

# Approximating the Split Closure

Matteo Fischetti, Domenico Salvagnin

Department of Information Engineering, University of Padova, 6/B 35131 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 few 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

*History:* Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received October 2011; revised April 2012, September 2012; accepted October 2012. Published online in *Articles in Advance* March 14, 2013.

## 1. Introduction

In this paper, we consider a mixed-integer linear program (MIP) in standard form:

$$\min cx, \quad (1)$$

$$Ax = b, \quad (2)$$

$$x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}, \quad (3)$$

where  $A$  is an  $m \times n$  rational matrix, and  $b, c$  are rational vectors of appropriate size. The index set  $J := \{1, \dots, p\} \subseteq N := \{1, \dots, n\}$  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 ingredients for the practical effectiveness of the B&C algorithm, as confirmed in many computational surveys, such as 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  $\pi_{p+1} = \dots = \pi_n = 0$  and  $\pi_0 \in \mathbb{Z}$ , we define as *split* the disjunction

$$\pi x \leq \pi_0 \vee \pi x \geq \pi_0 + 1. \quad (4)$$

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

$$\begin{aligned} P_1 &= \{x \in P: \pi x \leq \pi_0\} \quad \text{and} \\ P_2 &= \{x \in P: \pi x \geq \pi_0 + 1\} \end{aligned} \quad (5)$$

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. Because we do not consider higher rank closures, the adjective “first” will be omitted in what follows.

Recent computational studies in Fischetti and Lodi (2007), Balas and Saxena (2008), and 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, as shown in Caprara and Letchford (2003), and unacceptably slow in practice. Given the practical strength (and inherent difficulty) of the split closure, a lot of research has been done in recent 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 (2012), Cornuéjols and Nannicini (2011), and Fischetti and Salvagnin (2011).

In this paper we present a new overall framework for approximating the split closure that merges different ideas from the previous approaches. The paper is organized as follows. Section 2 surveys current state-of-the-art approaches for optimizing heuristically over the split closure. In §3 we describe our new

hybrid procedure along with algorithmic techniques, and computational results are presented in §4. Some conclusions are finally drawn in §5.

## 2. Literature

According to its definition, a split cut  $\gamma x \geq \gamma_0$  is defined by the choice of a split  $(\pi, \pi_0)$  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: the corresponding optimization problem is NP-hard and computationally very expensive in practice (Balas and Saxena 2008, Dash et al. 2010).

However, the task simplifies a lot if one fixes either the split or the set of multipliers in advance—almost all separation procedures involving split (or MIR/GMI) cuts can be interpreted this way. For example, given a fixed split disjunction  $(\pi, \pi_0)$  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 (Balas 1971) that reads:

$$\min \gamma x^* - \gamma_0, \quad (6)$$

$$\gamma \geq uA - \pi u_0, \quad (7)$$

$$\gamma_0 \leq ub - \pi_0 u_0, \quad (8)$$

$$\gamma \geq vA + \pi v_0, \quad (9)$$

$$\gamma_0 \leq vb + (\pi_0 + 1)v_0, \quad (10)$$

$$\langle \text{normalization condition} \rangle, \quad (11)$$

$$u_0, v_0 \geq 0. \quad (12)$$

In the previous LP, variables  $(u, u_0)$  certify the validity of the cut with respect to (w.r.t.)  $P_1$ , whereas variables  $(v, v_0)$  certify the validity of the cut w.r.t.  $P_2$ . The feasible set of (6)–(12) is a cone, and a normalization condition (11) is often added to truncate it.

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, whereas 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  $P$ .

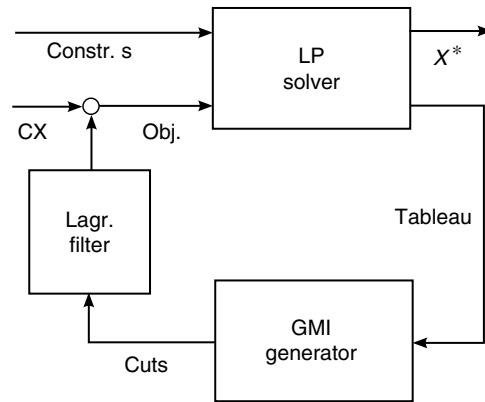


Figure 1 Basic Relax-and-Cut Scheme for GMI Cuts

### 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). The basic framework is depicted in Figure 1, whereas a more detailed algorithmic description is given in Algorithm 1.

