Approximating the split closure

Matteo Fischetti, Domenico Salvagnin

Department of Information Engineering, University of Padova, Italy

{matteo.fischetti@unipd.it, domenico.salvagnin@unipd.it}

The split closure has been proved in practice to be a very tight approximation of the integer hull formulation of a generic mixed-integer linear program. However, exact separation procedures for optimizing over the split closure have unacceptable computing times in practice, hence many different heuristic strategies have been proposed in the last years. In this paper we present a new overall framework for approximating the split closure, that merges different ideas from the previous approaches. Computational results prove the effectiveness of the proposed procedure compared to the state of the art, showing that a good approximation of the split closure bound can be obtained with very reasonable computing times.

Key words: Mixed-Integer Programming, Gomory cuts, split cuts

1.   Introduction

In this paper, we consider a Mixed Integer Linear Program (MIP) in standard form:

$$\min cx \tag{1}$$

$$Ax = b \tag{2}$$

$$x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p} \tag{3}$$

where A is an $m \times n$ rational matrix, and b, c are rational vectors of appropriate size. The index set $J := \{1, \ldots, p\} \subseteq N := \{1, \ldots, n\}$ indicates integer variables. We will denote by P the Linear Programming (LP) relaxation of (1)-(3), obtained by dropping the integrality requirement on x_j for all $j \in J$.

The state of the art algorithm for solving problems of the form (1)-(3) is *branch-and-cut* (B&C). In the last decades, (general-purpose) cutting planes have proved to be one of the main ingredient for the practical effectiveness of the B&C algorithm, as confirmed in many computational surveys Bixby et al. (2000, 2004). Among them, one of the main cutting plane families is that of *split cuts* addressed in Cook et al. (1990). For any $\pi \in \mathbb{Z}^n$ with

$\pi_{p+1} = \ldots = \pi_n = 0$ and $\pi_0 \in \mathbb{Z}$, we define as *split* the disjunction

$$\pi x \leq \pi_0 \ \vee \ \pi x \geq \pi_0 + 1. \tag{4}$$

Any inequality which is valid for the union $P_1 \cup P_2$, where

$$P_1 = \{x \in P : \pi x \leq \pi_0\} \text{ and } P_2 = \{x \in P : \pi x \geq \pi_0 + 1\}, \tag{5}$$

is called a *split cut*. Many classes of valid inequalities studied in the literature turn out to be split cuts, among them: (strengthened) lift-and-project cuts by Balas et al. (1996); Balas and Perregaard (2003), Gomory mixed-integer (GMI) cuts by Gomory (1963), Mixed Integer Rounding (MIR) cuts by Nemhauser and Wolsey (1990), and intersection cuts from splits as studied in Balas (1971). The intersection of all split inequalities valid for a given rational polyhedron P is called the *first split closure* of P, and it is proved in Cook et al. (1990) that this intersection is again a polyhedron. As we do not consider higher rank closures, adjective "first" will be omitted in what follows.

Recent computational studies in Fischetti and Lodi (2007); Balas and Saxena (2008); Dash et al. (2010) show that the split closure is surprisingly tight—on average, it closes almost 80% of the integrality gap on the MIPLIB 3.0 instances. Unfortunately, separating or optimizing exactly over the split closure is NP-hard, and unacceptably slow in practice. Given the practical strength (and inherent difficulty) of the split closure, a lot of research has been done in the last years in devising fast and effective heuristic procedures for separating split cuts, including Marchand and Wolsey (2001); Dash and Goycoolea (2010); Andersen and Weismantel (2010); Bonami (2010); Cornuéjols and Nannicini (2011); Fischetti and Salvagnin (2011).

The present paper is organized as follows. Section 2 surveys current state-of-the-art approaches for optimizing heuristically over the split closure. In Section 3 we describe our new hybrid procedure. Various elaborations of the basic ideas along with algorithmic techniques and computational results are presented in Section 4. Some conclusions are finally drawn in Section 5.

2. Literature

According to its definition, a split cut $\gamma x \geq \gamma_0$ is defined by the choice of a split (π, π_0) and by a certificate that the cut is valid for P_1 and P_2—because we are dealing with LPs, this certificate is polynomial in size and is provided by dual LP multipliers.

As such, an exact separation procedure for split inequalities must be able to optimize simultaneously over the set of splits and dual multipliers. Unfortunately, the corresponding optimization problem is NP-hard and computationally very expensive in practice, as shown in Balas and Saxena (2008) and Dash et al. (2010).

However, the task simplifies a lot if one fixes either the split or the set of multipliers in advance. Actually, almost all separation procedures involving split (or MIR/GMI) cuts can be interpreted this way. For example, given a fixed split disjunction (π, π_0) and a point $x^* \in P$, it is possible to find a violated split cut $\gamma x \geq \gamma_0$ (or show that none exists) as a compact linear problem due to Balas (1971), that reads:

$$\min \gamma x^* - \gamma_0 \tag{6}$$

$$\gamma \geq uA - \pi u_0 \tag{7}$$

$$\gamma_0 \leq ub - \pi_0 u_0 \tag{8}$$

$$\gamma \geq vA + \pi v_0 \tag{9}$$

$$\gamma_0 \leq vb + \pi_0(v_0 + 1) \tag{10}$$

$$\langle \text{normalization condition} \rangle \tag{11}$$

$$u_0, v_0 \geq 0 \tag{12}$$

In the above LP, variables (u, u_0) certificate the validity of the cut w.r.t. P_1, while variables (v, v_0) certificate the validity of the cut w.r.t. P_2. Normalization condition (11) is needed because the feasible set of (6)-(12) is a cone.

Another (extreme) example is that of GMI cuts read from a row of a tableau. In this case the set of multipliers is determined by the corresponding row of the basis inverse, while the split disjunction is chosen with a closed form formula that depends on the coefficients of the tableau row.

We will now describe the basic ideas behind the most successful separation heuristics for split cuts. In the following, we will call "large LP" the linear programming problem defined by the original constraints plus the split cuts generated in the previous iterations, and we will denote by x^* one of its optimal vertices. In addition, we will call "small LP" the ones defining P.

2.1. Relax-and-cut

We briefly outline the basic relax-and-cut framework for GMI cuts that we introduced in Fischetti and Salvagnin (2011).

Consider a generic MIP of the form (1)-(3). For the sake of simplicity, we assume that the LP relaxation polyhedron P is bounded and nonempty, so as to guarantee the existence of an optimal solution for any linear objective function.

