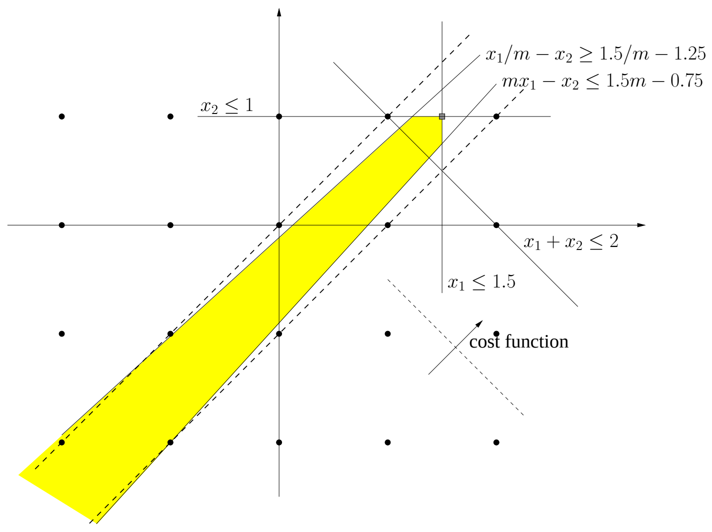
where $m > 1$ is a given parameter close to 1. The solution to the LP relaxation is $(1.5, 1)$, with an objective value of -2.5. The intersection cut obtained from the disjunction $x_1 - x_2 \leq 0 \vee x_1 - x_2 \geq 1$ is $x_1 + x_2 \leq 2$, which gives an objective value of -2. Now suppose we branch 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:

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

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 3.

4. Branching on Disjunctions Defining the GMI Cuts

We need a procedure for selecting promising split disjunctions for branching. As we discussed in the introduction, optimizing over the set of all split disjunctions is strongly **NP**-hard [26]. [19] simply suggests to concentrate on a finite class of general disjunctions generated directly from the current optimal basis — the set $\mathcal{G}$ of split disjunctions defining the GMI cuts that can be read from the simplex tableau. The GMI cuts as defined by Gomory [16] arise from simple disjunctions (8) with $\hat{\pi} = e_i$ for $i \in B_I$ where $\bar{x}_i \notin \mathbb{Z}$. These cuts have been shown to be very effective in practice [10]. The reasons



$x_1/m - x_2 \geq 1.5/m - 1.25$
$mx_1 - x_2 \leq 1.5m - 0.75$
$x_2 \leq 1$
$x_1 + x_2 \leq 2$
$x_1 \leq 1.5$
cost function

Figure 3. Representation of Example 3.2 for $m = 1.1$.

for this choice are the following. First, the set $\mathcal{G}$ is not only finite but relatively small. Its cardinality at a given node of the Branch-and-Bound tree equals the number of integer variables with fractional values in the current basic solution. Second, these disjunctions are fast to obtain. They can be read from the current tableau with a closed form formula (9). Third, as we explained at the end of Section 2, these disjunctions can be viewed as strengthened simple disjunctions (with respect to the cut depth) which suggests that they could perform better than the elementary disjunctions.

The branching procedure proposed in [19] is as follows. Consider the set of all GMI disjunctions arising from elementary disjunctions for a specific basic solution, and select a subset $\mathcal{S}$ of it, containing the most promising disjunctions according to the chosen criterion for comparison. The distance (7) cut off by the underlying intersection cut, is used as a criterion for selecting promising disjunctions, picking those with the largest distance. The cardinality of $\mathcal{S}$ is limited to a parameter k, which can be used to manage the computational effort at different levels, e.g. a larger k can be used close to the root where branching decisions are more important and a smaller k in the deep levels. Finally, in view of Example 3.2, we apply strong branching to the disjunctions in $\mathcal{S}$, in the spirit of [6,28], to evaluate the true impact of each disjunction. Note that the computational complexity of this algorithm is dominated by the strong branching phase.

Once strong branching is performed and we know the objective function improvement at the children nodes $c^\top \bar{x}_1, c^\top \bar{x}_2$, we use

$$\gamma \min(c^\top \bar{x}_1, c^\top \bar{x}_2) + (1 - \gamma) \max(c^\top \bar{x}_1, c^\top \bar{x}_2), \tag{11}$$

with $0 \leq \gamma \leq 1$, as a measure of quality of a disjunction, attempting to increase the LP bound. This approach is not new: for instance, [2] proposes $\gamma = 5/6$ in the context of branching on elementary disjunctions.

