Split cuts from sparse disjunctions

Ricardo Fukasawa Laurent Poirrier Shenghao Yang*

Department of Combinatorics and Optimization
University of Waterloo, Canada

July 31, 2018

Abstract

Split cuts are arguably the most effective class of cutting planes within a branch-and-cut framework for solving general Mixed-Integer Programs (MIP). Sparsity, on the other hand, is a common characteristic of MIP problems, and it is an important part of why the simplex method works so well inside branch-and-cut. In this work, we evaluate the strength of split cuts that exploit sparsity. In particular, we show that restricting ourselves to sparse disjunctions—and furthermore, ones that have small disjunctive coefficients—still leads to a significant portion of the total gap closed with arbitrary split cuts. We also show how to exploit sparsity structure that is implicit in the MIP formulation to produce splits that are sparse yet still effective. Our results indicate that one possibility to produce good split cuts is to try and exploit such structure.

1 Introduction

Cutting planes are fundamental in solving mixed-integer linear programs (MIPs). Over the last 25 years, commercial solvers have accomplished remarkable progress, achieving a machine-independent speed-up of the solution process by more than a factor of 450,000 [9]. General-purpose cutting plane techniques, such as Gomory Mixed Integer (GMI) cuts and Mixed Integer Rounding (MIR) cuts, are arguably the most important contributors to this progress (see, for example, [11]).

To study the impact that a particular family of cuts may have, both theoretically and computationally, a common approach is to consider the *closure* of those cuts. Given a family of cuts, its closure is defined as the intersection of all cuts belonging to the same family that are obtainable from the original MIP formulation. On the theoretical side, topics range from determining the polyhedrality of closures [15] to analyzing their strength [7].

*{rfukasawa,lpoirrier,s286yang}@uwaterloo.ca

On the computational front, several authors proposed strategies to empirically evaluate the strength of different closures by computing the amount of integrality gap they close: we thus have computational evaluations of the Chvátal closure [20], the split closure [6], the projected Chvátal-Gomory closure [13], the MIR closure [17] and the lift-and-project closure [12].

We focus on the results on the split closure (or equivalently the MIR closure), which consider the class of all split cuts [15], since these are the cuts that are most useful in practice to solve MIPs [11]. Besides efforts on efficient generation of strong split cuts (see for instance [2, 16, 22, 23]), the split closure was shown to be a very tight approximation to the convex hull of all feasible solutions in the corresponding MIP [6, 23]. On average it closes more than 75% of the integrality gap on MIPLIB 3.0 [10] instances. The purpose of this work is to determine what will be the effect on this integrality gap if we restrict ourselves to a subset of split cuts defined by its sparsity properties. In the following discussion we motivate such choice of restriction.

Sparsity is a natural condition that helps in the linear algebra routines of the simplex method [26], thus it is a desirable property of cutting planes for MIPs. Indeed, in almost every cut generation procedure described in the articles we mentioned above, specific heuristics are implemented to impose sparsity in the cuts, e.g., introducing a penalty term in the objective of a cut generating problem to make the resulting cut sparser [20], applying a coefficient reduction algorithm to reduce the number of nonzeros the split cut [16], and discarding all dense cuts to ensure that only sparse cuts are added [22]. The effect of sparsity has also recently been noted in a computational study by Walter [27] where it is shown that equivalent, but denser versions of the same cuts negatively affect performance of MIP solvers. Due to all this interest, there has also been some recent work to analyze theoretically the strength of sparse cutting planes [18, 19].

One additional motivation to study the effect of sparsity is the recent result of Bergner et al. [8] where they show that several benchmark instances have an almost block-diagonal structure called *arrowhead*, that is, a structure with several blocks that are linked only by few linking variables and constraints. This shows that not only are these benchmark instances sparse (on average, MIPLIB 2010 [25] instances only have 1.62% density), but in many cases such sparsity has an identifiable structure that can be exploited.

The main contributions of this work can be stated as follows:

- We implement an approximate separation routine based on the work on Balas and Saxena [6] that separates only split cuts whose split disjunctions are sparse and whose split coefficients are small
- We show, empirically, that in spite of those restrictions, the gap closed by this subclass of split cuts is still quite significant (on average 91% of the split closure gap).

- Finally, we consider split cuts computed only from individual blocks of the arrowhead structure of the instances. We show that they also largely preserve the strength of general split cuts, in terms of gap closed.

These results help shed some light into what are important classes of split cuts that we can focus our attention on studying.

In the rest of this paper, we present in more details such results. Section 2 lays out the basic approach of Balas and Saxena [6] for the separation of split cuts, and briefly introduces the automatic decomposition of Bergner et al. [8]. In Section 3, we detail the implementation of our split cut separator. In particular, we describe exactly what measures we took to obtain cuts that are numerically stable and effective, while being verifiably valid. Section 4 presents the results of our computational experiments.

2 Background

In this section, we formally present the background necessary to explain our experiments. We start by introducing how to optimize over an approximation of the split closure and then discuss the developments related to the arrowhead decomposition.

2.1 Optimizing over the split closure

Consider a general MIP:

$$\min\{c^\top x : Ax = b,\ x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}\} \tag{MIP}$$

where $A \in \mathbb{Q}^{m\times n}$ has full row rank, $c \in \mathbb{Q}^n$ and $b \in \mathbb{Q}^m$. The linear programming relaxation of (MIP) is

$$\min\{c^\top x : x \in P\} \tag{LP}$$

where $P = \{x \in \mathbb{R}_+^n : Ax = b\}$. For any $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}$ such that $\pi_j = 0$ for $j \geq p+1$, a *split disjunction* is defined as

$$\pi^\top x \leq \pi_0 \ \vee\ \pi^\top x \geq \pi_0 + 1.$$

An inequality $\alpha^\top x \geq \beta$ valid for $P^{(\pi,\pi_0)}$ where

$$P^{(\pi,\pi_0)} = \operatorname{conv}\big(\{x \in P : \pi^\top x \leq \pi_0\} \cup \{x \in P : \pi^\top x \geq \pi_0 + 1\}\big)$$

is called a *split cut* [15].

The problem of finding a violated split is $\mathcal{NP}$-hard in general [14]. Following Farkas' lemma, a most-violated split cut $\alpha^T x \geq \beta$ for $P^{(\pi,\pi_0)}$ can be found by solving the Cut Generating Linear Program

$$\begin{aligned}
\min\ & \alpha^\top x - \beta \\
\text{s.t. } & \alpha = A^\top y + s - y_0\pi \\
& \alpha = A^\top z + t + z_0\pi \\
& \beta = b^\top y - y_0\pi_0 \\
& \beta = b^\top z + z_0(\pi_0 + 1) \\
& \texttt{normalization condition} \\
& y, z \in \mathbb{R}^m,\ s, t \in \mathbb{R}^n_+,\ y_0, z_0 \in \mathbb{R}_+.
\end{aligned} \tag{CGLP(π,π_0)}$$

A derivation of (CGLP(π,π_0)) and in-depth discussion of the normalization condition were presented in [21]. The following remark on the nonnegativity of the multipliers y and z is useful in simplifying our CGLP.

Remark 1. *Suppose* (CGLP(π,π_0)) *has an optimal solution under some choice of normalization, and let* $(\hat\alpha, \hat\beta, \hat y, \hat z, \hat s, \hat t, \hat y_0, \hat z_0)$ *be an optimal solution. Then*

$$\begin{aligned}
y_i^* &:= \max\{0, y_i - z_i\}, \quad i = 1, 2, \ldots, m \\
z_i^* &:= \max\{0, z_i - y_i\}, \quad i = 1, 2, \ldots, m \\
\alpha^* &:= \hat\alpha + A^\top(y^* - \hat y) \\
\beta^* &:= \hat\beta + b^\top(y^* - \hat y) \\
s^* &:= \hat s, \quad t^* := \hat t, \quad y_0^* := \hat y_0, \quad z_0^* := \hat z_0
\end{aligned}$$

is also an optimal solution (assuming that it, too, satisfies the normalization condition).

Therefore, we may assume w.l.o.g. in (CGLP(π,π_0)) that all multipliers are nonnegative since, as we will see below, our normalization allows it. In all subsequent discussions we assume $y, z \in \mathbb{R}^m_+$.

The *split closure* $\mathcal{C}$ is defined as

$$\mathcal{C} = \bigcap_{\substack{(\pi,\pi_0)\in\mathbb{Z}^n\times\mathbb{Z} \\ \pi_j=0,\, j\geq p+1}} P^{(\pi,\pi_0)}.$$

Balas and Saxena [6] implemented an iterative procedure that alternates between a Master Problem and a Separation Problem to find

$$\min\{c^\top x : x \in \mathcal{C}\}.$$

At each iteration, the Master Problem is a linear program of the form

$$\min\{c^\top x : x \in P,\ \alpha^t x \geq \beta^t,\ t \in T\} \tag{MP}$$

where $\{\alpha^t x \geq \beta^t : t \in T\}$ is the set of all split cuts generated by the Separation Problem so far. If $\hat{x}$ is an optimal solution to (MP), the Separation Problem then finds a valid cut violated by $\hat{x}$, or proves that $\hat{x} \in \mathcal{C}$. The Separation Problem is a mixed-integer nonlinear program obtained from (CGLP(π, π_0)) with normalization $y_0 + z_0 = 1$, and allowing (π, π_0) to vary over $\mathbb{Z}^n \times \mathbb{Z}$. Formally, the separation problem is stated as:

$$\begin{aligned}
\min\ & \alpha^\top \hat{x} - \beta \\
\text{s.t.}\ & \alpha = A^\top y + s - y_0 \pi \\
& \alpha = A^\top z + t + z_0 \pi \\
& \beta = b^\top y - y_0 \pi_0 \\
& \beta = b^\top z + z_0(\pi_0 + 1) \\
& 1 = y_0 + z_0 \\
& y, z \in \mathbb{R}^m_+,\ s, t \in \mathbb{R}^n_+,\ y_0, z_0 \in \mathbb{R}_+ \\
& (\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z},\ \pi_j = 0,\ j \geq p+1.
\end{aligned} \tag{SP}$$

In [6], (SP) is shown to be equivalent to a parametric mixed-integer linear program with scalar parameter θ,

$$\min_{0 \leq \theta \leq \frac{1}{2}} \text{MILP}(\theta)$$

where MILP(θ) is given by

$$\begin{aligned}
\min\ & s^\top \hat{x} - \theta(\pi^\top \hat{x} - \pi_0) \\
\text{s.t.}\ & A^\top w + s - t - \pi = 0 \\
& \qquad b^\top w - \pi_0 = 1 - \theta \\
& w \in \mathbb{R}^m,\ s, t \in \mathbb{R}^n_+ \\
& (\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z},\ \pi_j = 0,\ j \geq p+1.
\end{aligned} \tag{MILP(θ)}$$

Therefore, the optimum to (SP) can be approximated from above by solving a finite sequence of problems MILP(θ) with varying values for θ.