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:

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

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

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

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-Urruty and Lemaréchal (1996).

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,

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

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 (Bonami 2012):

$$v^* := \max v = y_k - \lceil x_k^* \rceil, \quad (15)$$

$$Ay = b, \quad (16)$$

$$0 \leq y \leq x^*/d^*, \quad (17)$$

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  $v^* < 0$ .

The previous model has a neat geometrical interpretation (Bonami 2012) that will be used in §4.3 to speed up the separation procedure. Indeed, the order of business of separation is to determine whether  $x^* \in \text{conv}(P_1 \cup P_2)$ , 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 assume without loss of generality (w.l.o.g.) that  $y$  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, because both  $x^*$  and  $y$  satisfy the equation system  $Ax = b$ , whereas  $z = (x^* - \delta^* y)/(1 - \delta^*) \geq 0$  is because of the variable upper bounds in (17). Therefore, if the CGLP is able to find a solution  $y$  with  $v \geq 0$ ,

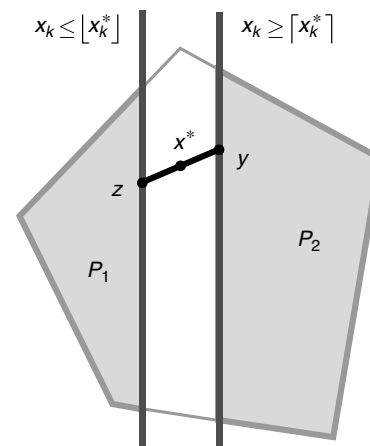


Figure 2 Geometric Interpretation of CGLP (15)–(17)

then we have a certificate that the point  $x^*$  belongs to  $\text{conv}(P_1 \cup P_2)$  and cannot be separated. Otherwise, we can derive a violated disjunctive cut by duality; see Balas and Saxena (2008) and Bonami (2012) for details. Note that if  $x_j$  happens to be free for some  $j \in N$ , then the associated  $y_j$  becomes free, and both its bounds vanish from the CGLP model.

The CGLP (15)–(17) has some 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 plays an important role in understanding the correspondence between split and GMI cuts. The result was first proven in Andersen et al. (2005b), whereas a shorter proof appeared recently in Dash et al. (2011a). A new proof is provided next and uses a recent result of Bonami (2012).

**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.** Because  $x^* \in P$  does not belong to the split closure, there exists a split  $(\pi, \pi_0)$  from which a disjunctive cut violated by  $x^*$  can be derived. Given  $(\pi, \pi_0)$ , we show how to determine the basis  $B$  of  $A$  and the integer tableau-row combination producing the claimed violated GMI cut. 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:

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

$$Ay = b, \quad (19)$$

$$-\pi y + y_{n+1} = 0, \quad (20)$$

$$0 \leq y \leq x^*/d^*, \quad (21)$$

$$y_{n+1} \text{ free}, \quad (22)$$

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  $y^*$  the associated optimal vertex. Observe that  $v^*$  must be fractional, because 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 with  $B'$  where  $y_{n+1}$  is basic, with  $\alpha_{n+1} = 0$ . It was shown in Bonami (2012, 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  $\text{conv}(P_1 \cup P_2)$  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 (not necessarily feasible) basis  $B$  of  $A$ . To this end, observe that the structure of  $B'$  is:

$$\begin{bmatrix} B & 0 \\ -\pi_B & 1 \end{bmatrix}, \quad (23)$$

where  $\pi_B$  denotes the projection of  $\pi$  on the set of basic variables of  $y$ , and  $B$  is a basis of  $A$ . The inverse of  $B'$  can be stated as

$$\begin{bmatrix} B^{-1} & 0 \\ \pi_B B^{-1} & 1 \end{bmatrix}; \quad (24)$$

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

$$[\pi_B B^{-1} \quad 1] \begin{bmatrix} A & 0 \\ -\pi & 1 \end{bmatrix} \begin{bmatrix} y \\ y_{n+1} \end{bmatrix} = [\pi_B B^{-1} \quad 1] \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad (25)$$

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  $\pi_B$ ) 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.  $\square$

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 also considering primal-infeasible bases and, if nonelementary disjunctions are used, also taking combinations of tableau rows before deriving the GMI cuts.