5. Strengthening the GMI Disjunctions

GMI cuts derived from elementary disjunctions are very strong in practice; but can we do better? [4,15] experiment with cutting planes derived as GMI cuts from split disjunctions, with good results. In our framework, their procedure can be seen as a method for finding a disjunction (8) that gives rise to an intersection cut with better cut coefficients on the continuous variables. The starting disjunction $\hat{\pi}$ is then plugged into (9) as usual. Clearly, this approach is computationally more expensive: the elementary disjunctions of Section 4 can be read from the tableau with no additional cost, but finding a strong split disjunction of the form (8) is not as simple, as there is an infinite number of them.

[14] proposes the following approach, which has also been modified and enhanced for cutting plane generation in [15]. The motivating idea traces back to [4]. We look at the expression of α_j (5) for the intersection cut derived from (9):

$$\alpha_j \begin{cases} \max\left(\frac{\left\langle\sum_{i\in B_I}\hat{\pi}_i\bar{x}_i\right\rangle}{\left\langle\sum_{i\in B_I}\hat{\pi}_i\bar{a}_{ij}\right\rangle}, \frac{1-\left\langle\sum_{i\in B_I}\hat{\pi}_i\bar{x}_i\right\rangle}{1-\left\langle\sum_{i\in B_I}\hat{\pi}_i\bar{a}_{ij}\right\rangle}\right) & \text{if } j \in J_I \\ \max\left(\frac{\left\langle\sum_{i\in B_I}\hat{\pi}_i\bar{x}_i\right\rangle}{\sum_{i\in B_I}\hat{\pi}_i\bar{a}_{ij}}, \frac{1-\left\langle\sum_{i\in B_I}\hat{\pi}_i\bar{x}_i\right\rangle}{-\sum_{i\in B_I}\hat{\pi}_i\bar{a}_{ij}}\right) & \text{if } j \in J_C \end{cases} \tag{12}$$

where $\alpha_j = \infty$ if its denominator is zero. A larger α_j means a smaller cut coefficient in (6), hence a stronger cut, as can be seen from (7); and by Proposition 3.1, we argue that a disjunction with large α_j will be strong for branching as well. Therefore, we study a method for increasing α_j by acting on $\hat{\pi}$.

It seems difficult to optimize α_j for $j \in J_I$ because both terms of the fraction are nonlinear. Furthermore, for $j \in N_I$, α_j is always at least 1, independent of the choice of $\hat{\pi}$. For $j \in J_C$, α_j can be smaller than 1, therefore we concentrate on trying to improve these α_j. From (12) we see that the denominator of α_j for $j \in J_C$ is a linear function of $\hat{\pi}$, whereas the numerator is a nonlinear function of $\hat{\pi}$ and is always between 0 and 1. For this reason we attempt to minimize the denominator, i.e. $\sum_{i\in B_I}\hat{\pi}_i\bar{a}_{ij}$ for $j \in J_C$, over integral vectors $\hat{\pi}$. More specifically, we would like to minimize $\|\tilde{d}\|$, where

$$\tilde{d} = (\sum_{i\in B_I}\hat{\pi}_i\bar{a}_{ij})_{j\in J_C} \ . \tag{13}$$

Since we try to improve the disjunction by looking at the cut coefficients on the continuous variables, the method described in this section is only suitable for mixed-integer instances.

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

$$\min_{\hat{\pi}\in\mathbb{Z}^{|B_I|}\setminus\{0\}} \|\sum_{i\in B_I}\hat{\pi}_i d_i\|. \tag{14}$$

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 [3].

[4] proposes a heuristic for (14) 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.

The idea of [14] 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:

$$\min_{\hat{\pi}^k \in \mathbb{R}^{|R_k|}, \hat{\pi}^k_k = 1} \| \sum_{i \in R_k} \hat{\pi}^k_i d_i \|, \tag{15}$$

and then rounding the coefficients $\hat{\pi}^k_i$ to the nearest integer $\lfloor \hat{\pi}^k_i \rceil$. There are several reasons for imposing $\hat{\pi}^k_k = 1$. One reason is that not only do we want to find a short vector, but it is also important to find a vector $\hat{\pi}^k$ with small norm: 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 $\hat{\pi}$: in this space, disjunctions that cut off a larger volume have a small $\|\hat{\pi}\|$. We will come back to this issue in Section 6. 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 $\| \sum_{i \in R_k} \hat{\pi}^k_i d_i \|$ with respect to $\hat{\pi}^k_i$ for all i, we obtain an $|R_k| \times |R_k|$ linear system that yields the optimal (continuous) solution.

