Split Cuts From Sparse Disjunctions

by

Shenghao Yang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2019

Statement of Contributions

This thesis is based on the manuscript *Split cuts from sparse disjunctions*, by Ricardo Fukasawa, Laurent Poirrier and Shenghao Yang, which has been submitted to Mathematical Programming Computation and is currently under revision.

Abstract

Cutting planes are one of the major techniques used in solving Mixed-Integer Linear Programming (MIP) models. Various types of cuts have long been exploited by MIP solvers, leading to state-of-the-art performance in practice. Among them, the class of split cuts, which includes Gomory Mixed Integer (GMI) and Mixed Integer Rounding (MIR) cuts from tableaux, are arguably the most effective class of general cutting planes within a branch-and-cut framework. Sparsity, on the other hand, is a common characteristic of real-world MIP problems, and it is an important part of why the simplex method works so well inside branch-and-cut. A natural question, therefore, is to determine how sparsity can be incorporated into split cuts and how effective are split cuts that exploit sparsity. In this thesis, we evaluate the strength of split cuts that arise from sparse split disjunctions. In particular, we implement an approximate separation routine that separates only split cuts whose split disjunctions are sparse. We also present a straightforward way to exploit sparsity structure that is implicit in the MIP formulation. We run computational experiments and conclude that, one possibility to produce good split cuts is to try sparse disjunctions and exploit such structure.

Acknowledgements

First of all, I would like to thank my supervisors Dr. Ricardo Fukasawa and Dr. Laurent Poirrier, for the patient guidance and invaluable advice they generously provided throughout my time as their student. Both Ricardo and Laurent have been extremely caring and supportive. It would never have been possible for me to complete this work without their encouragement and dedication. I was very lucky to have the opportunity to work with and learn from them.

I would also like to thank my readers, Dr. Levent Tunçel and Dr. Bertrand Guenin, for their time to read my thesis and provide insightful feedbacks.

Last but not least, I would like to thank my parents, for their unconditional support, and my wife, Xiaoyu, for bringing so much joy and love into my life.

Dedication

To my wife, and our parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A *mixed-integer linear programming* (MIP) problem is a mathematical optimization problem where the objective and the constraints are linear and some or all of the variables are restricted to be integers. Many real-world problems in planning, transportation, telecommunications, economics and finance can be naturally formulated and solved as MIPs [40]. Formally, a MIP is stated as:

$$\begin{aligned} \min\ & c^\top x \\ \text{s.t.}\ & x \in P \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p}) \end{aligned} \tag{1.1}$$

where $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ is a rational polyhedron in $\mathbb{R}^n$, and $1 \leq p \leq n$.

Due to wide range of applications of MIP models in practice, numerous computer codes have been developed to solve optimization problems of form (1.1), including both commercial solvers like CPLEX [3], Gurobi [2], FICO Xpress [1], and non-commercial ones like SCIP [38]. Over the last 25 years, MIP solvers have accomplished remarkable progress, achieving a machine-independent speed-up of the solution process by more than a factor of 450,000 [18]. Among many other technical developments, general-purpose cutting plane methods are arguably the most important contributors to this progress (see, for example, [20]).

The focus of this thesis is on a particular class of cutting planes called *split cuts* and their computational properties. In the rest of this chapter we discuss the basic concepts and previous results that motivate our study. We begin by defining cutting planes for MIPs.

1.1 Cutting planes

A general solution technique for (1.1) is to use its *linear programming relaxation* along with *cutting planes*. The linear programming relaxation simply drops the integrality constraints on the variables and results in a linear program

$$\begin{aligned} \min \ & c^\top x \\ \text{s.t.} \ & x \in P \end{aligned} \tag{1.2}$$

which can be solved quickly in practice, for example, by the simplex method. The concept of cutting planes is fundamental in integer programming. We now formally define what cutting planes are.

Definition 1. A **valid inequality** for (1.1) is a linear constraint $\alpha^\top x \geq \beta$ that does not eliminate any feasible integer solutions. That is, for every $x \in P \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})$ we have $\alpha^\top x \geq \beta$. A valid inequality is also called a **cutting plane**, or **cut**.