2.2 Automatic detection of double-bordered block-diagonal structure

The idea of exploiting block-diagonal structure in sparse matrices has been widely discussed in the contexts of numerical linear algebra and mathematical programming. One motivation is that the diagonal blocks usually give rise to small independent subproblems well suited for parallel processing. Applications include solving systems of linear equations arising from a discretization of a continuous domain, LU and QR factorizations, and decomposition-based solution methods for structured (mixed-integer) linear programs. In general, the constraint matrix A of (MIP) does not admit a block-diagonal form, but it can be put into a k-way *double-bordered block-diagonal form*

$$\begin{bmatrix} D^1 & & & & F^1 \\ & D^2 & & & F^2 \\ & & \ddots & & \vdots \\ & & & D^k & F^k \\ A^1 & A^2 & \cdots & A^k & G \end{bmatrix} \tag{DB-k}$$

for some $k \geq 1$. This is sometimes informally called the arrowhead form. The constraints associated with rows in A^i are called *linking constraints*, and the variables associated with columns in F^i are called *linking variables*.

Given a sparse matrix, Aykanat et al. [3] considered the problem of obtaining a DB-k form by permuting its rows and columns. They reduce the matrix permutation problem to that of graph and hypergraph partitioning. However, even when the number k of blocks is fixed, computational experiments show that the resulting DB-k forms demonstrate significant variability and are very sensitive to input parameters. To cope with this, Bergner et al. [8] proposed to use a proxy measure to automatically detect the "best" DB-k form, for the purpose of applying Dantzig-Wolfe reformulations to general MIPs. Figure 1 shows a few examples of MIPLIB instances, with black dots representing nonzero coefficients of the constraint matrix. The bottom row shows a rearrangement of the columns/rows of the matrix evidencing the DB-k structure.

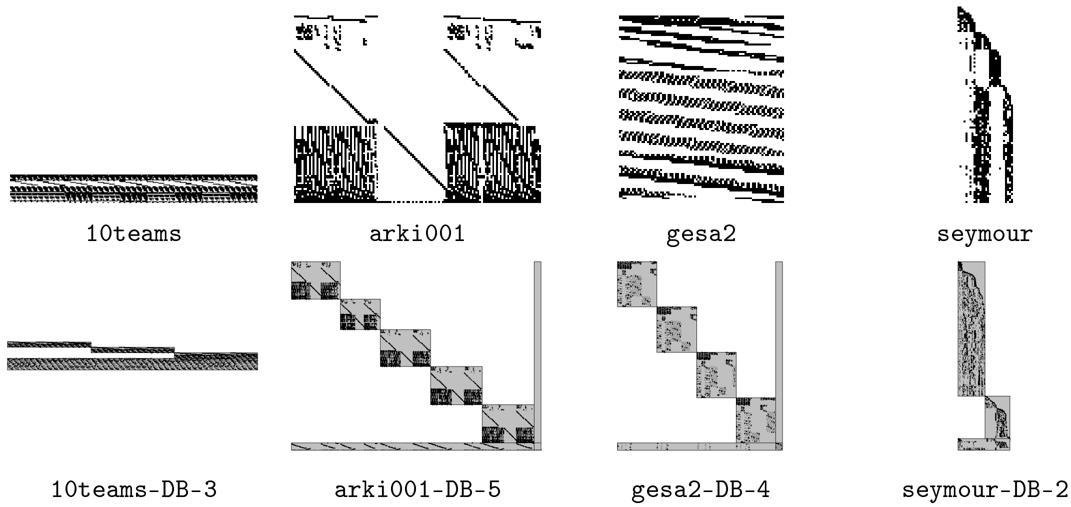


Figure 1: Original problem structure versus its DB-k forms

3 Implementation

In this section we outline the computational details of our implementation. We follow the idea of Balas and Saxena [6] to approximate the optimal value of (SP) by solving a sequence of parametric MILPs. Features prefixed by an asterisk (*) were already present in [6].

***Parameter grid.** We denote by Θ the set of values of θ for which MILP(θ) will be solved. A uniform parameter grid Θ of points between 0 and 0.5 is created. The initial size of Θ is t, and we increase the number of grid points whenever necessary following the criteria in Algorithm 2.

***Stabilizing objective.** To avoid unnecessarily weak cut coefficients (see [6] for a short discussion), we replace $\hat{x}$ in the objective of MILP(θ) with

$$\tilde{x}_j := \max\{\hat{x}_j, \delta\}, \quad \forall\, j,$$

for δ a small positive constant.

***Cut strengthening.** Once a feasible solution $(\bar{w}, \bar{s}, \bar{t}, \bar{\pi}, \bar{\pi}_0)$ to MILP($\bar{\theta}$) with a negative objective value is found, we feed $(\bar{\pi}, \bar{\pi}_0)$ to the corresponding *Cut Generating Linear Program* (CGLP($\bar{\pi}, \bar{\pi}_0$)) with normalization

$$e^\top y + e^\top z + e^\top s + e^\top t + y_0 + z_0 = \kappa$$

for a fixed positive constant κ. This normalization is shown in [21] to produce stronger cuts than the normalization $y_0 + z_0 = 1$ used in deriving MILP(θ).

***Cut lifting.** We work in the subspace of the variables that are not at one of their bounds in the incumbent solution, and lift the resulting cuts to the full space following the approach described in [5].

***Set covering.** In an effort to impose some orthogonality in the set of split disjunctions, every time a split $(\bar{\pi}, \bar{\pi}_0)$ is found, we solve the set covering problem

$$\min_{z \in \{0,1\}^p} \left\{ \sum_{j=1}^{p} \min\{\hat{x}_j - \lfloor \hat{x}_j \rfloor, \lceil \hat{x}_j \rceil - \hat{x}_j\} \, z_j : \sum_{j=1}^{p} \mathbb{I}_{[\pi_j \neq 0]} z_j \geq 1, \forall \pi \in \mathcal{S} \right\} \qquad (\text{StCvIP}(\hat{x}, \mathcal{S}))$$

where $\mathcal{S}$ is the set of splits already discovered, and $\mathbb{I}_{[k \neq 0]} = 1$ if $k \neq 0$, $\mathbb{I}_{[k \neq 0]} = 0$ if $k = 0$. Let $\hat{z}$ be an optimal solution to (StCvIP($\hat{x}, \mathcal{S}$)), then we impose $\pi_j = 0$ for all $j \in \{j : \hat{z}_j \neq 0\}$ when solving the next MILP(θ).

Fractionality constraint. Split disjunctions (π, π_0) where $\pi^\top \hat{x}$ is too close to either π_0 or $\pi_0 + 1$ usually give rise to weak split cuts. To avoid that, we impose the bounds

$$\sigma \leq \pi^\top \hat{x} - \pi_0 \leq 1 - \sigma \qquad \text{(Con1)}$$

for a small $\sigma > 0$.

Sparsity constraint. To impose the condition that π is sparse with at most M nonzero entries, we introduce binary variables $r \in \{0,1\}^p$ and constraints

$$-Ur_j \leq \pi_j \leq Ur_j, \quad \forall\, j = 1, \ldots, p, \quad \text{and} \quad \sum_{j=1}^{p} r_j \leq M, \qquad \text{(Con2)}$$

where U is an artificial upper bound on the magnitude of the components of π.

Structure constraint. Given a DB-k form of the constraint matrix A, to compute split disjunctions whose support lie entirely in a block D^i, we simply impose that:

$$\begin{aligned} \pi_j = s_j = t_j = 0, \quad \forall\, j \notin \mathcal{C}^i \\ w_j = 0, \quad \forall\, j \notin \mathcal{R}^i \end{aligned} \qquad \text{(Con3)}$$

where $\mathcal{C}^i$ and $\mathcal{R}^i$ are column and row index set of D^i, respectively.

Validity check. For every split cut $\alpha^\top x \geq \beta$ generated from CGLP with splits (π, π_0), we provide another certificate for the validity of the cut. Let

$$\hat{\beta}_l := \min_{x \in P}\{\alpha^\top x : \pi^\top x \leq \pi_0\} \quad \text{and} \quad \hat{\beta}_u := \min_{x \in P}\{\alpha^\top x : \pi^\top x \geq \pi_0 + 1\}.$$

Then it should always hold that $\beta \leq \min\{\hat{\beta}_l, \hat{\beta}_u\}$. If the inequality fails to hold, then the cut is invalid and we discard it. This may be the case due to numerical issues within the LP or MIP solver.