Optimizing over the split closure is equivalent to solving the following linear program, where the exponentially large family of (rank-1) split cuts $\gamma^i x \geq \gamma_0^i$ has been added to the constraints of the original formulation:

$$z_1 := \begin{cases} \min cx \\ x \in P \\ \gamma^i x \geq \gamma_0^i, \quad i = 1, \ldots, M \end{cases} \tag{13}$$

A standard solution approach for system above consists in dualizing the split cuts in a Lagrangian fashion, thus obtaining the Lagrangian dual problem

$$\max_{u \geq 0} \left\{ L(u) := \min\{cx + \sum_{i=1}^{M} u_i(\gamma_0^i - \gamma^i x) : x \in P\} \right\} \tag{14}$$

whose optimal value is known to coincide with z_1.

Because (14) has a piecewise-linear concave objective function, its solution can be attempted through very simple iterative procedures known as subgradient methods, or through more sophisticated and accurate schemes such as the bundle method; see e.g. Hiriart-Hurruty and Lemaréchal (1993).

In our context, the family of cuts to be dualized is far too large to be computed explicitly, so only some of its members are explicitly considered and stored, using a data structure called *cut pool*. The (initially empty) cut pool is iteratively extended by means of (rank-1) GMI cuts that are heuristically generated, on the fly, during the process of solving the Lagrangian dual problem. In particular, if the Lagrangian subproblem at iteration k is solved by the simplex method and an optimal vertex x^k of P with fractional components is found, we can just read a round of rank-1 GMI cuts from the optimal LP tableau and feed the cut pool.

Several algorithmic enhancements have been proposed in Fischetti and Salvagnin (2011) to improve and speed up the basic subgradient method when applied to (14). In particular
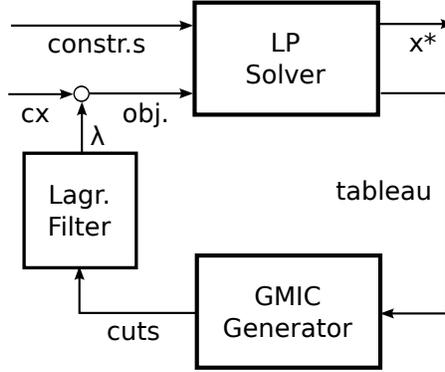
4

Figure 1: Basic relax-and-cut scheme for GMI cuts.

- recompute, from time to time, the optimal Lagrangian multipliers u_i for *all* the cuts in the current pool. This can be done quite efficiently by solving (through a standard row-generation scheme) the *large LP* defined by the original constraints plus all the cuts in the pool.

- use a very small subgradient step length parameter, in order to have more parameterized LPs where warm-start is more effective.

- drastically reduce the number of subgradient iterations.

Algorithm 1: Relax-and-cut for GMI cuts—the basic scheme.

> **input** : MIP $\equiv \min\{c^T x : x \in P,\ x_j \text{ integer } \forall j \in J\}$, number of main iterations L, subgradient iteration limit $I_{\max}$
>
> **output**: a polyhedron P' approximating the first GMI closure

1 $x^* = \texttt{LPSolve}\ (P)$
2 $pool = \texttt{ReadGMI}\ (P)$
3 $P' = P \cap \{\text{cuts in } pool\}$
4 $x^* = \texttt{LPSolve}\ (P')$
5 **if** x^*_j *is integer* $\forall j \in J$ **then return** P'
6 **for** *i=1* **to** L **do**
7      $u_0 =$ optimal LP dual vector for cuts in *pool*
8      $pool = \texttt{SubgradientRun}\ (P, pool, I_{\max})$
9      $P' = P \cap \{\text{cuts in } pool\}$
10      $x^* = \texttt{LPSolve}\ (P')$
11   **if** x^*_j *is integer* $\forall j \in J$ **then return** P'
12 **end**
13 **return** P'

The relax-and-cut GMI framework is depicted in Figure 1, while a more detailed algorithmic description in given in Algorithm 1. In the following, we will denote by main iteration (or simply iteration) the code executed between two consecutive recomputations of the optimal Lagrangian multipliers, i.e., lines 7–11 in Figure 1.

From the split closure point of view, the relax-and-cut method is sampling near-optimal Lagrangian bases of the original LP to generate rank-1 GMI cuts. Note that these bases are always primal-feasible for the original system P, and that the method is only using the disjunctions that are implicit in the derivation of the GMI cuts from the rows of a given tableau.

2.2.   Lift-and-Project

Lift-and-project cuts are obtained by solving system (6)-(12) for elementary splits (i.e., $\pi x = x_k$, for some $k \in J$). Cut generating linear programs (CGLPs) have been studied extensively for this purpose, and many different normalization conditions have been proposed in the literature; see Fischetti et al. (2011) for a recent paper on this topic. We consider the trivial normalization $u_0 + v_0 = 1$, and report the properties of the corresponding CGLP for the elementary split $x_k \leq \lfloor x_k^* \rfloor \vee x_k \geq \lceil x_k^* \rceil$, where $x^* \in P$ is the point to separate and x_k^* is fractional.

Taking the linear programming dual of (6)-(12) with normalization $u_0 + v_0 = 1$, the CGLP can be simplified and restated in the following equivalent "primal" form that works on the original variable space; see e.g. Balas and Saxena (2008):

$$v^* := \max \ v = y_k - \lceil x_k^* \rceil \tag{15}$$

$$Ay = b, \tag{16}$$

$$0 \leq y \leq x^*/d^* \tag{17}$$

where $d^* = \pi x^* - \pi_0 = x_k^* - \lfloor x_k^* \rfloor \in (0, 1)$, and a violated disjunctive cut (for the given disjunction) exists if and only if $v^* < 0$. In case x_j happens to be free for some $j \in N$, the associated y_j becomes free as well as both its bounds vanish from the CGLP model.

The model above has a neat geometrical interpretation. Indeed, the order of business of separation is to determine whether $x^* \in conv(P_1 \cup P_2)$, which is in turn equivalent to the existence of two points $z \in P_1$ and $y \in P_2$ and of a multiplier $\lambda \in [0, 1]$ such that $x^* = \lambda y + (1 - \lambda)z$; see Figure 2 for an illustration. Note that one can w.l.o.g. assume that
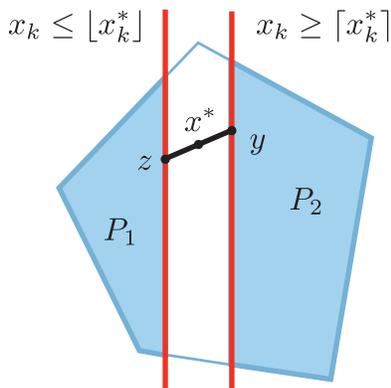
Figure 2: Geometric interpretation of CGLP (15)-(17).

y and z satisfy the two sides of the split at equality, i.e., $y_k = \lceil x_k^* \rceil$ and $z_k = \lfloor x_k^* \rfloor$, hence $x_k^* = \lfloor x_k^* \rfloor + d^* = \delta^* y_k + (1 - \delta^*) z_k$ and $\lambda = \delta^*$ is easily computed. The existence of $y \in P_2$ is clearly equivalent to (16) and $y, v \geq 0$. As to z, it is enough to observe that condition $Az = b$ poses no problem, as both x^* and y satisfy the equation system $Ax = b$, whereas $z = (x^* - \delta^* y)/(1 - \delta^*) \geq 0$ is due to the variable upper bounds in (17). Therefore, if the CGLP is able to find a solution y with $v \geq 0$, then we have a certificate that the point x^* belongs to $conv(P_1 \cup P_2)$ and cannot be separated. Otherwise we can derive a violated disjunctive cut by duality, see Balas and Saxena (2008) and Bonami (2010) for details.