### 2.3. Row-Aggregation Heuristics

Theorem 1 highlighted the importance of taking combinations (with integral coefficients) of tableau rows before deriving the GMI cuts. Such a combination is needed in general to actually achieve the split closure—just considering all GMI cuts from all feasible and infeasible bases of  $P$ , called the strengthened 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), one of the most successful cutting plane heuristics for general MIPs.

If we restrict ourselves to aggregations of tableau rows with integral coefficients, then we obtain the so-called *reduce-and-split* method, proposed for the first time in Andersen et al. (2005a). The idea behind this method is to perform row aggregation before deriving the GMI cut, 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 can be given arbitrarily large coefficients by the GMI formula, whereas integer variables always get a coefficient between 0 and 1. 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 previous definition 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 at a 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 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 (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 different points of view; hence, they have complementary strength. Although *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 (Bonami 2012) 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 Algorithm 2. We decided to use the *relax-and-cut* scheme as a backbone, and to “plug” other procedures into it. This is justified by the fact that the *relax-and-cut* scheme has proven to be very effective (and numerically stable) in handling a large amount of cuts, and 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

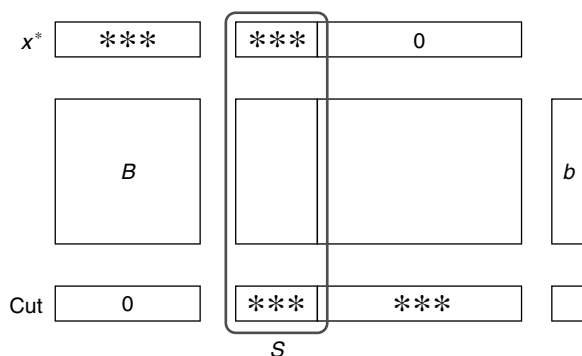


Figure 3 Variables in  $S$  Are Responsible for Not Cutting  $x^*$

in §2.3, computing all possible aggregations of two tableau rows with integer multipliers in the set  $\{-2, -1, 1, 2\}$  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  $x^* = \text{LPSolve}(P)$ 
2  $pool = \text{ReadGMI}(P)$ 
3  $P' = P \cap \{\text{cuts in } pool\}$ 
4 while not termination condition do
5    $x^* = \text{LPSolve}(P')$ 
6   if  $x_j^*$  is integer  $\forall j \in J$  then break
7    $u = \text{optimal LP dual vector for cuts in } pool$ 
8    $pool = \text{SubgradientRun}(P, pool, I_{\max}, u, doAggr)$ 
9   foreach  $x_j \in J$  s.t.  $x_j^* \notin \mathbb{Z}$  do
10      $\text{SolveCGLP}(x^*, x_j \leq \lfloor x_j^* \rfloor \vee x_j \geq \lceil x_j^* \rceil)$ 
11     if violated cut found then strengthen it and  

        add it to  $pool$ 
12   end
13    $P' = P \cap \{\text{cuts in } pool\}$ 
14 end
15 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-feasible solution is known (because an integer-feasible solution is needed for computing the integrality gap). In the end, we were left with 54 instances from MIPLIB 3.0 and 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, whereas 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 2.66 GHz, with 8 GB of RAM (only one CPU was used by each process). Every method was given a time limit of one 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  $10^{10}$ . The cut is also discarded if too dense; the maximum allowed density being dynamically calculated as the maximum density of the original constraints, multiplied by 10. All 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 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 fractionality to  $10^{-4}$  and the minimum efficacy to  $10^{-5}$ . Our pool size is limited to 1 GB 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 of 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

$$\begin{aligned} \min & cx, \\ Ax + Is &= b, \\ l &\leq x \leq u, \\ x &\in \mathbb{R}^n \quad s \in \mathbb{R}_+^m, \end{aligned}$$