Cleaning up cut coefficients. To prevent cut coefficients from being too large or too small, once a split cut is returned by CGLP(π, π_0), we scale the cut so that the greatest absolute value of cut coefficients equals 10^4. Furthermore, after scaling we set all cut coefficients whose absolute value is less than 10^{-6} to zero. In general, setting a nonzero cut coefficient to zero may strengthen the cut and make it invalid, but since our tolerance is small, the effect is small as well. Nonetheless, the validity of the cut is always subsequently certified by the independent checker. Note that this scaling process also serves as an implicit dynamism control, i.e., the ratio between the greatest and the smallest absolute value of cut coefficients is no greater than 10^{10}.

The cut generation procedure is summarized in Algorithm 1.

Algorithm 1: Cut Generation$(\hat{x}, \Theta, \gamma, \tau)$

Input: Incumbent solution $\hat{x}$, parameter grid Θ, upper cutoff limit $\gamma < 0$, time limit τ, minimum cut violation $\epsilon > 0$, required properties of split disjunctions (Con1)-(Con3). Polyhedron $P = \{x \in \mathbb{R}^n_+ : Ax = b\}$ describing the constraint set of (LP).

Output: A set $\mathcal{K}$ of split cuts violated by $\hat{x}$.

$\mathcal{K} \leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$.

for $\theta \in \Theta$ **do**

  Solve StCvIP$(\hat{x}, \mathcal{S})$ and impose partial orthogonality if needed. Add fractionality, sparsity, and structure constraints (Con1)-(Con3) to MILP(θ) as indicated.

  Solve MILP(θ) with time limit τ.

  if *Found a feasible solution* (π, π_0) *to MILP(θ) with objective value* $\leq \gamma$. **then**

    Perform cut strengthening to get cut $\alpha^\top x \geq \beta$.

    Perform cut lifting on cut $\alpha^\top x \geq \beta$.

    Perform cut cleaning on cut $\alpha^\top x \geq \beta$.

    if $\alpha^\top \hat{x} - \beta \leq -\epsilon$ **then**

      $\beta_l \leftarrow \min_{x \in P} \{\alpha^\top x : \pi^\top x \leq \pi_0\}$, $\beta_u \leftarrow \min_{x \in P} \{\alpha^\top x : \pi^\top x \geq \pi_0 + 1\}$.
      $\beta^* \leftarrow \min\{\beta_l, \beta_u\}$.

      if $\beta \leq \beta^*$ **then**

        $\mathcal{K} \leftarrow \mathcal{K} \cup \{\alpha^\top x \geq \beta\}$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{\pi\}$.

return $\mathcal{K}$.

Time limit on the MIP solver. Mixed-integer linear programs are much harder to solve than linear programs in general. As a result, even finding a feasible solution to MILP(θ) can be extremely time-consuming. We observed that this is frequently the case, in particular, when separating a point that is close to the closure we aim to optimize over. Therefore, a deterministic time limit of 800 ticks (roughly 1 second) is set for each MILP(θ)

we process. We use CPLEX's *deterministic time* (ticks) so that the results are reproducible and comparable across different machines.

Dynamics. At each iteration, if no cut is generated because we could not find a feasible solution to MILP(θ), we increase the time limit to 48,000 ticks (roughly 60 seconds) and the upper cutoff limit of the objective value. If there is no improvement in the optimal objective value of the Master Problem for a while (see Algorithm 2), we increase the number of grid points and add more cuts per iteration. Furthermore, in order to control the number of cuts presented in the Master Problem, we delete all cuts that are nonbinding in the incumbent solution every five iterations.

Global time limit. The whole process is terminated if the entire computation time exceeds a global time limit.

Details of the iterative procedure are described in Algorithm 2

Algorithm 2: Overall cut generation loop

---

1 Initialization.
   Choose initial parameter grid size t, upper objective value cutoff limits $\gamma_1 < \gamma_2 < 0$, deterministic time limits $\tau_1 = 800$ ticks, $\tau_2 = 48{,}000$ ticks. Set iteration counter $\texttt{Iter} = 0$. Denote k the number of blocks in a given $\texttt{DB}$-k from; if no decomposition is available, set $k = 1$.
2 $\texttt{TimeLimit} \leftarrow \tau_1$, $\texttt{Cutoff} \leftarrow \gamma_1$.
3 $\texttt{Iter} \leftarrow \texttt{Iter} + 1$. Solve (MP) and obtain optimal solution $\hat{x}$. Denote n the number of consecutive iterations where no improvement in the optimal objective value is made. Delete nonbinding cuts if necessary.
4 **if** $n = 100$ **then return** $\hat{x}$.
5 Update parameter grid size in the current iteration.
   if $0 \leq n \leq 39$ **then** $s \leftarrow 2^{\lfloor 0.1n \rfloor} t$. **else** $s \leftarrow 16t$.
6 Set parameter grid Θ uniformly with $|\Theta| \leftarrow s$.
7 Separation.
   for $j = 1, \ldots, k$ **do** Generate a set $\mathcal{K}^{(j)}$ of cuts following $\mathrm{CutGen}(\hat{x}, \Theta, \texttt{Cutoff}, \texttt{TimeLimit})$ for block j.
8 **if** $\bigcup_{j=1}^{k} \mathcal{K}^{(j)} \neq \emptyset$ **then** Add cuts to (MP), **go to** 2.
9 **else if** $\texttt{TimeLimit} = \tau_1$, $\texttt{Cutoff} = \gamma_1$ **then** $\texttt{TimeLimit} \leftarrow \tau_2$, **go to** 7.
10 **else if** $\texttt{TimeLimit} = \tau_2$, $\texttt{Cutoff} = \gamma_1$ **then** $\texttt{Cutoff} \leftarrow \gamma_2$, **go to** 7.
11 **else if** $\texttt{TimeLimit} = \tau_2$, $\texttt{Cutoff} = \gamma_2$ **then return** $\hat{x}$.

---

4 Computational experiments

In this section we first discuss the practical setup for our experiments, then present our computational results. We implemented our code in C, with IBM ILOG CPLEX 12.7.1 as black-box MIP and LP solver. The computations were conducted on an assortment of machines with `x86_64` architecture CPUs. In order to ensure reproducibility, all machines used the same single-threaded binary code, and all time limits made use of CPLEX's *deterministic time* feature, aside from the global time limit.

4.1 Choice of model parameter values

The values of various model parameters used in the computation are summarized in Table 1. An asterisk (*) indicates that the parameter does not apply to all experiments. We also present below a brief motivation for our choices.

measure	**parameter**	**value**
maximum number of nonzero components in π (*)	M	10
bounds on $\lvert\pi_j\rvert$, $1 \leq j \leq p$ (*)	U	1 or 100
initial number of grid points (without `DB`-k form)	t	80
initial number of grid points (with `DB`-k form)	t	20
normalization constant	κ	10^4
minimum nonzero objective coefficient	δ	10^{-4}
upper cutoff limits of objective value	γ_1, γ_2	$-10^{-3}, -10^{-5}$
minimum cut violation	ϵ	10^{-6}
fractionality bound	σ	0.025

Table 1: Model parameter values used in computation

In their experimental analysis, Balas and Saxena [6] noted that the split disjunctions they computed generally featured two interesting characteristics. Although not being intentionally restricted,

(i) most split disjunctions had a support of size between 10 and 20, irrespective of the size of the problem; and

(ii) most split disjunctions did not have very large coefficients, with the average coefficient size per iteration typically being less than 5.

We chose the sparsity parameter of $M = 10$ to reflect the lower end of that spectrum. When attempting to limit the size of the split coefficients, we chose bounds $U = 1$ (i.e., $-1 \leq \pi_j \leq 1$, for all $1 \leq j \leq p$) since these would be the simplest splits obtainable. When only sparsity constraints were enforced, we set $U = 100$ (i.e., $-100 \leq \pi_j \leq 100$, for all $1 \leq j \leq p$) to allow for splits with somewhat larger coefficients.

At each iteration, the initial parameter grid size depends on whether a DB-k form of the constraint matrix is supplied or not. If no DB-k form is given, we set $t = 80$, and 80 MILP(θ) are processed; if a decomposition is given, then we set $t = 20$ for each of the k blocks, and therefore $20k$ MILP(θ) are processed in total.

For the fractionality bound σ on the set of split disjunctions, a natural value could be, for example, the integrality tolerance 10^{-6}. Although more split cuts may be obtained by using such a loose bound, we impose a rather strict 0.025 bound instead. In practice, this led to more gap closed per iteration on average and more gap closed overall within our time limits. Adding a fractionality bound also helped preventing MILP(θ) from yielding unviolated split disjunctions due to numerical errors.

4.2 Computational results

4.2.1 First experiment: How does our implementation compare with the best available results?

To check whether our implementation was reasonable, we first tested it on the MIPLIB 3.0 [10] instances, in a configuration where it approximates a straightforward split cut separator, i.e., without any sparsity or structure constraints on the split disjunctions. Artificial lower and upper bounds ± 100 are applied on the disjunction coefficients ($U = 100$), which allows for a reasonably large subset of all disjunctions to be considered. The entire computation time for each instance is limited to 24 hours, including the time taken to check cut validity[1]. At termination, we measure the final percentage of integrality gap closed, which is then compared with the best of the bounds given in [6] and [17]. For each instance, we look at that percentage of gap closure divided by the best of the analogous results in [6] and [17]. We do this comparison on the 57 instances where the best known gaps are strictly positive. Table 2 shows the number of instances that fall within various categories based on this ratio. In particular, on 8 instances, we closed more gap than the best available result, and on 25 instances we closed at least 90% relative gap.