Once these linear systems are solved and we have the optimal coefficients $\hat{\pi}^k \in \mathbb{R}^{|R_k|}$ for all $k \in \{1, \dots, |B_I|\}$, we round them to the nearest integer. Then, we consider the norm of $\sum_{i \in R_k} \lfloor \hat{\pi}^k_i \rceil d_i$. If $\| \sum_{i \in R_k} \lfloor \hat{\pi}^k_i \rceil d_i \| < \|d_k\|$, then we have an improvement with respect to the original row of the simplex tableau; in this case, we use

$$\sum_{i \in R_k} \hat{\pi}^k_i x_i = \sum_{i \in R_k} \hat{\pi}^k_i \bar{x}_i - \sum_{j \in J} \sum_{i \in R_k} \hat{\pi}^k_i \bar{a}_{ij} x_j, \tag{16}$$

instead of row $\bar{a}_k$ in order to compute a GMI disjunction, and consider the possibly improved disjunction for branching.

It is natural to ask how to choose $R_k \subset B_I$. Although using $R_k = B_I$ is possible, in that case two problems arise: first, the size of the linear systems may become too large, 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 $\bar{a}_{kj} = 0$ implies $\alpha_j = \infty$, i.e. the cut is strong on that variable. Therefore, we would like those coefficients that are 0 in row $\bar{a}_k$ to be left unmodified when we compute $\sum_{j \in R_k} \lfloor \hat{\pi}^k_j \rceil \bar{a}_j$.

6. On Non-Dominated Disjunctions

Although solving the shortest vector problem (14) 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 equal to $1/\|\lambda\|$ as can be seen from (9). Therefore, in this space, disjunctions that cut off a larger volume have a small $\|\lambda\|$. We illustrate this with an example.

Example 6.1. Consider the following tableau, where x_1, x_2 are binary variables and y_1, y_2 are continuous:

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

The solution to the shortest vector problem (14) 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 $99x_1 + 98x_2 \leq 65 \vee 99x_1 + 98x_2 \geq 66$. The heuristic method of Section 5 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 the heuristic method dominates the one computed through the exact solution of the shortest vector problem. Figure 4 gives a picture of this.

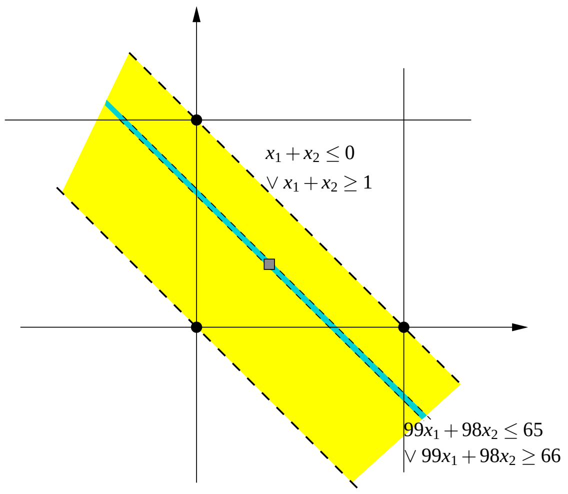


Figure 4. Representation of the disjunctions discussed in Example 6.1.

It is clear from Example 6.1 why disjunctions with small coefficients are likely to perform better. It is intuitive to think that, at least in the unit hypercube, the coefficients of “good” disjunctions will be small. However, this is not true in general. We formalize our statament.

For a polyhedron P, we say that the split disjunction $D(\pi^1, \pi_0^1)$ *dominates* $D(\pi^2, \pi_0^2)$ if $P \cap \{{\pi^1}^\top x \leq \pi_0^1\} \subseteq P \cap \{{\pi^2}^\top x \leq \pi_0^2\}$ and $P \cap \{{\pi^1}^\top x \geq \pi_0^1 + 1\} \subseteq P \cap \{{\pi^2}^\top x \geq \pi_0^2 + 1\}$, with at least one of the two inclusions being strict. The dominating disjunction is obviously to be preferred to the dominated one for branching, as it induces the same partition of the feasible integer points, while generating smaller feasible regions for the two children. Thus, we are interested in finding non-dominated disjunctions only. Do the coefficients of non-dominated disjunctions have a "nice" characterization, so that we can restrict our search to disjunctions with small norm? Unfortunately, the answer is negative in general. Even by restricting our attention to 0/1 polytopes, no polynomial bound (in the dimension n) can be given on the size of the coefficients of non-dominated disjunctions.