where variables  $s$  are the slacks of the constraints (if the original constraint is already an equality, the corresponding slack is fixed to 0). 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 to improve numerical accuracy.
2. Get the  $i$ th row  $\rho$  of  $B^{-1}$ .
3. Compute the tableau row  $\alpha x + \rho s = \beta$ , where  $\alpha = \rho A$  and  $\beta = \rho b$ . We use standard sparse linear algebra structures to compute the tableau row working directly on the support of the row. If the tableau row gets too dense, we discard it. We also discard the row if an integer free variable gets a noninteger coefficient, or if a continuous free variable gets a nonzero coefficient (it is impossible to derive a cut in these cases).
4. Shift and complement variables 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  $b_j$  that 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, right-hand-side shifting, only once.
5. Apply the GMI formula to the coefficients in the support of the row; slack variables corresponding to constraints involving only integer variables with integer coefficients are considered integer as well.
6. Undo variable shifts and complements.
7. Back-substitute slacks to have a cut expressed in the space of structural variables only. Note that we do not consider cancellations 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 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 three 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 aforementioned, 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 1,000.
- 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 nonbasic 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, whereas closing approximately the same amount of gap as fast (and significantly more for the harder MIPLIB 2003 testbed).

#### 4.3. Lift-and-Project

Following the description in Bonami (2012), we implemented a basic cutting plane scheme based on lift-and-project separation. The main purpose of this

**Table 1** Average Gap Closed and Computing Times Original and Improved Versions of Relax-and-Cut

Method	MIPLIB 3.0		MIPLIB 2003	
	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

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 (2012), 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 (Bonami 2012). It is worth noting that such an equivalence result is usually stated for LPs in standard form, which is not the computational form used by simplex implementations. However, it can be shown that the equivalence still holds for the simplex method with general upper bounds, provided that the tableau row used to derive the GMI cut includes the upper-bound constraints on the variables (weighted by their corresponding dual multipliers) in the combination.