On the other hand, we closed less than 1% relative gap on 27 instances. As shown in Figure 2 where each dot represents an individual instance, those are generally the instances that have the most integer variables. While there are many plausible explanations for our poor performance in this large set of instances, an important one is that the parameters in our code were not fine-tuned for this experiment, but rather for the experiments considering

[1]Note that neither [6] nor [17] have any cut validity procedure and also that [6] has no time limit on their experiments

relative gap closed	# instances
$> 100\%$	8
$\geq\ \ 99\%$	21
$\geq\ \ 90\%$	25
$\geq\ \ 50\%$	28
$<\ \ \ 1\%$	27

Table 2: Gap closed as a percentage of the best known gap closure (from [6] and [17])

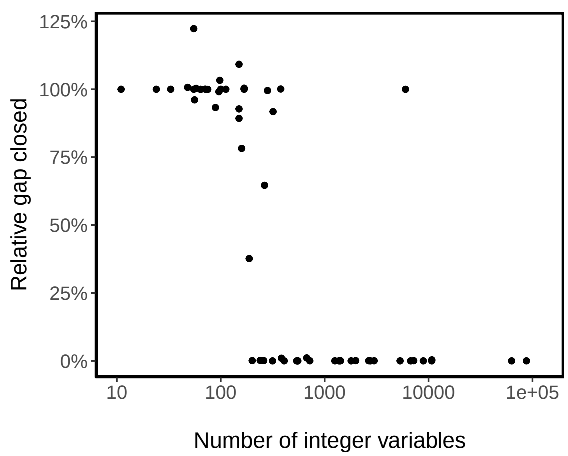


Figure 2: Gap closed as a percentage of the best known gap closure (from [6] and [17]), vs. number of integer variables

sparsity. When changing the values of parameters such as the number $|\Theta|$ of grid points and the fractionality bound σ, we were able to close significantly more gap on these 27 instances. However, the purpose of this experiment was just to determine if our implementation was reasonable compared to other ones, which seems to be the case, as further evidenced by the experiments on subsequent sections.

4.2.2 Second experiment: How does sparsity help?

In this section, we evaluate the relative strength of split cuts (i) whose split disjunctions are sparse and (ii) whose split coefficients are also small. We ran our implementation again on the MIPLIB 3.0 instances, first with the additional sparsity constraint obtained by setting $M = 10$. Then, we additionally considered ± 1 bounds on the disjunction coefficients (that is, setting $U = 1$). As was the case earlier, a time limit of 24 hours was set for all computations. Table 3 shows the details of our results. The first column of the table shows the best gap given in [6] and [17], followed by results obtained with arbitrary disjunctions, sparse disjunctions, and sparse disjunctions with ± 1 bound, respectively.

The last column in each setting shows the percentage of the total computation time that was spent checking cut validity. Observe that on a few large instances, the time spent on checking took most of the computation time. For example, in computing the gap closed by sparse disjunctions with ± 1 bounds on the instance `air04`, of the 24 hours spent, only 8% contributed to the actual computation. The remaining 92% was all dedicated to the verification of cut validity. We should thus expect that the bound obtainable on these large instances should be greater than the result shown in Table 3, had we chosen a longer time limit. Nonetheless, by restricting ourselves to split disjunctions with at most 10 nonzero coefficients, we still obtained significantly better results in terms of relative gap closed on instances that have a large number of integer variables, as opposed to the poor performance we observed with arbitrary disjunctions.

Besides allowing for more gap closed in less time, another related interesting effect to observe is the sparsity of the cuts produced. Observing that sparse disjunctions do not necessarily lead to sparse cuts, Figure 3 compares the densities of cuts (i.e., proportion of cut coefficients that are nonzero) obtained from different sets of split disjunctions. For each of the 60 MIPLIB 3.0 instances, we computed the average cut density by considering all the cuts that were used to obtain the results in Table 3. This resulted in 60 average cut densities for each set of split disjunctions. We then plot the distribution of these average cut densities in Figure 3. The horizontal lines in the figures represent the range of densities (with outliers omitted), the rectangles represent the 25-75 percentile interval and the solid vertical line represents the median. We consider as "outliers" cuts that are extremely dense, as determined by the following. Let r be the difference in density between the 25th and 75th percentile. Any cut with density of more than $1.5r$ above the 75 percentile is considered an outlier. Observe that sparse disjunctions did indeed lead to sparser split cuts in general: While the median density was 0.332 with arbitrary disjunctions, it dropped to 0.116 with sparse ones, and 0.103 with sparse ± 1 disjunctions.

To better evaluate the strength of the split cuts in the most restricted experiment (disjunctions with at most 10 nonzero ± 1 coefficients), we extended the time limit to a week and recomputed the gap closed on MIPLIB 3.0. The resulting average integrality gap is 68.4%, accounting for 91% of the 75.2% average for the best in [6] and [17]. Figure 4 shows a breakdown of the 57 instances whose best gap is strictly positive, according to the relative gap closed in this case. Surprisingly, we lost almost nothing (at most 2%) on more than half of MIPLIB 3.0 instances. Furthermore, we closed at least 90% relative gap on more than two thirds of the instances.

Our conclusion from this experiment is twofold. First, split cuts based on sparse disjunctions with small coefficients are almost as strong as general split cuts. Secondly, they tend to be sparser.

Best gap [6], [17]	Instance	$M = +\infty$, $U = 100$ Gap closed	# cuts binding	Time (s)	% time checking	$M = 10$, $U = 100$ Gap closed	# cuts binding	Time (s)	% time checking	$M = 10$, $U = 1$ Gap closed	# cuts binding	Time (s)	% time checking
100.00	`10teams`	0.00	6697	86400	28.97	73.21	2339	86400	37.42	70.05	682	86400	23.28
100.00	`air03`	0.37	334	86400	97.91	100.00	160	147	87.68	100.00	318	127	85.90
91.23	`air04`	0.00	2025	86400	98.09	32.90	298	86400	85.78	71.89	497	86400	91.94
61.98	`air05`	0.04	382	86400	97.73	35.88	364	86400	78.08	62.06	363	86400	63.69
83.95	`arki001`	0.00	2300	86400	94.09	65.95	272	86400	0.29	40.47	236	86400	0.50
99.60	`bell3a`	99.64	97	86011	0.03	74.64	112	4217	0.04	74.99	138	368	0.29
92.95	`bell5`	93.26	379	81961	0.00	92.78	144	70243	0.00	92.57	166	1582	0.03
46.52	`blend2`	30.07	166	6898	2.67	38.83	79	12991	0.03	42.64	93	1568	0.17
65.17	`cap6000`	65.16	554	86400	0.62	63.92	30	3044	0.77	58.37	36	1438	0.82
0.22	`dano3mip`	0.00	980	86400	74.50	0.00	493	86400	84.23	0.19	384	86400	64.80
8.20	`danoint`	7.88	466	86400	17.92	7.25	249	86400	2.87	8.15	354	86400	3.09
100.00	`dcmulti`	99.96	287	86400	0.72	99.99	254	84971	0.02	99.85	299	13670	0.06
100.00	`egout`	100.00	229	485	0.49	100.00	236	795	0.11	100.00	176	131	0.22
19.08	`fast0507`	0.00	754	86400	98.55	0.00	331	86400	97.84	0.42	265	86400	54.98
99.68	`fiber`	0.02	460	12533	33.98	29.49	220	86400	0.04	64.74	294	86400	0.04
99.75	`fixnet6`	99.84	537	86400	0.08	99.72	573	86400	0.03	99.85	330	86400	0.02
100.00	`flugpl`	100.00	125	659	0.03	100.00	125	629	0.03	100.00	108	3	3.33
100.00	`gen`	89.28	500	40720	11.35	98.09	422	13816	0.69	100.00	484	393	5.95
99.70	`gesa2`	0.01	358	524	12.21	85.93	219	86400	0.03	99.69	226	86400	0.02
99.97	`gesa2_o`	0.03	502	859	7.69	72.05	275	86400	0.07	93.00	228	86400	0.02
95.81	`gesa3`	0.94	516	27194	10.58	87.32	460	10802	0.91	95.98	343	86400	0.02
95.20	`gesa3_o`	1.04	378	56047	10.73	94.77	275	86400	0.13	95.99	223	86400	0.02
98.38	`gt2`	37.05	2723	86400	1.29	93.34	273	86400	0.00	92.01	107	21716	0.00
58.48	`harp2`	0.02	175	270	17.51	22.79	169	9040	0.08	42.73	173	86400	0.02
100.00	`khb05250`	100.00	379	275	2.25	100.00	318	1607	0.46	100.00	368	305	1.54
95.20	`l152lav`	0.08	476	10562	86.43	31.04	197	86400	4.01	41.89	188	86400	1.54
93.75	`lseu`	87.45	150	86400	0.01	69.08	65	86400	0.00	74.15	74	86400	0.00
14.02	`mas74`	15.31	161	86400	0.72	10.36	47	86400	0.00	11.64	48	86400	0.00
26.52	`mas76`	24.60	129	86400	0.31	12.08	51	86400	0.00	13.99	60	86400	0.00
51.70	`misc03`	40.44	252	86400	11.87	49.67	124	86400	0.03	51.44	211	67654	0.01
100.00	`misc06`	100.00	268	287	5.12	100.00	275	321	4.36	100.00	132	44	13.18
20.11	`misc07`	0.02	1845	86400	20.02	15.79	204	86400	0.30	14.39	206	86400	0.07
100.00	`mitre`	0.00	4444	86400	6.08	7.25	1504	86400	0.56	0.27	1547	86400	0.37
36.16	`mkc`	0.00	5901	86400	12.85	28.85	477	86400	1.37	53.98	574	86400	1.68
99.98	`mod008`	91.74	460	86400	1.10	51.70	124	86400	0.00	52.41	132	18113	0.00
100.00	`mod010`	0.05	2738	86400	49.87	71.15	209	86400	3.44	100.00	403	9843	4.75
72.44	`mod011`	71.79	741	86400	2.70	67.96	958	86400	1.27	72.65	899	86400	1.20
92.18	`modglob`	95.21	376	1592	0.49	96.44	288	25457	0.03	94.43	220	86400	0.00
100.00	`nw04`	0.01	350	86400	99.67	41.29	439	86400	99.19	78.46	343	86400	17.89
87.42	`p0033`	87.42	135	37162	0.00	82.26	39	86400	0.00	83.13	140	65633	0.00
74.93	`p0201`	0.07	513	36	36.66	66.95	136	86400	0.10	70.83	151	86400	0.04
99.99	`p0282`	99.51	117	86400	0.19	98.69	131	86400	0.00	98.32	119	86400	0.00
99.42	`p0548`	0.00	6407	47069	1.17	92.90	337	86400	0.01	95.14	343	86400	0.00
99.90	`p2756`	0.00	5771	86400	1.78	83.88	258	86400	0.06	88.31	419	86400	0.05
0.00	`pk1`	0.00	6328	86400	0.26	0.00	341	86400	0.00	0.00	356	86400	0.00
97.03	`pp08a`	97.01	136	86400	0.04	97.03	184	86400	0.00	97.05	168	86400	0.00
95.81	`pp08aCUTS`	95.78	160	86400	0.18	95.68	152	86400	0.00	95.81	155	86400	0.00
77.51	`qiu`	78.05	330	86400	3.00	78.04	345	86400	2.61	78.02	307	86400	0.82
100.00	`qnet1`	0.03	570	784	70.89	70.90	246	86400	0.18	100.00	299	11525	0.23
100.00	`qnet1_o`	0.04	436	275	33.92	95.29	261	86400	0.01	100.00	247	73048	0.01
23.40	`rentacar`	28.62	319	27635	9.92	32.62	256	69737	7.77	9.46	198	10352	22.90
100.00	`rgn`	100.00	474	69983	0.18	74.11	142	86400	0.00	74.64	194	86400	0.00
70.70	`rout`	0.00	3860	86400	5.07	43.45	223	86400	0.29	60.81	246	86400	0.23
89.74	`set1ch`	0.16	514	86400	2.32	89.76	421	51807	0.00	89.75	230	58286	0.00
61.52	`seymour`	0.00	3428	86400	41.11	0.03	350	86400	22.90	16.49	179	86400	12.36
0.00	`stein27`	0.00	255	15471	0.00	0.00	213	11433	0.00	0.00	182	6474	0.01
0.00	`stein45`	0.00	4499	86400	0.41	0.00	2595	86400	0.34	0.00	3850	86400	0.32
33.93	`swath`	0.00	5782	86400	68.18	10.19	264	86400	8.55	31.78	249	86400	2.13
100.00	`vpm1`	100.00	262	65472	0.04	95.69	161	86400	0.00	100.00	435	790	0.24
81.05	`vpm2`	81.37	220	86400	0.04	81.17	188	86400	0.00	81.38	206	86400	0.00
75.17	average	36.99	1352	60246	21.54	60.17	348	68104	10.58	65.60	343	60771	8.01