The CGLP (15)-(17) has same handy properties, namely:

- It has the same size and structure of the original model—indeed we just added some upper bounds on the variables and changed the objective function. In addition, because the constraint matrix A is the same, the set of bases of (15)-(17) is the same as in (1)-(3). Note however that a same basis B can be primal feasible in (15)-(17) but not in (1)-(3), because the bounds are different.

- Because of (17), it works automatically on the support of x^* (which may not necessarily be a vertex of P).

- It is structurally very similar to the LP relaxation of the MIP at hand, and can easily be reoptimized by using the optimal LP-relaxation basis as an initial warm start.

The following result, due to Andersen et al. (2005b), plays an important role in understanding the correspondence between split and GMI cuts; a new constructive (very short)

proof is provided, that uses a recent result of Bonami (2010).

Theorem 1. *A point $x^* \in P$ not belonging to the split closure can always be separated by a GMI cut derived from a combination with integral coefficients of the rows of the tableau associated with a (not necessarily feasible) basis B of A.*

Proof. As $x^* \in P$ does not belong to the split closure, there exists a split (π, π_0) from which a disjunctive cut violated by x^* can be derived. Given (π, π_0), we show how to determine the basis B of A and the integer tableau-row combination producing the claimed violated GMI. We first transform the disjunction into an elementary one by introducing an additional free and integer variable $x_{n+1} = \pi x$. The resulting CGLP then reads

$$v^* := \max \ y_{n+1} - \lceil \pi x^* \rceil \tag{18}$$

$$Ay = b \tag{19}$$

$$-\pi y + y_{n+1} = 0 \tag{20}$$

$$0 \leq y \leq x^*/d^* \tag{21}$$

$$y_{n+1} \text{ free} \tag{22}$$

and has an optimal value $v^* < 0$ by hypothesis. Then let B' be an optimal basis of the above CGLP, with y_{n+1} free and hence basic, and y^* be the associated optimal vertex. Observe that v^* must be fractional, as choosing $y = x^*$ with $y_{n+1} = \pi x^*$ yields a feasible CGLP solution of value $\pi x^* - \lceil \pi x^* \rceil = -(1 - \delta^*) > -1$, i.e., $v^* \in (-1, 0)$. Hence y^*_{n+1} is fractional as well, and one can generate a GMI cut (in its fractional form) $\alpha y \geq \alpha_0$ from the tableau row associated B' where y_{n+1} is basic, with $\alpha_{n+1} = 0$. It was shown in Bonami (2010), Theorem 10, that the strengthened disjunctive cut derived from the above CGLP is equivalent to the GMI cut $\alpha x \geq \alpha_0$, hence $\alpha x \geq \alpha_0$ is valid for $conv(P_1 \cup P_2)$ and violated by x^*. It then remains to be shown that $\alpha x \geq \alpha_0$ is in fact a GMI cut derived from a combination with integral coefficients of the rows of the tableau associated with a (non necessarily feasible) basis B of A. To this end, observe that the structure of B' is:

$$\begin{bmatrix} B & 0 \\ -\pi_B & 1 \end{bmatrix} \tag{23}$$

where π_B denotes the projection of π on the set of basic variables, and B is a basis of A. The inverse of B' can be stated as

$$\begin{bmatrix} B^{-1} & 0 \\ \pi_B B^{-1} & 1 \end{bmatrix} \tag{24}$$

hence the tableau row where y_{n+1} is basic reads

$$\begin{bmatrix} \pi_B B^{-1} & 1 \end{bmatrix} \begin{bmatrix} A & 0 \\ -\pi & 1 \end{bmatrix} \begin{bmatrix} y \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} \pi_B B^{-1} & 1 \end{bmatrix} \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{25}$$

i.e.

$$\pi_B B^{-1} A y - \pi y + y_{n+1} = \pi_B B^{-1} b$$

which is nothing but the combination (with integral coefficients π_B) of the rows of the tableau associated with B, plus the equation $y_{n+1} - \pi y = 0$ with all-integer coefficients that does not affect the GMI cut derived from that row. This concludes the proof. $\qquad \square$

From the split closure point of view, strengthened lift-and-project cuts enrich the set of cuts that can be produced by the relax-and-cut procedure by considering also primal infeasible bases and, if non-elementary disjunctions are used, also taking combinations of tableau rows before deriving the GMI cuts.

2.3. Reduce-and-split and aggregation heuristics

The theorem at the end of previous section highlighted the importance of taking combinations (with integral coefficients) of tableau rows before deriving the GMI cuts. Such kind of combination is needed in general to actually achieve the split closure—just considering all GMI cuts from all feasible and infeasible bases of P, which is called the lift-and-project closure, is *not* enough.

Row aggregation itself is a very broad topic and is common to many cutting plane separators. Aggregating constraints yields a single-row relaxation, whose structure is typically much simpler than that of the original problem, thus allowing for a simpler derivation of cutting planes. Many standard procedures fall into this broad category, notably the MIR heuristic of Marchand and Wolsey (2001), which is one of the most successful cutting plane heuristic for general MIPs.