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

```

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 one hour is reached.

It was suggested in Bonami (2012) that to reduce the number of CGLPs solved at each iteration—and thus, to speed up 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 lead 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 (2012) 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 lead to the most effective warm start for the simplex method. Our computational experience confirms these findings, so this technique is implemented in our code. According to our computational experience, our implementation of the lift-and-project algorithm (L&P) is equivalent to the one in Bonami (2012), for what concerns both computing time and gap closed.

Starting from the basic implementation just presented, we tried to push the pure lift-and-project approach further by also considering nonelementary disjunctions. The main issue with nonelementary disjunctions is that there 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, this time 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  $O(m^2)$  nonelementary 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 2. 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 nonelementary disjunctions is that usually only a small number of them

**Table 2** 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



will eventually yield a violated cut, meaning that the vast majority of CGLPs are built and solved in vain. Therefore, 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 §2.2 provides some insights. Whenever the CGLP fails, i.e., it was not possible to derive a violated cut by a given disjunction  $(\pi, \pi_0)$ , we have as a by-product a pair of points  $(y, z)$  that acts as a certificate that  $x^*$  cannot be cut using  $(\pi, \pi_0)$ . Trivially, if we are given a different disjunction  $(\pi', \pi'_0)$  and the pair  $(y, z)$  satisfies  $y \in P_1$  and  $z \in P_2$ , then  $x^*$  cannot be cut by using  $(\pi', \pi'_0)$  as well. A similar check was also developed independently by Dash et al. (2011b).

The previous 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 this, whenever the CGLPs fails, we add the corresponding line to a pool of lines  $l \in \mathbb{R}^n$ , and we associate a step-size interval  $[t_{\min}, t_{\max}]$  to each line. This interval, which can be computed with a ratio-test procedure, describes how far a point  $x^* + tl$  can be moved along line  $l$  without leaving  $P$ . Whenever we are given a disjunction  $(\pi, \pi_0)$  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 a cache of lines. The corresponding results are shown in Table 2, where the columns *Delta* denote 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 §3, which we call *mix*, is straightforward. However, some care must be taken in implementing row aggregation, because 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 tried all combinations only on the set  $S$ ; see Figure 3. Note that this is enough to find out whether a given combination yields a cut that 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 particular, we selected rows  $i + 1, \dots, i + 10$ ). In our computational experience, this reduction did not significantly impact the effectiveness of the separated cuts.

Table 3 reports the aggregated results for all the variants of our hybrid procedure. There, *mix2* refers to the version of *mix* where nonelementary disjunctions are also tried, whereas 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 Table 3, our basic *mix* procedure turns out to be very effective on both testbeds, significantly improving 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.

As far as aggregation is concerned, we note that adding row aggregation, even with the very simple algorithm implemented here, yields a noticeable

**Table 3** 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
seq-best	71.2	25.19	53.4	953.37
seq-best-tuned	71.2	12.37	53.4	289.74
Cplex B&C	94.3	2.18	80.1	657.45

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, whereas it incurs a  $2\times$  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.

It was observed in Bonami (2012) that by sequentially running the three rank-1 cut generators *rc*, *L&P*, and *DG* (the *def* variant in Dash and Goycoolea 2010) and just taking the best bound, one can close approximately 70% of the gap in MIPLIB 3.0. Row *seq-best* in Table 3 reports the results that one would obtain through this sequential procedure. It is clear, however, that blindly adding the running times of the three methods does not give a competitive procedure, mainly because *DG* is very slow on average. To fix this issue, we ideally put a very strict time limit on *DG*, namely, 10 seconds for MIPLIB 3.0 and 50 seconds for MIPLIB 2003, and optimistically assume it closes the same amount of gap as if it had a time limit of one hour. We call the resulting procedure *seq-best-tuned*. Table 3 clearly shows that *seq-best-tuned* is much faster than *seq-best*, while closing (by construction) the same gap. If we compare *seq-best-tuned* with the proposed variants, we see that our methods outperform the trivial combination significantly. Indeed, on MIPLIB 3.0, *mix2 + aggr* is almost four times faster than *seq-best-tuned*, whereas on MIPLIB 2003 *mix* is already quite close as far as gap is concerned, but is more than seven times faster.