Table 3: Gap closed for the (i) full split closure, (ii) sparse split cuts only, and (ii) sparse ± 1 split cuts only.

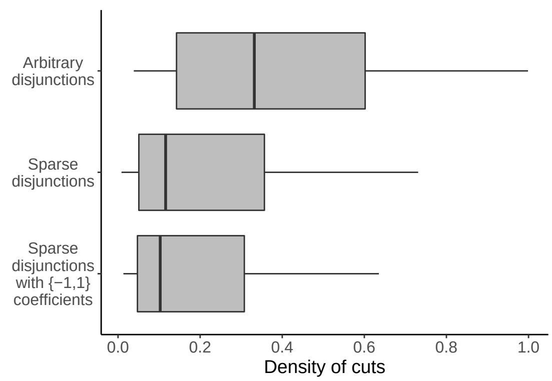


Figure 3: Distribution of cut densities with different experimental settings.

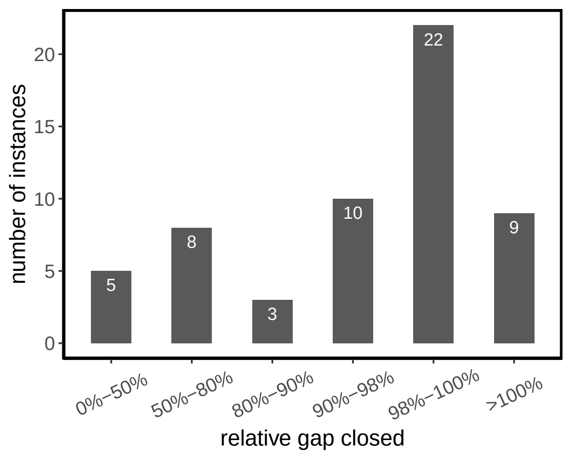


Figure 4: Distribution of gap closed with time limit of one week.

4.2.3 Third experiment: How does structured sparsity help?

Problem-specific DB-k forms provide a natural way to exploit sparsity. The potential advantages of generating split disjunctions whose support lies entirely within individual blocks are to produce split cuts that are both sparse and mutually orthogonal—two vital characteristics that make a cut effective. Moreover, working with small blocks in a DB-k decomposition may potentially reduce the computational time required to find a violated cut. On the other hand, restricting ourselves to such a narrow class of cuts can result in a much weaker cut family. The experiments in this section were designed to try and quantify these tradeoffs.

We use GCG 2.1.1 [24] as a black-box tool to generate the required DB-k forms on MI-

PLIB 3.0 instances, and then implement our model with the additional structure constraint (Con3) on the disjunctions, as described in Section 3. Furthermore, for comparison purposes,

- we have kept the sparsity parameter $M = 10$ and coefficient bound $U = 1$ on split disjunctions;
- for each instance with a given decomposition, we adhered to that decomposition in all iterations, i.e., we didn't change the structural requirement on disjunctions from one iteration to another;
- we ignored all linking constraints and linking variables by setting the corresponding multipliers to zero;
- the time limit was set to one week.

Table 4 shows the final gap closed by restricting split disjunctions with the structures given by `DB`-k forms for $k = 2, 3, 4, 5$ (GAPk). The first four columns of Table 4 are the result from previous section, with the same one week time limit, obtained by using disjunctions with $M = 10$ and $U = 1$ but no structure constraint (GAPnodb). The last column represents the highest gap closed between all `DB`-k forms. We removed from the table three instances where the gap closed without `DB`-k was zero (`pk1,stein27,stein45`) and eight instances where no `DB`-k form was found for any $k \in \{2, 3, 4, 5\}$ (`air03`, `cap6000`, `mas74`, `mas76`, `mod008`, `nw04`, `p0033`, `rentacar`).

Note that the set of split cuts we used to obtain the results on Table 4 is extremely restrictive: (i) the corresponding disjunctions have at most 10 nonzero coefficients which are either 1 or -1, and (ii) the cuts are obtained by aggregating only rows and columns that belong to a single block in a `DB`-k form. Despite being so selective, these cuts close a significant amount of gap in most cases. In fact, of the 49 instances left, the average gap closed without `DB`-k is 75% and the best gap closed among all `DB`-k is 58%.

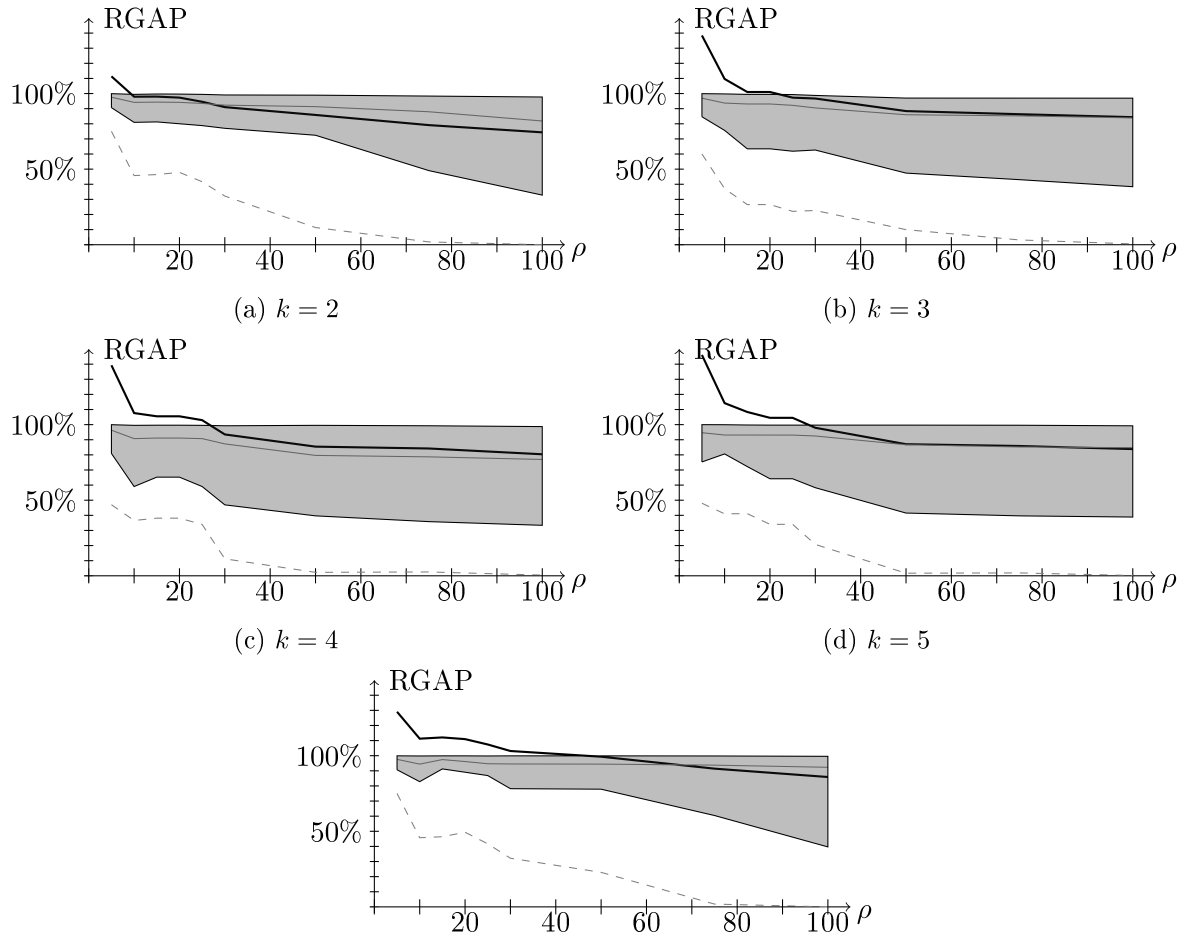
While the above averages already indicate that the disadvantage of using `DB`-k forms does not seem to be too big in terms of gap closed, it seems that using `DB`-k decompositions may not always pay off. To try and discard bad decompositions, we filtered the results in Table 4. The results are summarized in Figure 5. For a given `DB`-k decomposition and a value of ρ, we first removed from the `DB`-k results the ones obtained from a decomposition where either the percentage of linking constraints or variables were above ρ percent. Then, for each remaining instance, we computed the relative gap closed (RGAP) as:

$$\frac{\text{GAP}k}{\text{GAPnodb}} \tag{RGAP}$$

The following statistics are shown in Figure 5 for the instances remaining after filtering:

Instance	Without DB-k				DB-2				DB-3				DB-4				DB-5				Best
	Gap closed	# cuts binding	Time (h)	% time checking	Gap closed	# cuts binding	Time (h)	% time checking	Gap closed	# cuts binding	Time (h)	% time checking	Gap closed	# cuts binding	Time (h)	% time checking	Gap closed	# cuts binding	Time (h)	% time checking	gap (DB-k)
10teams	95.35	1179	168.00	14.28	100.00	1576	168.00	3.82	100.00	1706	168.00	5.56	100.00	1282	168.00	1.83	100.00	1292	168.00	1.80	100.00
air04	92.70	651	168.00	37.22	30.42	134	43.12	15.46	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	30.42
air05	63.36	187	168.00	18.39	0.00	0	0.62	0.00	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00
arki001	40.56	321	168.00	0.11	32.70	157	10.30	0.14	32.66	332	53.69	0.08	32.30	159	1.44	0.41	32.31	196	4.07	0.50	32.70
bell3a	74.99	138	0.14	0.37	70.74	75	0.00	5.00	70.72	60	0.00	5.00	70.74	60	0.00	10.00	70.66	56	0.00	5.00	70.74
bell5	92.57	166	0.49	0.06	91.70	75	0.03	0.17	91.94	79	0.01	0.97	84.67	77	0.01	0.38	86.83	58	0.00	1.25	91.94
blend2	42.64	93	0.48	0.15	19.79	16	0.00	1.25	19.79	15	0.00	5.00	19.79	17	0.00	0.00	19.79	12	0.00	0.00	19.79
dano3mip	0.29	539	168.00	16.68	0.22	211	168.00	29.17	0.27	270	168.00	28.80	0.33	320	168.00	28.90	0.31	307	168.00	19.93	0.33
danoint	9.02	398	168.00	1.01	0.93	63	2.81	0.64	0.79	57	0.31	3.89	0.45	60	0.15	4.90	0.70	61	0.03	28.30	0.93
dcmulti	99.85	299	3.33	0.09	93.86	204	13.56	0.01	73.88	163	10.50	0.01	76.06	149	2.91	0.04	64.74	118	3.34	0.01	93.86
egout	100.00	176	0.07	0.25	94.53	114	0.01	0.67	92.20	106	0.00	4.00	89.75	141	0.00	3.33	93.12	87	0.00	2.22	94.53
fast0507	4.76	313	168.00	53.19	0.00	0	0.01	0.00	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00
fiber	76.68	229	168.00	0.01	62.76	179	168.00	0.01	72.71	188	139.54	0.00	24.22	98	55.60	0.00	15.89	52	9.93	0.01	72.71
fixnet6	99.87	359	165.27	0.00	85.64	300	1.43	0.06	82.43	359	0.01	4.19	84.65	314	0.12	0.63	84.19	412	0.02	3.06	85.64
flugpl	100.00	103	0.00	3.33	90.16	15	0.00	0.00	65.03	13	0.00	0.00	7.05	4	0.00	0.00	45.83	4	0.00	0.00	90.16
gen	100.20	484	0.08	5.97	100.20	282	0.15	2.72	100.20	298	0.08	5.05	100.20	327	0.03	13.15	100.20	468	0.04	8.38	100.20
gesa2	99.98	208	91.63	0.01	99.57	171	16.60	0.03	99.56	169	1.87	0.09	99.50	94	15.27	0.01	99.57	197	14.92	0.01	99.57
gesa2_o	97.47	236	168.00	0.01	38.67	108	16.01	0.00	38.66	130	20.46	0.00	38.65	183	5.04	0.01	38.64	134	11.80	0.01	38.67
gesa3	96.03	232	44.06	0.01	95.94	247	19.87	0.01	95.86	234	27.34	0.01	95.94	269	53.90	0.01	95.94	277	45.17	0.01	95.94
gesa3_o	96.05	195	68.67	0.01	95.86	180	26.36	0.01	95.95	194	71.99	0.00	46.54	164	2.35	0.02	59.77	240	1.37	0.06	95.95
gt2	92.01	182	15.41	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00
harp2	44.68	177	57.42	0.02	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	NaN	0.00
khb05250	100.00	368	0.13	1.75	77.73	38	0.00	1.18	77.73	38	0.00	1.25	77.73	38	0.01	1.36	77.73	38	0.00	1.25	77.73
l152lav	45.68	170	168.00	0.28	0.57	21	1.24	0.04	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.57
lseu	74.21	77	57.50	0.00	52.88	73	0.00	0.91	19.87	46	0.00	0.00	38.58	26	0.00	0.00	4.21	5	0.00	0.00	52.88
misc03	51.44	211	15.18	0.01	0.00	6	0.00	NaN	0.00	0	0.00	0.00	0.00	0	0.00	NaN	0.00	0	0.00	NaN	0.00
misc06	100.00	184	0.03	14.35	91.18	45	0.00	11.67	26.55	29	0.00	6.00	71.44	50	0.00	10.00	97.87	58	0.00	8.89	97.87
misc07	15.02	323	147.32	0.02	0.45	114	0.34	0.57	0.00	0	0.00	0.00	0.00	0	0.00	NaN	NA	NA	NA	NA	0.45
mitre	9.34	1896	168.00	0.27	48.45	3372	168.00	0.61	79.74	2606	168.00	0.62	80.70	7548	168.00	1.16	78.63	4527	168.00	1.16	80.70
mkc	66.60	1190	168.00	1.03	68.14	478	168.00	0.23	68.05	412	168.00	0.32	69.94	620	168.00	0.16	73.10	401	168.00	0.16	73.10
mod010	100.00	329	11.55	2.89	0.00	0	1.39	0.00	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00
mod011	84.95	1119	168.00	0.65	88.53	1110	168.00	0.93	92.77	1221	132.39	1.08	88.41	1329	168.00	1.07	87.15	1114	168.00	1.00	92.77
modglob	94.43	215	31.91	0.00	92.27	147	1.42	0.04	81.91	127	0.23	0.11	85.69	141	0.08	0.33	83.21	121	0.03	0.59	92.27
p0201	71.50	112	168.00	0.01	54.38	86	71.09	0.00	50.00	72	21.19	0.01	0.00	0	0.00	NaN	0.00	0	0.00	NaN	54.38
p0282	98.42	119	123.23	0.00	3.56	61	0.03	0.32	83.88	52	0.01	0.50	1.52	10	0.00	0.00	83.87	35	0.00	0.71	83.88
p0548	96.27	336	168.00	0.00	89.44	278	167.78	0.00	89.62	291	168.00	0.00	90.02	295	168.00	0.00	88.50	322	132.15	0.00	90.02
p2756	89.25	517	159.30	0.01	86.98	446	27.46	0.01	85.73	508	26.97	0.01	84.92	589	22.32	0.01	85.90	262	7.64	0.04	86.98
pp08a	97.04	186	41.72	0.00	95.62	183	0.14	0.06	94.16	197	0.01	1.07	95.51	191	0.01	1.05	95.20	196	0.01	1.90	95.62
pp08aCUTS	95.81	195	46.24	0.00	92.92	190	0.17	0.26	92.85	178	0.04	0.76	93.36	169	0.03	1.46	89.14	186	0.02	2.09	93.36
qiu	78.09	372	168.00	0.17	78.09	393	168.00	0.12	0.00	0	0.37	0.00	78.09	436	37.60	0.56	0.00	0	0.12	0.00	78.09
qnet1	100.00	307	4.04	0.15	2.45	34	1.78	0.00	1.77	23	1.88	0.00	2.28	50	3.15	0.00	1.91	27	0.28	0.01	2.45
qnet1_o	100.00	266	8.81	0.03	20.96	7	0.00	NaN	20.96	7	0.00	0.00	20.96	6	0.00	NaN	20.96	6	0.00	0.00	20.96
rgn	74.65	185	78.20	0.00	68.30	63	18.70	0.00	47.98	104	0.01	1.03	38.38	156	0.00	5.56	NA	NA	NA	NA	68.30
rout	68.32	269	168.00	0.07	41.49	380	29.62	0.10	42.25	467	29.21	0.08	40.34	391	17.98	0.12	69.64	407	26.14	0.11	69.64
set1ch	89.75	456	8.59	0.01	89.73	319	0.30	0.26	89.73	384	0.30	0.23	89.73	406	0.11	0.66	89.73	417	0.08	0.79	89.73
seymour	57.96	949	168.00	17.66	44.63	498	168.00	21.07	54.59	369	168.00	23.44	60.20	808	168.00	24.60	58.73	920	168.00	24.18	60.20
swath	33.19	276	168.00	0.47	18.86	87	0.09	6.33	16.68	94	0.04	7.03	11.28	84	0.01	23.60	12.81	87	0.01	30.00	18.86
vpm1	100.00	499	0.23	0.21	78.18	270	0.05	0.48	34.55	106	0.00	2.50	34.55	89	0.01	0.58	42.12	90	0.00	2.00	78.18
vpm2	81.36	191	31.31	0.00	75.12	150	0.64	0.03	72.22	143	0.10	0.14	54.77	85	0.02	0.59	66.37	173	0.01	0.82	75.12
average	75.35	371	89.88	3.90	54.38	264	37.08	2.13	50.13	242	31.56	2.22	46.52	352	28.45	2.79	47.25	273	25.82	2.98	58.13

Table 4: Gap closed for the full split closure, and for sparse ± 1 split cuts for DB-k only.