If we restrict ourselves to aggregations of tableau rows with integral coefficients, then the most successful heuristic is reduce-and-split by Andersen et al. (2005a). The idea behind this method is to perform row aggregation before deriving the GMI cut, in order to have stronger coefficients on certain "sensitive" variables. In the original scheme, the set S of sensitive variables is made of the continuous variables. The rationale behind this choice is that continuous variables are given the "worst" coefficients by the GMI formula: while integer variables always get a coefficient between 0 and 1, the same cannot be said for continuous
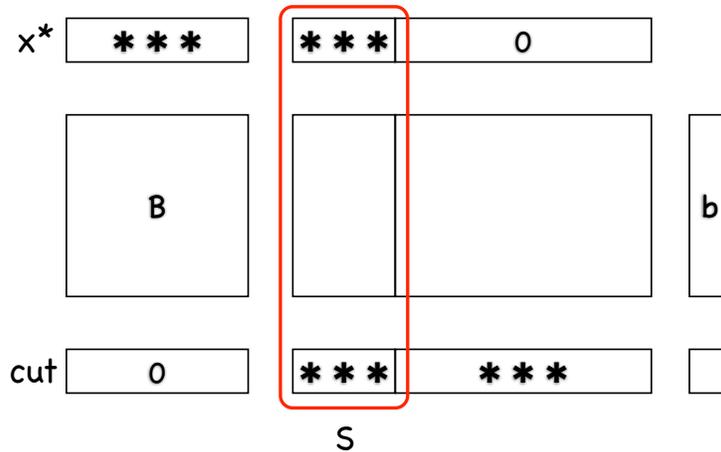
Figure 3: Variables in S are responsible for not cutting x^*.

variables. For this reason, looking for a combination where the coefficients of the continuous variables are as small as possible is likely to give a stronger cut.

If the point to separate, x^*, is a vertex of the polyhedron that we are using to generate GMI cuts, then the definition above is quite good and reasonable. However, if this is not the case, then a different choice is more natural. In particular, if we are using the tableau associated with basis B, then it makes sense to define S as the set of variables in the support of x^* that are *not* basic with respect to B. With this definition, variables in S are the ones responsible for not being able to cut the point (see Figure 3). In addition, if we look for integer multipliers such that the aggregated constraint has zero coefficients for all $x_j \in S$, we get the maximally violated cuts studied in Caprara et al. (2000) and Andersen and Weismantel (2010).

Different implementations of reduce-and-split differ in the algorithm used for computing the aggregation multipliers. The original method by Andersen et al. (2005a) uses a greedy heuristic, adding one tableau row the time and looking at each step for the integer multiplier that minimizes the norm of the combination (projected on S). A more recent development by Cornuéjols and Nannicini (2011) solves instead a convex optimization problem to find continuous multipliers yielding the smaller such norm, and then rounds them to the nearest integer.

Another strategy for computing row aggregations is a brute force approach, where every possible aggregation with integer multipliers with absolute value $\leq D$ of up to k rows is tried, deriving a cut from each of them and keeping the best. Needless to say, this strategy

is very time consuming and is not viable if the range of multipliers or the maximum number of rows is not aggressively restricted. Still, this approach was used in a computational study on two-row cuts by Dey et al. (2010) by setting $k = 2$ and $D = 3$, with the interesting outcome that, for typical MIPs, the split cuts derived from these aggregations are the most effective among many other families of two-row cuts.

3. A new hybrid procedure

The methods presented in the previous section tackle split separation from a different points of view, hence they have complementary strength. While relax-and-cut turns out to be remarkably effective in dealing with a huge number of cuts and to sample GMI cuts from primal-feasible bases, it lacks the chance of using infeasible bases (as in lift-and-project) and of combining rows with integer multipliers (as in reduce-and-split).

In addition, the computational comparison of these methods done in Bonami (2010) shows that no method is clearly dominating the others, and that even a trivial combination can already provide a better approximation of the split closure within reasonable computing times. The purpose of the hybrid procedure presented in this section is therefore to combine the strengths of the different methods in a sound way, to have a fast and effective tool for approximating the split closure of a given MIP.

Our basic hybrid scheme is described in Figure 2. We decided to use the relax-and-cut scheme as backbone, and to "plug" other procedures into it. This is justified by the fact that the relax-and-cut scheme has proved to be very effective (and numerically stable) in handling a large amount of cuts, and it also turns out to be the fastest available method for big-size instances.

According to Algorithm 2, the basic relax-and-cut scheme is augmented by calling the lift-and-project separator at the end of each main iteration. In particular, just after the subgradient method has finished, we try every elementary disjunction violated by x^* and solve the corresponding CGLP.

As far as row aggregation is concerned, we decided to implement the brute force approach described in Section 2.3, computing all possible aggregations of two tableau rows with multipliers in the range $\{-2, 2\}$ and deriving a cut from the best one. Row aggregation is enabled by setting $doAggr = 1$, and is tried whenever a basis of the Lagrangian subproblem is used to derive GMI cuts.

Algorithm 2: A new hybrid scheme.

input : MIP $\equiv \min\{c^T x : x \in P, x_j \text{ integer } \forall j \in J\}$, subgradient iteration limit $I_{\max}$, aggregation flag *doAggr*

output: a polyhedron P' approximating the first GMI closure

1 $x^* = \texttt{LPSolve}\ (P)$

2 $pool = \texttt{ReadGMI}\ (P)$

3 $P' = P \cap \{\text{cuts in } pool\}$

4 $x^* = \texttt{LPSolve}\ (P')$

5 **if** x_j^* *is integer* $\forall j \in J$ **then return** P'

6 **while** *not termination condition* **do**

7 $\quad$ $x^* = \texttt{LPSolve}\ (P')$

8 $\quad$ **if** x_j^* *is integer* $\forall j \in J$ **then break**

9 $\quad$ $u_0 = $ optimal LP dual vector for cuts in *pool*

10 $\quad$ $pool = \texttt{SubgradientRun}\ (P, pool, I_{\max}, doAggr)$

11 $\quad$ **foreach** $x_j \in J$ *s.t.* $x_j^* \notin \mathbb{Z}$ **do**

12 $\quad\quad$ $\texttt{SolveCGLP}\ (x^*, x_j \leq \lfloor x_j^* \rfloor \lor x_j \geq \lceil x_j^* \rceil)$

13 $\quad\quad$ **if** *violated cut found* **then** strengthen it and add it to *pool*

14 $\quad$ **end**

15 $\quad$ $P' = P \cap \{\text{cuts in } pool\}$

16 **end**

17 **return** P'

Note that the termination condition for the main loop has been replaced and we no longer fix the maximum number of main iterations since the beginning, see the next section for details.

4. Computational results

We tested our different split-closure separators on the problem instances in MIPLIB 3.0 from Bixby et al. (1998) and MIPLIB 2003 from Achterberg et al. (2006). Following Dash and Goycoolea (2010), we omitted all instances where there is no improvement after one round of GMI cuts read from the optimal tableau, or where no integer solution is known. In the end, we were left with 54 instances from MIPLIB 3.0 and with 20 instances from MIPLIB 2003. To improve readability, we report the outcome of our experiments through average figures only. As far as running times are concerned, we always report shifted geometric means, with a shift of 0.1, while we report arithmetic means for the amount of gap closed.

We implemented our codes in C++, using IBM ILOG Cplex 12.2 as black box LP solver. All tests have been performed on a PC with an Intel Core i5 CPU running at 2.66GHz, with

8GB of RAM (only one CPU was used by each process). Every method was given a time limit of 1 hour per instance.