The last row in Table 3, *Cplex B&C*, reports the results obtained by running CPLEX 12.2 with one-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, because CPLEX does not implement a pure rank-1 cutting plane algorithm, but is free to use any machinery, among which are 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 three 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 clear that any aggressive cutting plane strategy is not going to pay off if the instance to solve is easy. For this reason, we restricted our attention to the instances in our testbed that cannot be solved by CPLEX 12.2 with default settings in less than 10 minutes. In Table 4 we report a comparison between CPLEX 12.2 with its default settings (thus using all four cores of our machine) with and without the split cuts generated by *mix + aggr*, which, according to Table 3, can be considered the best tradeoff between gap closed and computing time. All runs were given a time limit of 10,800 seconds (three hours). For each instance and for both methods under comparison, Table 4 reports computing time, the final integrality gap (column *f. gap*), and the number of nodes (in thousands). For *cplex + mix + aggr*, the table also reports the computing time spent at the root node within our split cut separator (column *sep. t.*), as well as the corresponding percentage integrality gap closed (column *cl. gap*). Note that our split cut separator is sequential, so it uses just one core.

To minimize 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 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 4, our very aggressive cutting plane strategy yields mixed results. Although separation is quite cheap on average, taking approximately 100 seconds in geometric mean, still, on some instances the time spent in separating cutting planes is pretty high. Even worse, sometimes the resulting LP gets too slow to solve and the subsequent enumeration phase is less effective (see, for example, *net12*). On other instances, as, for example, *glass4*, the time spent separating cuts, and the number of cuts itself, is negligible, yet the B&C slows down considerably. In our view, however, this can be because of the erratic behavior of the enumeration procedure. However, on some instances, as, for example, *alc1s1* and *nsrand-ixx*, the strategy seems to really pay off.

The computational experience described suggests that not only the instance must be very hard for an

**Table 4** Effect of Adding Split Cuts Derived with *mix + aggr* to CPLEX 12.2 on Hard MIP Instances

Instance	cplex default			cplex + mix + aggr				
	f. gap (%)	Time (s)	Nodes (k)	f. gap (%)	Time (s)	Nodes (k)	sep. t. (s)	cl. gap (%)
alc1s1	4.2	10,800	2,158	0.0	9,728	493	100	92.0
atlanta-ip	2.0	10,800	29	7.0	10,800	17	1,102	1.0
dano3mip	16.2	10,800	15	20.2	10,800	1	94	0.2
danooint	0.0	1,496	610	3.0	10,800	195	59	4.9
glass4	0.0	1,497	6,300	0.0	6,489	22,559	1	0.0
mkc	0.1	10,800	1,452	0.2	10,800	171	162	82.6
momentum1	0.0	10,800	991	3.7	10,800	109	621	64.5
momentum2	0.0	3,722	97	12.6	10,800	24	288	45.4
msc98-ip	0.0	4,895	12	0.0	4,203	5	1,737	51.3
net12	0.0	3,940	4	19.5	10,800	1	1,611	23.5
nsrand-ipx	0.0	1,261	535	0.0	115	12	26	83.8
protfold	29.2	10,800	88	49.4	10,800	38	51	13.6
rd-rplusc21	100.0	10,800	858	100.0	10,800	3,192	140	0.0
seymour	1.0	10,800	806	0.9	10,800	44	1,426	65.4
sp97ar	0.4	10,800	877	0.1	10,800	146	226	73.5
swath	15.9	10,800	2,363	8.2	10,800	2,321	1	34.9
timtab2	19.3	10,800	5,303	11.9	10,800	2,402	14	70.2
	11.1	6,374	336	13.9	7,545	92	104	41.6

aggressive cut generator to be worthwhile, but the gap closed by cutting planes must be significant to really speed up enumeration. To better illustrate this point, let us consider the instances in the previous selection where our *mix + aggr* is able to close more than 70% of the integrality gap (as reported in column cl. gap). This criterion selects the very tough instances reported in Table 5, including *sp97ar* whose initial integrality gap is very small, yet finding an optimal solution is very hard (to remove variability because of heuristics, for this instance we also considered the case where we give the known optimal solution to CPLEX since the beginning, and we call it *sp97ar-opt*).

In Table 5 we then report the comparison between CPLEX 12.2 with its default settings with and without the split cuts generated by *mix + aggr*, on the above instances alone. All runs were given a time limit of 36,000 seconds (10 hours). According to Table 5, our aggressive cut separation procedure can indeed prove helpful in solving those hard MIPs. In particular, it speeds up the solution of instance *nsrand-ipx* by one order of magnitude and allows

for the solution of *alc1s1* 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 solve 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 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.

## 5. Conclusions

The split closure attracted considerable attention in recent years, because 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 very tough instances, with remarkably good results.

## Acknowledgments

This research was supported by the *Progetto di Ateneo* on “Computational Integer Programming” of the University

**Table 5** Effect of Adding Split Cuts Derived with *mix + aggr* to CPLEX 12.2 on Selected Instances

Instance	cplex default		cplex + mix + aggr		