Proposition 6.2. The size of the coefficients of non-dominated disjunctions for 0/1 polytopes of dimension n cannot be polynomially bounded in n.

Proof. It is known [29] that the largest integer coefficient in the facet description of a full-dimensional 0/1 polytope can be exponential in n. Let $a^\top x \geq b$ be the hyperplane, which we can assume to have all integer coefficients, describing such a facet with a large coefficient. Consider the polytope defined by $P = \{x \in [0,1]^n | a^\top x \geq b, a^\top \leq b + 0.5\}$. The disjunction $D(\pi, \pi_0)$ with $\pi = a, \pi_0 = b$ is non-dominated, and in fact gives the convex hull of the integer points in one branching step. However, its largest coefficient has size exponential in n. $\square$

Therefore, even though in low dimension non-dominated disjunctions have small integer coefficients, in general there is no hope of finding a nice characterization of their coefficients. The method described in Section 5 tries to generate disjunctions with small coefficients heuristically, following the intuition of Example 6.1.

7. Computational experiments

The ideas proposed in [14,19] were tested in a Branch-and-Bound framework implemented on top of Cplex [18]. The test set consists of all instances in `MIPLIB2.0`, `MIPLIB3` and `MIPLIB2003`, excluding those that can be solved in less than 50 nodes by branching on simple disjunctions, and those for which less than 50 nodes can be processed in an hour. Also removed were the instances with zero integrality gap, which leaves 84 instances.

We report tests with three branching algorithms:

- Branching on single variables (Simple Disjunctions, SD);
- Branching on the disjunctions defining the GMI cuts at the optimal LP basis (GMI Disjunctions, GD);
- Branching on the disjunctions defining the GMI cuts after the strengthening procedure described in Section 5 (Improved GMI Disjunctions, IGD).

In order to evaluate the effect of branching on split disjunctions, we focus primarily on the integrality gap closed by branching. An additional important factor is the number of infeasible children which are created by branching: in fact, if one of the two sides of the branching disjunction is infeasible, the number of nodes in the enumeration tree does not grow. This can be seen as adding a cutting plane (i.e. the feasible side of the disjunction) to the current node. In case such a disjunction a discovered, it is always preferred to the ones that create two children. Note that if both sides of the disjunction are infeasible, the node is infeasible.

7.1. Branching for Eight Levels

In this experiment, we branch at the top eight levels of the Branch-and-Bound tree, and compare the resulting gap closed. At each node, for the SD algorithm we consider all fractional integer variables for branching, whereas for GD we consider all simple GMI disjunctions. Note that the number of candidate branching objects is the same for both SD and GD. We set $\gamma = 5/6$ in (11) as suggested by [2], trying to increase the LP bound in both children nodes. In this experiment, GD performs better, mainly due to the larger gap closed by branching on split disjunctions. We observe an interesting secondary effect: branching on GMI disjunctions tends to produce more infeasible children, which additionally decreases the amount of enumeration. We record this phenomenon by counting the number of active nodes at the ninth level.

Table 1 contains a summary of the results. We report average values, and the number of times that one method is better than the other according to the comparison criterion.

Percentage gap closed	average (# better)
Simple disjunctions (SD):	32.1% (20)
GMI disjunctions (GD):	41.7% (48)

Active nodes at level 9	average (# better)
Simple disjunctions (SD):	114.6 (16)
GMI disjunctions (GD):	66.7 (53)

Gap closed and active nodes together	# better
Simple disjunctions (SD):	6
GMI disjunctions (GD):	45

Table 1. Comparison of SD and GD after eight levels of branching. Branch-and-Bound.