(a) $k = 2$

(b) $k = 3$

(c) $k = 4$

(d) $k = 5$

(e) best

Figure 5: Distribution of relative gap closed for `DB`-k forms for several values of ρ

- Average (bold line)
- 10-th percentile (dashed line)
- Median (solid line)
- 25-75th percentile (shaded region)

Note that, while in principle (RGAP) should be always at most 100%, due to time limits, it is possible that the result from (GAPnodb) is not as high as it should be, resulting in (RGAP) above 100%. The results in Figure 5 show that eliminating `DB`-k forms with a high number of linking variables or constraints is indeed a good indicator to filter out results where split cuts from DB-k form do not close too much gap.

The above results show that the gap loss is not too big when restricting ourselves to split cuts from DB-k form. We now try to understand how structured sparsity helps to produce more effective cuts. Table 5 shows the average support size in the first 100 disjunctions (of a given type) obtained by our implementation, and the corresponding average cut density, for instances 10teams, mkc, and seymour. We picked 10teams as an extreme example where, without utilizing a DB-2 structure, highly sparse disjunctions (8.9 nonzero entries, which accounts for only 5% of the 1800 integer variables) have produced almost completely dense cuts. Instance mkc and seymour were picked because they represent reasonably large instances that are also in MIPLIB 2003. We observe that, as expected, exploiting the DB-2 structure yields sparser cuts. Furthermore, the last row of Table 5 shows that disjunctions with arbitrarily many nonzero entries that are much denser still lead to sparse cuts when exploiting problem structure.

In Figure 6 we compare the distributions of average cut densities on the 40 MIPLIB 3.0 instances whose DB-2 forms have at most 50% linking variables or constraints. We picked DB-2 as a candidate for comparison because this is the simplest DB-k decompostion, having just 2 blocks. Other decompositions that contain more blocks all demonstrate a similar pattern. The 50% threshold was applied so that instances whose DB-k forms have a high number of linking variables or constraints are excluded from comparison. As discussed earlier, split cuts based on these decompositions are unlikely to close much gap, regardless of how sparse they are. The cut densities in category "Sparse disjunctions with {-1,1} coefficients" are computed based on the cuts obtained in the previous section, and the cut densities under "Structured sparse disjunctions" are computed based on the results with DB-2 forms. As seen in Figure 6, block structures lead to the sparsest cuts: Even comparing with the disjunctions ($M = 10, U = 1$) that previously led to the sparsest cuts, it further decreased the median of cut densities from 0.051 to 0.037, and the 75 percentile from 0.103 to 0.052.

Finally, we illustrate one more potential advantage of using split cuts based on DB-k forms. Figure 7 shows the evolution of the average gap closed in terms of runtime of our cut procedure and in terms of number of cuts added in our cut procedure. It can be seen that the split cuts obtained by using DB-k forms converge faster to a gap closer to the final gap both in terms of time (Figure 7a) and in terms of number of cuts (Figure 7c). The gain in terms of time is even more pronounced if we focus on large instances, that is, instances with at least 1000 variables, among which at least 50 are integer (Figure 7b). The grey lines labelled "DB-k*" in Figure 7 represent the average gap closed had we chosen for each instance the DB-k form that closes the most gap after adding up to 500 cuts. While it is hard to completely attribute these gains to a few factors only, we note that the most apparent difference between these cuts and those generated earlier is their higher degree of both sparsity and orthogonality.

Instance	Disjunction type	Average support size of disjunctions (#)	Average cut density (%)
`10teams`	$M = 10,\ U = 1$	8.9	95.6
`10teams`	$M = 10,\ U = 1$, with `DB`-2	8.3	63.9
`mkc`	$M = 10,\ U = 1$	9.8	13.0
`mkc`	$M = 10,\ U = 1$, with `DB`-2	9.6	3.6
`seymour`	$M = 10,\ U = 1$	9.4	9.1
`seymour`	$M = 10,\ U = 1$, with `DB`-2	8.6	4.4
`seymour`	$M = +\infty,\ U = 1$, with `DB`-2	206.7	9.0

Table 5: Disjunction and cut density for three example instances.

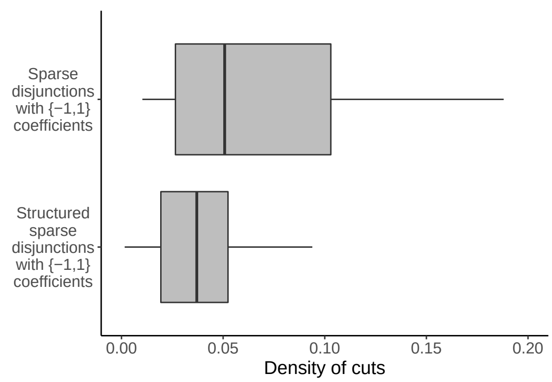
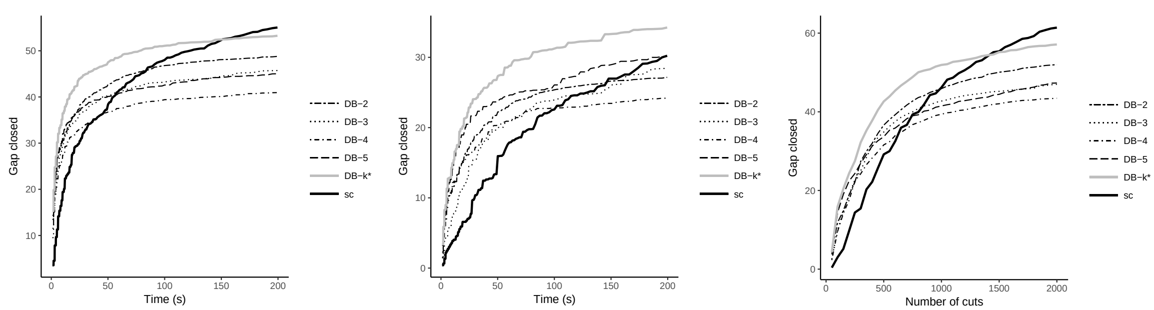


Figure 6: Distribution of cut densities for `DB`-2



(a) Average gap closed over time

(b) Average gap closed over time for large instances

(c) Average gap closed per cuts added

Figure 7: Evolution of average gap closed

4.2.4 Results on MIPLIB 2003 instances

Our final set of experiments was to run our code on larger instances than were previously available in the literature. For this purpose, we ran our code on MIPLIB 2003 [1] instances. However, since these instances are typically larger than the ones available in MIPLIB 3.0, we were able to run our code only using the parameters $M = 10$ and $U = 1$ and imposing a time limit of two weeks. Table 6 shows the results for those instances that are in MIPLIB 2003 but not in MIPLIB 3.0. Since there are no previous split closure numbers for those instances, we compare against the lift-and-project results of Bonami [12]. Compared to lift-and-project, significantly more gap can still be closed with the split closure approximation that does not exploit `DB`-k structure. Also, note that, even though the average results for `DB`-k based cuts are not as good, there are some instances where these results are significantly better than any of the other approaches, closing as much as 100% of the gap.

5 Conclusion

The main motivation for this work was to search for subsets of split cuts with promising computational properties. Our approach was to develop a tool that can empirically answer the following question: How much can we restrict the set of split cuts that we separate over, while retaining enough of the strength of the first split closure? While our tool is rather general (it can be seen as a continuation to Balas and Saxena's separation algorithm [6]), the specific restrictions that we explore aim at two desirable characteristics: First, we want sparse cuts, because they are beneficial to the linear algebra that underlies MIP solution methods. Secondly, we want cuts that are computed from different parts of the constraint matrix, and involve varied subsets of the variables. The latter point corresponds to generating cuts that are (approximately) mutually orthogonal, to as high a degree as possible, and it has been observed [6, 23] to be favorable in getting tighter relaxations with fewer cuts.

Our experiments show that explicitly enforcing sparsity of the split *disjunctions*, and bounding the magnitude of their coefficients, yields one such promising family of split cuts. We observe that the resulting cuts themselves are sparse too, which was expected but not a priori obvious. More surprisingly, even in an extreme setting where we only allow 10 nonzero disjunction coefficients with values ± 1, we obtain cuts that are 91% as effective as all split cuts together (in terms of gap closed, and compared to the best known results for the split closure [6, 17]). Note, for context, that were we to only allow one nonzero coefficient, we would obtain the lift-and-project closure of Balas, Ceria and Cornuéjols [4].