As far as the GMI cut generation is concerned, for a given LP basis we try to generate a GMI cut from every row where the corresponding basic variable has a fractionality of at least 10^{-3}. The cut is however discarded if its final dynamism, i.e., the ratio between the greatest and smallest absolute value of the cut coefficients, is greater than 10^{10}. All the generated GMI cuts are stored in our cut pool. As to the large LP solution, we implemented the cut-pool selection strategies described in Achterberg (2007) and in Andreello et al. (2007), where cutting planes are selected according to their efficacy, i.e., the ratio between their violation and their norm, with a minimum of 10^{-4}, and their mutual orthogonality, with a minimum of 0.1 for the cosine between any two cuts. If the instance at hand has a maximum dynamism in the original constraints greater than 10^7, then we consider it as numerically challenging, and we lower the minimum violation to 10^{-4} and the minimum efficacy to 10^{-5}. Our pool size is limited to 1GB of RAM and can hold a number of cuts which is at most $10\times$ the number of original constraints: whenever one of the two limits is reached, we perform a purge operation based on cut efficacy (with respect to the last large LP solution x^* available) and cut age.

4.1. Implementing a GMI separator

GMI cuts are well-known for being among the simplest and fastest MIP cuts to derive. Indeed, given a tableau in standard form, a round of GMI cuts can be obtained by simply applying a closed-form formula to its rows. In practice, however, GMI separation is not so trivial, for various reasons:

- Modern solvers implement the revised simplex method, whose main feature is to *not* compute the whole tableau for carrying on simplex iterations. Thus, it may happen that generating the whole tableau is more expensive than optimizing the problem itself (even more so if we want to generate a new round cuts after a fast primal reoptimization, as in relax-and-cut).

- Modern solvers implement simplex with bounded variables. Thus, the GMI separator must deal with variables with arbitrary bounds. In addition, even if slack variables are internally added to the problem, the user expects GMI cuts to be expressed in the space of structural variables.

- Special care must be taken for preventing numerical issues and avoiding the generation of invalid cutting planes.

Our GMI separator is implemented as follows. Consider an LP in the general form

$$\min cx$$
$$Ax + Is = b$$
$$l \le x \le u$$
$$x \in \mathbb{R}^n \quad s \in \mathbb{R}_+^m$$

where variables s are the slacks of the constraints (if the original constraint is already an equality, the corresponding slack is fixed to zero). Let B be the chosen basis and x^* be the point to separate. Our GMI separator follows the following steps:

1. Force a basis refactor in order to improve numerical accuracy.

2. Get the i-th row ρ of B^{-1}

3. Compute the tableau row $\alpha x + \rho s = \beta$, where $\alpha = \rho A$ and $\beta = \rho b$. Note that, although tableau rows are in general not quite as sparse as the original constraints, still they are not usually fully dense. So we use standard tricks in order to work directly on the support of the row, without initializing a fully dense vector. In particular, whenever we add a row to the current combination, we mark the indexes of the variables in the support of the row as being part of the support of the combination. If the tableau row gets too dense, we discard it. We also discard the row if an integer free variables gets a non-integer coefficient, or if a continuous free variable gets a non-zero coefficient (it is impossible to derive a cut in these cases).

4. Shift and complement variables, in order to deal with nonnegative variables only. With bounded variables there is a degree of freedom on which bound to pick: we implemented a greedy heuristic that chooses, for each variable x_j, the bound b_j which is closer to x_j^*, much in the spirit of MIR heuristic implementations in Wolter (2006). Given a point to separate, this choice is shared by all cuts derived from the same tableau, so it is possible to perform computations such as, e.g., right-hand-side shifting, only once.

5. Apply the GMI formula to the coefficients in the support of the row. Note that when we read the model we try to detect whether some slack variables can be upgraded to an integral type, and we use this information in this step to get stronger coefficients.

6. Undo variable shifts and complements.

7. Back-substitute slacks in order to have a cut expressed in the space of structural variables only. Note that we do not consider cancelations when we compute the support of the row, hence this substitution cannot create fill-in and the support is still valid.

8. If the resulting cut is too dense or its dynamism is too high, we discard it.

4.2. Relax-and-cut improvements

The proof-of-concept implementations of the relax-and-cut procedure reported in Fischetti and Salvagnin (2011), namely `fast` and `faster`, followed a quite static and inflexible termination policy. In particular, the maximum number of iterations (L in Algorithm 1) was fixed to 10 from the beginning, regardless of the performance of the method on the instance at hand. This has two main drawbacks:

- The method cannot stop earlier in case of tailing off. It can stop only if no new violated cuts were found in the last iteration.

- The method cannot continue past the iteration limit even if it is still performing well.

To overcome these issues, we replaced the static termination policy of the original implementations with a new dynamic strategy, which is based on the performance of the method in the last iterations. In particular, the strategy implements a tailing off detection algorithm by comparing the increase in the lower bound achieved in the last iteration to the initial integrality gap. If the relative improvement is below a given threshold (10^{-3} in our code) for 3 consecutive iterations, then the method will stop, otherwise it will continue (until a maximum of 100 iterations are performed, which was however never reached in our testbed).

In addition to the above, we implemented the following tweaks to speed up the method without compromising its efficacy:

- We limited the number of GMI cuts read from a single basis to 200.

- We limited the number of primal simplex iterations done in a single subgradient iteration to 1000.

- We generated a GMI cut from a row of the tableau only if it is violated by x^*. Note that we can check whether a GMI cut will be violated by x^* before having computed the

whole cut, by just computing the right-hand side and the cut coefficients for variables in S only. It is well known that this set of variables is usually much smaller than the non-basic one, see for example Dash and Goycoolea (2010).

Table 1 reports the cumulative results of our new relax-and-cut implementation `rc` compared to `fast` and `faster`. The new procedure turns out to be significantly faster than our previous implementations, while closing approximately the some amount of gap as `fast` (and significantly more for the harder MIPLIB 2003 testbed).

Table 1: Average gap closed and computing times original and improved versions of relax-and-cut.

	MIPLIB 3.0		MIPLIB 2003	
method	cl.gap	time (s)	cl.gap	time (s)
fast	60.3%	2.50	44.2%	48.08
faster	59.4%	1.50	41.5%	28.08
rc	59.0%	0.73	48.2%	13.94

4.3. Lift-and-project

Following the description in Bonami (2010), we implemented a basic cutting plane scheme based on lift-and-project separation. The main purpose of this implementation is *not* to have a whole lift-and-project cutting plane algorithm, but just a reasonable implementation of the lift-and-project separator, to be plugged into our hybrid scheme. As such, we did not implement all the features described in Bonami (2010), but only those concerning the separator.