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

A graphical representation of the gap closed by SD and GD is shown in Figure 5.A. In the figure, dots correspond to test instances. The gap closed by SD is shown on the abscissa while that closed by GD is shown on the ordinate. The diagonal line represents

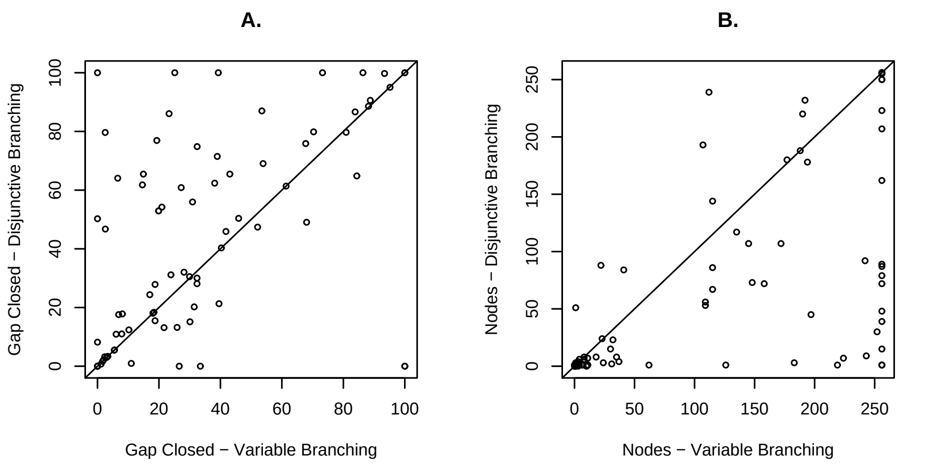


Figure 5. A. Gap closed (in percentage) after eight levels of branching: GD vs. SD. B. Number of active nodes after eight levels of branching: GD vs. SD. Every data point represents a test instance.

equality in the gap closed by both methods. We observe that most points lie in the upper-left triangle, corresponding to “GD outperforms SD.” Furthermore, most of the points that lie in the lower-right triangle are close to the diagonal line – there are few cases in which SD outperforms GD significantly.

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

The difference in the performance is best seen graphically. In Figure 5.B, we plot the number of active nodes at level nine produced by GD vs. that produced by SD. Not only do most of the points lie below the equality line but many of them reside in the bottom-right corner, corresponding to a significant difference in the number of nodes. On the other hand, out of the 16 instances for which SD outperforms GD, only eight lie visibly far from the equality line.

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

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

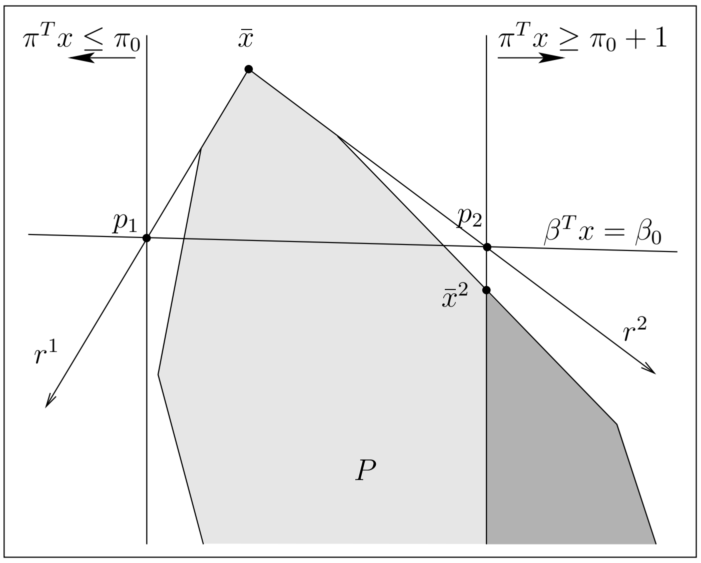


Figure 6. Disjunction with only one feasible child.

The combination of a larger improvement in the gap and a smaller number of active nodes is a very desirable effect and it deserves more attention. Branching on a disjunction that generates only one feasible child is equivalent to adding a single cut to the formulation. One may argue that this cut would be added by a branch-and-cut algorithm anyway. This is true in some cases but in others the disjunction inequality is stronger than the corresponding GMI cut. Figure 6 is an example. The cut generation procedure considers the polyhedral cone pointed at $\bar{x}$, relaxing some of the constraints defining P, and generates the intersection cut $\beta^T x \le \beta_0$. But it cannot detect the fact that one of the feasible sets of the children is empty. (Here, $P \cap \{x \in \mathbb{R}^n : \pi^T x \le \pi_0\}$.) When branching on $D(\pi, \pi_0)$, we essentially add the cut $\pi^T x \ge \pi_0 + 1$, which is stronger than $\beta^T x \le \beta_0$.