	f. gap (%)	Time (s)	f. gap (%)	Time (s)	sep. t. (s)
alc1s1	2.81	36,000	0.00	9,734	100
mkc	0.14	36,000	0.14	36,000	162
nsrand-ipx	0.00	1,261	0.00	114	25
sp97ar	0.30	36,000	0.00	13,742	219
sp97ar-opt	0.21	36,000	0.00	4,000	219
timtab2	19.09	36,000	9.33	36,000	14

of Padova, and by MiUR, Italy (PRIN project “Integrated Approaches to Discrete and Nonlinear Optimization”). The second author was also supported by the *Progetto di Eccellenza 2008–2009* of the “Fondazione Cassa di Risparmio di Padova e Rovigo,” Italy.

## References

- Achterberg T (2007) Constraint integer programming. Ph.D. thesis, Technische Universität Berlin, Berlin, Germany.
- Achterberg T, Koch T, Martin A (2006) MIPLIB 2003. *Oper. Res. Lett.* 34(4):1–12.
- Andersen K, Weismantel R (2010) Zero-coefficient cuts. Eisenbrand F, Shepherd F, eds. *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, Vol. 6080 (Springer, Berlin/Heidelberg), 57–70.
- Andersen K, Cornuéjols G, Li Y (2005a) Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts. *Management Sci.* 51(11):1720–1732.
- Andersen K, Cornuéjols G, Li Y (2005b) Split closure and intersection cuts. *Math. Programming* 102(3):457–493.
- Andreello G, Caprara A, Fischetti M (2007) Embedding  $\{0, 1/2\}$ -cuts in a branch-and-cut framework: A computational study. *INFORMS J. Comput.* 19(2):229–238.
- Balas E (1971) Intersection cuts—A new type of cutting planes for integer programming. *Oper. Res.* 19(1):19–39.
- Balas E, Jeroslow RG (1980) Strengthening cuts for mixed integer programs. *Eur. J. Oper. Res.* 4(4):224–234.
- Balas E, Perregaard M (2003) A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0–1 programming. *Math. Programming* 94(2–3):221–245.
- Balas E, Saxena A (2008) Optimizing over the split closure. *Math. Programming* 113(2):219–240.
- Balas E, Ceria S, Cornuéjols G (1996) Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. *Management Sci.* 42(9):1229–1246.
- Bixby RE, Ceria S, McZeal CM, Savelsbergh MWP (1998) An updated mixed integer programming library: MIPLIB 3.0. *Optima* 58:12–15.
- Bixby RE, Fenelon M, Gu Z, Rothberg E, Wunderling R (2000) MIP: Theory and practice—Closing the gap. Powell MJD, Scholtes S, eds. *System Modelling and Optimization* (Kluwer Academic Publisher, Boston), 19–49.
- Bixby RE, Fenelon M, Gu Z, Rothberg E, Wunderling R (2004) Mixed-integer programming: A progress report. Grötschel M, ed., *The Sharpest Cut: The Impact of Manfred Padberg and His Work* (SIAM, Philadelphia), 309–325.
- Bonami P (2012) On optimizing over lift-and-project closures. *Math. Programming Comput.* 4(2):151–179.
- Caprara A, Letchford AN (2003) On the separation of split cuts and related inequalities. *Math. Programming* 94(2–3):279–294.
- Caprara A, Fischetti M, Letchford AN (2000) On the separation of maximally violated mod- $k$  cuts. *Math. Programming* 87(1):37–56.
- Cook WJ, Kannan R, Schrijver A (1990) Chvátal closures for mixed integer programming problems. *Math. Programming* 47(1–3):155–174.
- Cornuéjols G, Nannicini G (2011) Practical strategies for generating rank-1 split cuts in mixed-integer linear programming. *Math. Programming Comput.* 3(4):281–318.
- Dash S, Goycoolea M (2010) A heuristic to generate rank-1 GMI cuts. *Math. Programming Comput.* 2(3–4):231–257.
- Dash S, Günlük O, Lodi A (2010) MIR closures of polyhedral sets. *Math. Programming* 121:33–60.
- Dash S, Günlük O, Raack C (2011a) A note on the MIR closure and basic relaxations of polyhedra. *Oper. Res. Lett.* 39(1):198–199.
- Dash S, Günlük O, Vielma JP (2011b) Computational experiments with cross and crooked cross cuts. Unpublished manuscript.
- Dey S, Lodi A, Tramontani A, Wolsey L (2010) Experiments with two row tableau cuts. Eisenbrand F, Shepherd FB, eds. *Integer Programming and Combinatorial Optim., 14th Internat. Conf., IPCO 2010, Lausanne, Switzerland, June 9–11, 2010. Proc., Lecture Notes in Computer Science*, Vol. 6080 (Springer, Berlin), 424–437.
- Fischetti M, Lodi A (2007) Optimizing over the first Chvátal closure. *Math. Programming* 110(1):3–20.
- Fischetti M, Salvagnin D (2011) A relax-and-cut framework for Gomory mixed-integer cuts. *Math. Programming Comput.* 3(2):79–102.
- Fischetti M, Lodi A, Tramontani A (2011) On the separation of disjunctive cuts. *Math. Programming* 128(1–2):205–230.
- Gomory RE (1963) An algorithm for integer solutions to linear programs. Graves RL, Wolfe P, eds. *Recent Advances in Mathematical Programming* (McGraw-Hill, New York), 269–302.
- Hiriart-Urruty J-B, Lemaréchal C (1996) *Convex Analysis and Minimization Algorithms Part 1: Fundamentals* (Springer-Verlag, Berlin).
- Laundy R, Perregaard M, Tavares G, Tipi H, Vazacopoulos A (2009) Solving hard mixed-integer programming problems with Xpress-MP: A MIPLIB 2003 case study. *INFORMS J. Comput.* 21(2):304–313.
- Marchand H, Wolsey LA (2001) Aggregation and mixed integer rounding to solve MIPs. *Oper. Res.* 49(3):363–371.
- Nemhauser GL, Wolsey L (1990) A recursive procedure to generate all cuts for 0–1 mixed integer programs. *Math. Programming* 46(1–3):379–390.
- Wolter K (2006) Implementation of cutting plane separators for mixed integer programs. Master’s thesis, Technische Universität Berlin, Berlin, Germany.

Copyright 2013, by INFORMS, all rights reserved. Copyright of Journal on Computing is the property of INFORMS: Institute for Operations Research and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.