The basic scheme is described in Algorithm 3. At each iteration of the cutting plane algorithm, the current relaxation is solved and an optimal vertex x^* is found. The CGLP is built and solved for all elementary disjunctions corresponding to fractional values x_j^* for $j \in J$. Whenever a violated cut is found, it is added to the current formulation. As far as strengthening is concerned, we decided not to implement the procedure of Balas and Jeroslow (1980), but rather to exploit the equivalence between the strengthened lift-and-project cut and the GMI cut read from an appropriate tableau row, as in Bonami (2010). It is worth noting that such equivalence result is usually stated for LPs in standard form, which is not

---

Algorithm 3: Lift-and-project—the basic scheme.

---

input : MIP $\equiv \min\{c^T x : x \in P, \ x_j \text{ integer } \forall j \in J\}$
output: a polyhedron P' approximating the first GMI closure

1 $P' = P$
2 **while** *not termination condition* **do**
3 $\quad$ $x^* = $ LPSolve (P')
4 $\quad$ **if** x_j^* *is integer* $\forall j \in J$ **then break**
5 $\quad$ $C = \emptyset$
6 $\quad$ **foreach** $x_j \in J$ *s.t.* $x_j^* \notin \mathbb{Z}$ **do**
7 $\quad\quad$ SolveCGLP $(x^*, x_j \leq \lfloor x_j^* \rfloor \lor x_j \geq \lceil x_j^* \rceil)$
8 $\quad\quad$ **if** *violated cut found* **then** strengthen and add it to C
9 $\quad$ **end**
10 $\quad$ **if** $C = \emptyset$ **then break**
11 $\quad$ $P' = P' \cap \{\text{cuts in } C\}$
12 **end**
13 **return** P'

---

the computational form used by simplex implementations. However, it can be shown that the equivalence still holds for the general bounded simplex, provided that the tableau row used to derive the GMI cut includes the upper bound constraints on the variables (weighed by their corresponding dual multipliers) in the combination.

The algorithm stops only if any of the following three conditions is satisfied:

1. the point x^* is feasible for the MIP;

2. no violated cut is found after having tried all elementary disjunctions;

3. a time limit of 1 hour is reached.

It was suggested in Bonami (2010) that to reduce the number of CGLPs solved at each iteration—and thus, to speedup the method—one should stick to a subset of the elementary disjunctions. In particular, one could stick to the disjunctions that yielded a cut in the previous iterations, and try the others only if this subset did not led to a violated cut. We did not implement this technique. However, we stop solving CGLPs at a given iteration whenever we have collected a reasonable number of "new" violated cuts, say 500.

It was also noted in Bonami (2010) that the order in which we solve the CGLPs at a given iteration can have a great impact on running times. In particular, sorting the disjunctions by increasing values of x_j^* seems to led to the most effective warm-start for the simplex method.

Our computational experience confirms these findings, so this technique is implemented in our code.

Table 2 compares our implementation of the lift-and-project algorithm (`L&P`) to the one in Bonami (2010). The results were obtained on roughly equivalent machines, and confirm that our implementation is competitive with the one in Bonami (2010), thus providing a reasonable starting point for our next developments.

Table 2: Comparison between our implementation of the basic L&P and the implementation in Bonami (2010).

| | | MIPLIB 3.0 | | MIPLIB 2003 | |
method	CPU	cl.gap	time (s)	cl.gap	time (s)
`L&P in Bonami (2010)`	Intel 2.93 GHz	64.4%	2.25	44.4%	121.53
`L&P (our)`	Intel 2.66 GHz	65.3%	2.00	44.8%	87.13

Starting from the basic implementation just presented, we tried to push the pure lift-and-project approach further, by considering also non elementary disjunctions. The main issue with non elementary disjunctions is that they are just too many. Even restricting the number of variables involved in the disjunction to just a few and the absolute value of the coefficients to small domains, the number of candidates is usually far too large to be tried explicitly. We still decided to try a brute force approach, yet restricting the set of possible candidates very aggressively and resorting to them only when elementary disjunctions fail. In particular, we only considered disjunctions

- involving exactly two integer basic variables that take a fractional value in x^*

- whose coefficients are in the set $\{+1, -1\}$.

It follows that we considered $O(m^2)$ non elementary disjunctions at every iteration in which the elementary ones failed. The corresponding algorithm is denoted by `L&P2`, and a comparison with `L&P` is given in Table 3. As reported, `L&P2` is able to close 5% more gap than `L&P`, although, not surprisingly, the performance hit is substantial—the slowdown is one order of magnitude in the MIPLIB 2003 testbed, on average.

An additional issue with non elementary disjunctions is that usually only a small number of them will eventually yield a violated cut, meaning that the vast majority of CGLPs are

Table 3: Average gap closed and computing times for several L&P versions.

method	MIPLIB 3.0			MIPLIB 2003		
	cl.gap	time (s)	Delta	cl.gap	time (s)	Delta
L&P	65.3%	2.00	–	44.8%	87.13	–
L&P_c	65.4%	1.98	5%	45.0%	86.14	2%
L&P2	70.2%	10.69	–	50.2%	761.64	–
L&P2_c	70.2%	8.12	44%	50.3%	508.31	38%

built and solved in vain. So it would be helpful to have a simple and fast criterion to skip some disjunctions without actually solving the corresponding CGLP. The geometrical interpretation described in Section 2.2 provides some insights. Whenever the CGLP fails, meaning that it was not possible to derive a violated cut by a given disjunction (π, π_0), we have as a by product a pair of points (y, z) that acts as a certificate that x^* cannot be cut using (π, π_0). Trivially, if we are given a different disjunction (π', π_0') and the pair (y, z) satisfies $y \in P_1$ and $z \in P_2$, then x^* cannot be cut by using (π', π_0') as well. A similar check was also developed independently by Dash et al. (2011b).

The above argument can be made slightly more powerful by noting that the pair (y, z) actually describes a line containing x^*. Starting from x^*, if moving along this line in one direction we can reach a point in P_1 and a point in P_2 in the other direction, then we still have a proof that we cannot separate x^* using the disjunction generating P_1 and P_2.

Because of the above, whenever the CGLPs fails we add the corresponding line to a pool of lines $l \in \mathbb{R}^n$, and we associate to each line a step-size interval $[t_{\min}, t_{\max}]$. This interval, which can be computed with a ratio-test procedure, describes how far a point $x^* + t\,l$ can be moved along line l without leaving P. Whenever we are given a disjunction (π, π_0) to separate x^*, we compute, for each line l in the cache, the two values

$$t_1 = \frac{\pi_0 - \pi x^*}{\pi l} \quad \text{and} \quad t_2 = \frac{\pi_0 + 1 - \pi x^*}{\pi l}$$

swapping t_1 and t_2 if $t_1 > t_2$. If the interval $[t_1, t_2]$ is contained in $[t_{\min}, t_{\max}]$, then we can skip the disjunction without building and solving the CGLP. Note that we must clear the cache whenever the point x^* changes.