Consequently, branching on a split disjunction that generates only one child can be viewed as strengthening the underlying intersection cut. Thus, branching on a split disjunction cannot be substituted by adding the corresponding intersection cut even when one of the disjunctive sets is empty. When both disjunctive sets are non-empty, branching on a split disjunction can still close more gap than the corresponding cut, as we showed in Section 3.

We do not consider branching on split disjunctions a substitute for cutting planes. The procedure comes into play when Branch-and-Cut decides to start branching. It is important to note that the observed good effects of branching on split disjunctions are not neutralized by adding cuts. We repeat the above experiment in a Cut-and-Branch framework where we add ten rounds of GMI cuts, MIR cuts, and knapsack cover cuts. As expected, aggressive cut generation closes a significant amount of gap (63% on average), leaving less work for the branching phase. As a result, the amount of gap closed by branching on the top eight levels is smaller and the difference between the two methods is smaller. Nevertheless, the mutual relation in performance is preserved, as seen in Table 2.

Percentage gap closed average (# better)	
Simple disjunctions (SD):	5.6% (11)
GMI disjunctions (GD):	7.4% (52)

Active nodes at level 9 average (# better)	
Simple disjunctions (SD):	107.6 (23)
GMI disjunctions (GD):	81.6 (44)

Gap closed and active nodes together # better	
Simple disjunctions (SD):	6
GMI disjunctions (GD):	39

Table 2. Comparison of SD and GD after eight levels of branching. Cut-and-branch.

7.2. Effect of the Disjunction Strengthening Procedure

In this section we want to evaluate the impact of the disjunction improvement procedure on the branching phase. We have already seen that GD is able to outperform SD in several respects. We want to see if the same holds true for IGD. Therefore, we design a similar experiment: we branch for 1000 nodes, and compare the integrality gap closed by each method (or the number of nodes, for instances solved to optimality in less than 1000 nodes). In this experiment, generating fewer feasible nodes is clearly an advantage, as it allows to progress further in the tree. Note that IGD can only be applied on mixed-integer instances, because the disjunction strengthening procedure requires the presence of continuous variable. Thus, for this experiment the test set consists of the 57 instances with more than one continuous variable only.

Since we are focusing on closing more integrality gap, in this experiment we set $\gamma = 1$ in (11). Besides, to speed up the computations, we do not apply strong branching to all possible branching disjunctions, but only to the 10 most promising ones. This setting is meant to mimick more closely what is done in commercial software, since strong branching can be very expensive. This allows us to better evaluate the computational overhead introduced by branching on split disjunctions. The most promising disjunctions are chosen as the 10 variables with larges fractional variable (for SD), or as the split disjunctions with largest distance cut off by the corresponding intersection cut (for GD and IGD).

For IGD, after some preliminary testing, we decided to set $M_{|R_k|} = 50$, i.e. we combine at most 50 rows during the disjunction streanghtening phase.

Table 3 shows that the increase in the gap closed per node by branching on GMI disjunctions is large compared to branching on single variables. Besides, the IGD method

Number of solved instances	
Simple disjunctions (SD):	15
GMI disjunctions (GD):	20
Improved GMI disjunctions (IGD):	20

Average number of nodes on instances solved by all methods	
Simple disjunctions (SD):	125.6
GMI disjunctions (GD):	98.1
Improved GMI disjunctions (IGD):	75.3

Average CPU time [sec] on instances solved by all methods	
Simple disjunctions (SD):	2.53
GMI disjunctions (GD):	5.23
Improved GMI disjunctions (IGD):	4.79

Average gap closed on instances not solved by any method	
Simple disjunctions (SD):	9.02%
GMI disjunctions (GD):	12.99%
Improved GMI disjunctions (IGD):	13.30%

Number of instances with largest closed gap (at least as much as the other methods)	
Simple disjunctions (SD):	34
GMI disjunctions (GD):	33
Improved GMI disjunctions (IGD):	36

Table 3. Results on mixed-integer instances after 1000 solved nodes

seems to be on average superior in all respects to the two other 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 57.