Next, in the same spirit of restricting the split disjunctions available to us, we exploit problem structure to impose static constraints on how cuts are generated. Specifically, we

L&P gap [12]	Str. L&P gap [12]	Instance	Without DB-k Gap closed	# cuts binding	Time (h)	% time checking	DB-2 Gap closed	# cuts binding	Time (h)	% time checking	DB-3 Gap closed	# cuts binding	Time (h)	% time checking	DB-4 Gap closed	# cuts binding	Time (h)	% time checking	DB-5 Gap closed	# cuts binding	Time (h)	% time checking	Best gap (DB-k)
78.76	78.76	`a1c1s1`	93.54	398	336.00	0.03	92.55	680	99.08	0.03	90.65	649	147.50	0.03	90.85	689	39.81	0.08	88.69	688	19.18	0.19	92.55
42.41	43.27	`aflow30a`	65.09	245	336.00	0.00	3.43	19	0.67	0.00	0.00	0	0.05	0.00	0.00	0	0.01	0.00	0.00	0	0.00	0.00	3.43
34.29	35.87	`aflow40b`	52.56	335	336.00	0.03	0.00	0	0.97	0.00	0.00	0	0.06	0.00	0.00	0	0.25	0.00	0.00	0	0.00	0.00	0.00
1.09	1.77	`atlanta-ip`	0.00	204	336.00	7.99	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.00	0.13	`glass4`	0.00	204	2.27	0.22	0.00	70	0.26	0.35	0.00	84	0.15	1.35	0.00	57	0.07	1.00	0.00	87	0.21	0.71	0.00
NA	NA	`manna81`	82.80	1944	336.00	0.03	96.84	1813	336.00	0.06	88.27	1613	336.00	0.15	100.00	1974	62.90	0.24	100.00	1796	34.64	0.29	100.00
44.88	45.15	`momentum1`	40.76	425	336.00	39.62	41.22	568	336.00	54.63	28.91	317	336.00	75.16	37.06	487	336.00	76.85	34.83	540	336.00	49.24	41.22
41.47	41.84	`momentum2`	65.68	824	336.00	64.68	26.28	200	25.55	70.23	14.01	278	43.26	51.01	27.59	478	30.57	64.42	17.92	523	22.55	65.76	27.59
42.23	44.65	`msc98-ip`	0.00	0	0.07	0.00	0.00	0	1.28	0.00	0.00	0	2.54	4.00	0.00	0	1.80	21.10	0.00	0	1.94	37.73	0.00
56.47	100.00	`mzzv11`	63.35	648	336.00	78.10	46.42	364	336.00	83.22	57.73	426	336.00	84.87	56.55	587	336.00	86.50	65.17	602	336.00	89.08	65.17
87.73	100.00	`mzzv42z`	79.92	612	336.00	77.71	43.22	400	336.00	84.71	91.73	483	336.00	86.99	71.99	739	336.00	84.86	91.18	827	336.00	80.45	91.73
22.73	22.71	`net12`	5.97	862	336.00	46.87	2.87	543	336.00	37.02	6.61	672	336.00	31.85	4.49	628	336.00	38.61	5.77	741	336.00	33.96	6.61
36.88	77.09	`nsrand-ipx`	65.38	1075	336.00	0.10	61.08	1057	336.00	0.09	50.25	843	336.00	0.06	52.82	907	336.00	0.10	49.61	878	336.00	0.08	61.08
0.19	26.32	`opt1217`	4.89	387	336.00	0.42	0.00	0	0.01	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00
10.29	10.83	`protfold`	37.41	2172	336.00	24.98	1.68	1055	336.00	29.50	1.14	1130	336.00	29.43	1.19	1273	336.00	32.47	1.29	1118	336.00	25.03	1.68
0.00	0.00	`rd-rplusc-21`	0.00	203	22.59	1.52	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
16.31	55.90	`roll3000`	37.90	504	291.02	0.57	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
42.06	59.91	`sp97ar`	65.04	533	336.00	0.54	47.62	410	336.00	1.29	41.47	355	336.00	1.15	41.12	318	336.00	1.40	25.91	267	336.00	1.58	47.62
26.99	42.45	`timtab1`	86.10	265	23.70	0.01	54.71	169	13.34	0.00	40.56	151	4.18	0.01	34.08	158	2.56	0.01	29.17	183	0.07	0.22	54.71
20.98	40.18	`timtab2`	84.67	430	283.14	0.01	55.39	297	21.38	0.02	46.24	279	23.24	0.01	31.92	275	26.04	0.01	31.40	270	5.42	0.01	55.39
64.12	64.12	`tr12-30`	88.05	680	30.95	0.01	85.60	676	5.99	0.05	84.60	699	3.75	0.06	85.01	679	0.64	0.29	84.04	692	0.75	0.23	85.60
31.90	42.43	average	48.53	617	255.13	16.35	31.38	396	136.03	17.20	30.58	380	138.70	17.43	30.22	440	119.84	19.43	29.76	439	116.04	18.31	34.97

Table 6: Gap closed in MIPLIB2003 instances.

start by computing block decompositions of our problems. Then, we force our split cut generator to use, for each cut, only constraints and variables from a single block. In a second series of experiments, we test this approach with arrowhead decompositions [8, 24] of the constraint matrices, while keeping the same limitations on the disjunctions as before. In this even more restricted setting, we observe a significant degradation of the average gap closure. However, we demonstrate that it is easy to determine *a priori* which instances will benefit from block decompositions, and which will not. With a very simple rule based on the number the linking constraints and variables, we are able to isolate the instances that are most suited for this technique. By using decompositions only when appropriate, we get a subset of instances on which, due to time limits, we close even more gap than without decomposition. Moreover, as a general rule, we observe that this setting lets us cut much more gap *per cut* on average. We attribute this desirable feature to the orthogonality of the cuts generated.

Overall, our results suggest that there exist small subsets of split cuts that exhibit advantageous properties, and that are yet to be exploited.

References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.

[2] Kent Andersen and Robert Weismantel. Zero-coefficient cuts. In Friedrich Eisenbrand and F. Bruce Shepherd, editors, *Integer Programming and Combinatorial Optimization*, pages 57–70, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[3] C. Aykanat, A. Pinar, and Ü. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25(6):1860–1879, 2004.

[4] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming*, 58(1–3):295–324, 1993.

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

[6] Egon Balas and Anureet Saxena. Optimizing over the split closure. *Mathematical Programming*, 113(2):219–240, 2008.

[7] Amitabh Basu, Pierre Bonami, Gérard Cornuéjols, and François Margot. On the relative strength of split, triangle and quadrilateral cuts. *Mathematical Programming*, 126(2):281–314, 2011.

[8] Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E. Lübbecke, Enrico Malaguti, and Emiliano Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1):391–424, 2015.

[9] Robert E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[10] Robert E. Bixby, Sebastián Ceria, Cassandra M. McZeal, and Martin W. P Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, (58):12–15, June 1998.

[11] Robert E. Bixby and Edward Rothberg. Progress in computational mixed integer programming - a look back from the other side of the tipping point. *Annals of Operations Research*, 149(02):37–41, 2007.

[12] Pierre Bonami. On optimizing over lift-and-project closures. *Mathematical Programming Computation*, 4(2):151–179, 2012.

[13] Pierre Bonami, Gérard Cornuéjols, Sanjeeb Dash, Matteo Fischetti, and Andrea Lodi. Projected Chvátal–Gomory cuts for mixed integer linear programs. *Mathematical Programming*, 113(2):241–257, 2008.

[14] Alberto Caprara and Adam N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming*, 94(2):279–294, Jan 2003.

[15] William J. Cook, Ravi Kannan, and Alexander Schrijver. Chvátal closures for mixed integer programs. *Mathematical Programming*, 47:155–174, 1990.

[16] Gérard Cornuéjols and Giacomo Nannicini. Practical strategies for generating rank-1 split cuts in mixed-integer linear programming. *Mathematical Programming Computation*, 3(4):281–318, 2011.

[17] Sanjeeb Dash, Oktay Günlük, and Andrea Lodi. MIR closures of polyhedral sets. *Mathematical Programming*, 121(1):33–60, 2010.

[18] Santanu S. Dey, Marco Molinaro, and Qianyi Wang. Approximating polyhedra with sparse inequalities. *Mathematical Programming*, 154(1):329–352, 2015.

[19] Santanu S. Dey, Marco Molinaro, and Qianyi Wang. Analysis of sparse cutting planes for sparse MILPs with applications to stochastic MILPs. *Mathematics of Operations Research*, 43(1):304–332, 2018.

[20] Matteo Fischetti and Andrea Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming*, 110(1):3–20, 2007.

[21] Matteo Fischetti, Andrea Lodi, and Andrea Tramontani. On the separation of disjunctive cuts. *Mathematical Programming*, 128(1):205–230, 2011.

[22] Matteo Fischetti and Domenico Salvagnin. A relax-and-cut framework for Gomory mixed-integer cuts. *Mathematical Programming Computation*, 3(2):79–102, 2011.

[23] Matteo Fischetti and Domenico Salvagnin. Approximating the split closure. *INFORMS Journal on Computing*, 25(4):808–819, 2013.

[24] Gerald Gamrath and Marco E. Lübbecke. Experiments with a generic dantzig-wolfe decomposition for integer programs. In Paola Festa, editor, *Experimental Algorithms*, pages 239–252, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[25] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[26] Uwe H. Suhl and Leena M. Suhl. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, 2(4):325–335, 1990.

[27] Matthias Walter. Sparsity of lift-and-project cutting planes. In Stefan Helber, Michael Breitner, Daniel Rösch, Cornelia Schön, Johann-Matthias Graf von der Schulenburg, Philipp Sibbertsen, Marc Steinbach, Stefan Weber, and Anja Wolter, editors, *Operations Research Proceedings 2012*, pages 9–14, Cham, 2014. Springer International Publishing.