We denote with L&P_c and L&P2_c the variants of L&P and L&P2 (respectively) that maintains such cache of lines. The corresponding results are shown in Table 3, where column

Delta denotes the percentage of CGLPs spared by using the cache. The effect of the cache is negligible when using elementary disjunctions only, but is noticeable for the nonelementary case. Unfortunately, a 40% saving in the number of CGLPs solved allows for only a 20% saving in running time (approximately), because of the bookkeeping overhead.

4.4. Hybrid procedure

Implementing the hybrid procedure described in Section 3, which we call `mix`, is pretty straightforward. However, some care must be taken in implementing row aggregation, since it can quickly become too time consuming. Indeed, our very first implementation exhibited a performance hit of about one order of magnitude compared to the corresponding method without aggregation, which is clearly unacceptable. To speed up the heuristic, we implemented the following:

- We considered only pairs of rows whose basic variables are both integer constrained and fractional in the current basis.

- We computed the starting tableau rows and all tried combination only on the set S. Note that this is enough to find out whether a given combination yields a cut which is more violated than the cuts obtained by the generating rows.

- Given a row i, we bounded the number of rows for which we try aggregations to a small constant number, say 10, thus reducing the number of pairs tried from quadratic to linear. In our computational experience, this reduction did not impact significantly the effectiveness of the separated cuts.

Table 4 reports the aggregated results for all the variants of our hybrid procedure. There, `mix2` refers to the version of `mix` where also nonelementary disjunctions are tried, while the suffix `+aggr` denotes the variants where row aggregation is turned on. For completeness, we also report the results obtained by adding row aggregation to the original `rc` method.

As seen from the table, our basic `mix` procedure turns out to be very effective on both testbeds, improving significantly over the individual `rc` and `L&P_c` methods, with very reasonable computing times. In particular, we could close 53% of the integrality gap on the MIPLIB 2003 testbed in less than 40 seconds on average. If we compared it to the `fast` variant in Fischetti and Salvagnin (2011), which was the previously-best method on this testbed, we have an increase in gap closed of 9%, in roughly the same amount of time.

Table 4: Average gap closed and computing times for the new hybrid procedure, compared to other methods.

method	MIPLIB 3.0		MIPLIB 2003	
	cl.gap	time (s)	cl.gap	time (s)
rc	59.0%	0.73	48.2%	13.94
L&P_c	65.4%	1.98	44.8%	87.13
L&P2_c	70.2%	8.12	50.3%	508.31
mix	66.9%	1.76	53.0%	38.26
mix2	69.5%	2.90	53.0%	56.31
rc+aggr	63.1%	1.28	51.8%	42.04
mix+aggr	70.2%	2.45	54.9%	74.19
mix2+aggr	71.2%	3.28	55.2%	110.09
Cplex B&C	94.3%	2.18	80.1%	657.45

As far as aggregation is concerned, we note that adding row aggregation, even with the very simple algorithm implemented here, yields a noticeable improvement with any method, increasing the gap closed by approximately 5% on MIPLIB 3.0 and by $2 - 3\%$ on MIPLIB 2003. With the current implementation, row aggregation is not very expensive in the easier testbed, while it incurs a 2x slowdown on the harder testbed.

The most effective method, although not surprisingly the slowest, is mix2+aggr, which on average can close 71.2% of the integrality gap in 3.28 seconds on the MIPLIB 3.0 testbed, and 55.2% in 110.09 seconds on the MIPLIB 2003. According to Balas and Saxena (2008) and Dash et al. (2010), on the MIPLIB 3.0 exact split closure optimization yields a gap closed of approximately 80%, so we are less than 9% away from the exact bound, but orders of magnitude faster.

The last row in Table 4, Cplex B&C, reports the results obtained by running Cplex 12.2 with 1-hour time-limit in its default settings, except for the number of threads which is fixed to 1. This is by no means a fair comparison, as Cplex does not implement a pure rank-1 cutting-plane algorithm, but is free to use any machinery, among which primal and dual preprocessing, symmetry breaking, higher rank cutting planes (not necessarily of the split family) and, of course, enumeration. However, we find that such a comparison is very useful to put numbers into the right perspective. In particular, because of the steady improvements in MIP solvers, instances of MIPLIB 3.0 have become nowadays way too easy and even 3 seconds on average can be too much for a cutting plane separator. At the same time, many

instances in the MIPLIB 2003 are still quite challenging, and for some of them the procedures presented in this paper may turn out to be instrumental for their effective solution—we will elaborate this statement in the next section.

4.5. Solving Hard MIPs

In this section, we address the question of whether our aggressive cut generation policies can help in solving MIPs in practice.

It is pretty clear that any aggressive cutting plane strategy is not going to pay off if the instance to solve is easy. Actually, our computational experience shows that not only the instance must be very hard for an aggressive cut generator to be worthwhile, but the gap closed by cutting planes must be significant to really speedup enumeration.

For the above reasons, we restricted our attention to the instances in our testbed that satisfy the following requirements: (1) they cannot be solved by Cplex 12.2 with default settings in less than 10 minutes, and (2) our `mix+aggr` is able to close more than 70% of the integrality gap. These criteria pinpoint the very tough instances of Table 5, including `sp97ar` whose initial integrality gap is very small, yet finding an optimal solution is very hard. To remove variability due to heuristics, for this instance we considered also the case where we give the known optimal solution to Cplex since the beginning, and we call it `sp97ar-opt`.

In Table 5 we report a comparison between a Cplex 12.2 with its default settings (thus using all 4 cores of our machine) with and without the split cuts generated by `mix+aggr`, which, according to Table 4, can be considered the best trade-off between gap closed and computing time. All runs were given a time limit of 36,000 seconds (10 hours).

To minimize the side-effects of our aggressive cut generation procedure, yet keeping as much information as possible, we opted for the following strategy: we first apply `mix+aggr` to the original problem instance, then we only keep in the formulation all the cuts that are tight at the end of our loop, and copy all the slack cuts to Cplex's own cut pool, trusting that the solver will make the best use of such cuts.

According to Table 5, our aggressive cut separation procedure can indeed prove helpful in solving (some) hard MIPs. In particular, it speeds up the solution of instance `nsrand-ipx` by one order of magnitude and allows for the solution of `a1c1s1` and `sp97ar`. According to Achterberg et al. (2006) and Laundy et al. (2009), both instances were previously solved only using particular ad-hoc "tricks" and with much larger computing times.