On the instances solved by all methods, SD is roughly twice as fast as 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 split disjunctions introduces a significant computational overhead at each node with respect to branching on simple disjunctions. The average time spent per node by the three methods, recorded as the geometric mean of the average time spent per node over all the instances, is as follows:

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

Therefore, the most evident drawback of branching on split 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 GMI cut associated with each row of the simplex tableau, and the reduction step proposed in Section 5 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. Additionally, generating the GMI disjunctions requires the computation of the optimal simplex tableau, which is not necessary (and is typically not carried out) when branching on single variables. The second reason is that, by branching on GMI 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.

In summary, computational experience with branching on split disjunctions shows that the size of the enumeration tree can be reduced by a factor of two or more on average. This is not quite sufficient to compensate for the increased computing time per node. A possibility for overcoming this drawback is to combine branching on single variables and on split disjunctions, using the latter disjunctions only when the gap closed is significantly greater.

References

[1] K. Aardal, R. E. Bixby, C. A. J. Hurkens, A. K. Lenstra, and J. W. Smeltink. Market split and basis reduction: Towards a solution of the Cornuéjols-Dawande instances. *INFORMS Journal on Computing*, 12(3):192–202, 2000.

[2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[3] M. Ajtai. 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.

[4] K. Andersen, G. Cornuéjols, and Y. Li. Reduce-and-split cuts: Improving the performance of mixed integer Gomory cuts. *Management Science*, 51(11):1720–1732, 2005.

[5] K. Andersen, G. Cornuéjols, and Y. Li. Split closure and intersection cuts. *Mathematical Programming A*, 102(3):457–493, 2005.

[6] D. Applegate, R. E. Bixby, V. Chvàtal, and W. Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS, 1995.

[7] E. Balas. Intersection cuts - a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.

[8] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[9] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

[10] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

[11] E. Balas and M. Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer gomory cuts for 0-1 programming. *Mathematical Programming*, 94(2-3):221–245, 2003.

[12] W. Cook, R. Kannan, and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.

[13] G. Cornuéjols and M. Dawande. A class of hard small 0-1 programs. In R. E. Bixby and E. A. Boyd, editors, *Proceedings of the 6th IPCO Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 284–293. Springer-Verlag, Berlin, 1998.

[14] G. Cornuéjols, L. Liberti, and G. Nannicini. Improved strategies for branching on general disjunctions. *Mathematical Programming A*, 2009. Published online.

[15] G. Cornuéjols and G. Nannicini. Reduce-and-split revisited: efficient generation of split cuts for mixed-integer linear programs. Technical report, Tepper School of Business, Carnegie Mellon University, April 2010.

[16] R. E. Gomory. An algorithm for the mixed-integer problem. Technical Report RM-2597, RAND Corporation, 1960.

[17] M. Grotschel, L. Lovász, and A. Schrijver. Progress in combinatorial optimization. In *Geometric methods in Combinatorial Optimization*, pages 167–183. Academic Press, Toronto, 1984.

[18] ILOG. *ILOG CPLEX 8.0 User's Manual*. ILOG S.A., Gentilly, France, 2002.

[19] M. Karamanov and G. Cornuéjols. Branching on general disjunctions. *Mathematical Programming A*, 2009. Published online.

[20] B. Krishnamoorthy and G. Pataki. Column basis reduction and decomposable knapsack problems. *Discrete Optimization*, 6(3):242–270, 2009.

[21] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[22] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 4(261):515–534, 1982.

[23] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[24] L. Lovász and H. E. Scarf. The generalized basis reduction algorithm. *Mathematics of Operations Research*, 17(3):751–764, 1992.

[25] A. Mahajan and T. K. Ralphs. Experiments with Branching using General Disjunctions. In *Proceedings of the Eleventh INFORMS Computing Society Meeting*, pages 101–118, 2009.

[26] A. Mahajan and T. K. Ralphs. On the Complexity of Selecting Disjunctions in Integer Programming. *SIAM Journal on Optimization*, 20(5):2181–2198, 2010.

[27] S. Mehrotra and Z. Li. On generalized branching method for mixed integer programming. Technical report, Northwestern University, Evanston, Illinois, 2004.

[28] J. Owen and S. Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear program. *Computational Optimization and Applications*, 20:159–170, 2001.

[29] G. M. Ziegler. Lectures on 0/1-polytopes. In G. Kalai and G. M. Ziegler, editors, *Polytopes – Combinatorics and Computation*, volume 29 of *DMV Seminars*, pages 1–42. Birkhäuser Basel, 2000.