As to `timtab2`, the strategy of adding our cuts at the root node only is not enough to close the instance, although the gap after 10 hours is more than halved. Note that `timtab2` has been solved without using any problem specific technique only very recently in in Dash and Goycoolea (2010), and required 92 hours of computing time and the separation of split cuts at *every* node of the branch-and-cut tree.

Table 5: Effect of adding split cuts derived with `mix+aggr` to Cplex 12.2.

	`cplex` default		`cplex`+`mix`+`aggr`	
instance	final gap	time (s)	final gap	time (s)
`a1c1s1`	2.81%	36,000	0.00%	9,734
`nsrand-ipx`	0.00%	1,261	0.00%	114
`sp97ar`	0.30%	36,000	0.00%	13,742
`sp97ar-opt`	0.21%	36,000	0.00%	4,000
`timtab2`	19.09%	36,000	9.33%	36,000

5.   Conclusions

The split closure attracted a considerable attention in recent years, as it provides a very tight approximation of the integer hull in practice. However, even an approximate optimization over the split closure is a very challenging computational task. In the present paper we have elaborated a combination of methods and ideas, both from the literature and new, that lead to a sound heuristic whose practical effectiveness has been tested on very large classes of test problems. The outcome is that one can achieve a very good approximation of the split closure bound in a computing time that is surprisingly small, at least if compared with previous proposals from the literature. The practical use of our separation scheme within state-of-the-art branch-and-cut solvers has been tested on four very tough instances, with remarkably good results.

Acknowledgements

References

Achterberg, T. 2007. Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin.

Achterberg, T., T. Koch, A. Martin. 2006. MIPLIB 2003. *Operations Research Letters* **34** 1–12. doi:10.1016/j.orl.2005.07.009. URL `http://www.zib.de/Publications/abstracts/ZR-05-28/`. See `http://miplib.zib.de`.

Andersen, K., G. Cornuéjols, Y. Li. 2005a. Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts. *Management Science* **51** 1720–1732.

Andersen, K., G. Cornuéjols, Y. Li. 2005b. Split closure and intersection cuts. *Mathematical Programming* **102** 457–493.

Andersen, K., R. Weismantel. 2010. Zero-coefficient cuts. Friedrich Eisenbrand, F. Shepherd, eds., *Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science*, vol. 6080. Springer Berlin / Heidelberg, 57–70.

Andreello, G., A. Caprara, M. Fischetti. 2007. Embedding {0,1/2}-cuts in a branch-and-cut framework: A computational study. *INFORMS Journal on Computing* **19** 229–238.

Balas, E. 1971. Intersection cuts — a new type of cutting planes for integer programming. *Operations Research* **19** 19–39.

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

Balas, E., R. G. Jeroslow. 1980. Strengthening cuts for mixed integer programs. *European Journal of Operational Research* **4** 224–234.

Balas, E., M. Perregaard. 2003. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. *Mathematical Programming* **94** 221–245.

Balas, E., A. Saxena. 2008. Optimizing over the split closure. *Mathematical Programming* **113** 219–240.

Bixby, R. E., S. Ceria, C. M. McZeal, M. W. P. Savelsbergh. 1998. An updated mixed integer programming library: MIPLIB 3.0. *Optima* **58** 12–15. See `http://www.caam.rice.edu/bixby/miplib/miplib.html`.

Bixby, R. E., M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling. 2000. MIP: theory and practice—closing the gap. *System modelling and optimization (Cambridge, 1999)*. Kluwer Acad. Publ., Boston, MA, 19–49.

Bixby, R. E., M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling. 2004. Mixed-integer programming: a progress report. M. Grötschel, ed., *The Sharpest Cut: The Impact of Manfred Padberg and His Work*. SIAM, 309–325.

Bonami, P. 2010. On optimizing over lift-and-project closures. Tech. rep., CNRS. URL `http://www.optimization-online.org/DB\_FILE/2010/10/2775.pdf`.

Caprara, A., M. Fischetti, A. N. Letchford. 2000. On the separation of maximally violated mod-k cuts. *Mathematical Programming* **87** 37–56.

Cook, W. J., R. Kannan, A. Schrijver. 1990. Chvátal closures for mixed integer programming problems. *Mathematical Programming* **47** 155–174.

Cornuéjols, Gerard, Giacomo Nannicini. 2011. Practical strategies for generating rank-1 split cuts in mixed-integer linear programming. *Mathematical Programming Computation* 1–38.

Dash, S., M. Goycoolea. 2010. A heuristic to generate rank-1 GMI cuts. *Mathematical Programming Computation* 231–257.

Dash, S., O. Günlük, A. Lodi. 2010. MIR closures of polyhedral sets. *Mathematical Programming* **121** 33–60.

Dash, S., O. Günlük, C. Raack. 2011a. A note on the MIR closure and basic relaxations of polyhedra. *Operations Research Letters* **3** 198–199.

Dash, S., O. Günlük, J. P. Vielma. 2011b. Computational experiments with cross and crooked cross cuts. URL `http://www.optimization-online.org/DB\_FILE/2011/06/3071.pdf`.

Dey, S., A. Lodi, A. Tramontani, L. Wolsey. 2010. Experiments with two row tableau cuts. Friedrich Eisenbrand, F. Bruce Shepherd, eds., *Integer Programming and Combinatorial Optimization, 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*, *Lecture Notes in Computer Science*, vol. 6080. Springer, 424–437.

Fischetti, M., A. Lodi. 2007. Optimizing over the first Chvàtal closure. *Mathematical Programming* **110** 3–20.

Fischetti, M., A. Lodi, A. Tramontani. 2011. On the separation of disjunctive cuts. *Mathematical Programming* 205–230.

Fischetti, M., D. Salvagnin. 2011. A relax-and-cut framework for Gomory mixed-integer cuts. *Mathematical Programming Computation* 79–102.

Gomory, R. E. 1963. An algorithm for integer solutions to linear programs. R. L. Graves, P. Wolfe, eds., *Recent Advances in Mathematical Programming*. McGraw-Hill, New York, 269–302.

Hiriart-Hurruty, C. Lemaréchal. 1993. *Convex Analysis and Minimization Algorithms*. Springer-Verlag.

Laundy, R., M. Perregaard, G. Tavares, H. Tipi, A. Vazacopoulos. 2009. Solving hard mixed-integer programming problems with xpress-MP: A MIPLIB 2003 case study. *INFORMS Journal on Computing* **21** 304–313.

Marchand, H., L. Wolsey. 2001. Aggregation and mixed integer rounding to solve mips. *Operations Research* **49** 363–371.

Nemhauser, G. L., L. Wolsey. 1990. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Mathematical Programming* **46**.

Wolter, Kati. 2006. Implementation of Cutting Plane Separators for Mixed Integer Programs. Master's thesis, Technische Universität Berlin.