Note that, some authors (e.g., [26]) require a cut, apart from being a valid inequality, to cuts off part of the feasible region of linear programming relaxation. In this thesis, we use cutting planes and valid inequalities interchangeably. We call a cut *violated* if it cuts off part of the feasible region of linear programming relaxation, and *non-violated* otherwise. Sometimes we also call a cut *valid* to emphasize that this cut is a valid inequality, and *invalid* to emphasis that, it cuts off part of the feasible region, but it is not a valid inequality (and thus it is not a cut).

Below we give a concrete example of a violated cutting plane.

Example 2. Consider a simple integer program of just 2 variables,

$$\begin{aligned} \max \ & \mathbf{1}^\top x \\ \text{s.t.} \ & \begin{bmatrix} -2 & 6 \\ 2 & 1 \end{bmatrix} x \leq \begin{pmatrix} 9 \\ 5 \end{pmatrix} \\ & x \in \mathbb{Z}^2_+. \end{aligned} \tag{1.3}$$

The feasible region of the linear programming relaxation of (1.3) is shown in Figure 1.1. It is a polyhedron on the plane with vertices $(0,0), (0,3/2), (3/2,2), (5/2,0)$. The linear inequality $5x_1 + 6x_2 \leq 16$ does not eliminate any feasible integer points, therefore it is a cut for (1.3). Furthermore, it cuts off the vertex $x^* = (3/2,2)$, and thus it is a violated cut. Note that solving the linear programming relaxation of (1.3) along with constraint $5x_1 + 6x_2 \leq 16$ readily yields the optimal integer solution!

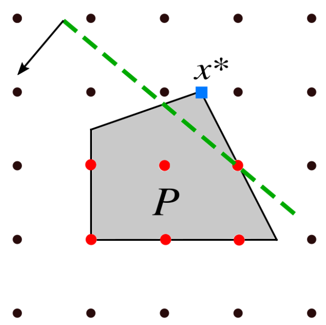


Figure 1.1: A cutting plane example

In fact, it turns out that the cut in Example 2 belongs to the family of split cuts introduced by Cook et al. [25], which has shown to be the most useful family of cuts to solve MIPs in practice [20]. We refer to [26] for a comprehensive review of several classical families of cuts and how they are related to each other.

A central concept in the study of cutting planes is that of closures. In the next section, we give a brief overview of known results on closures.

1.2 Closures

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, the associated closure is the convex set defined by the intersection of all cuts in the same family. Therefore, a closure can be seen as an approximation to the convex hull of all feasible integer solutions, or simply, the *integer hull*. On the theoretical side, topics range from determining polyhedrality of closures [25,45] and the complexity of optimizing a linear function over them [23,32], to analyzing relationships between different closures [28] and how well they approximate the integer hull [14]. On the computational front, several authors proposed strategies to empirically evaluate the strength of different closures by computing the amount of integrality gap (c.f. Definition 3) they close: we thus have computational evaluations of the Chvátal closure [33], the split closure [13], the projected Chvátal-Gomory closure [22], the MIR closure [29] and the lift-and-project closure [21].

Definition 3. Consider (1.1) and its linear programming relaxation (1.2). Suppose (1.1) is feasible and (1.2) is bounded below. Let Opt_{MIP} and Opt_{LP} denote the optimum of

(1.1) and (1.2), respectively. Given a family of cuts. Let $\mathcal{C}$ denote its closure, and let

$$\mathrm{Opt}_{CLO} := \min_{x \in \mathcal{C}} c^\top x.$$

Then the **integrality gap** closed by $\mathcal{C}$ is expressed as the quantity

$$\frac{\mathrm{Opt}_{CLO} - \mathrm{Opt}_{LP}}{\mathrm{Opt}_{MIP} - \mathrm{Opt}_{LP}}.$$

Usually, integrality gaps are computed individually on a set of benchmark instances, and the average gap closed by a given family of cuts is then compared with others. The Mixed Integer Programming LIBrary [4], or MIPLIB, is an electronically available library consisting of various real-world pure and mixed integer programs. Currently MIPLIB has five versions. The first version was initiated in 1992, and the sixth version was just released on November 5, 2018. Since its introduction, MIPLIB has become a standard test set used to compare the performance of mixed integer optimizers. In this thesis we will use MIPLIB 3.0 [19] and MIPLIB 2003 [5], which are the third and the forth versions of MIPLIB. We explain in more details about our choice of test sets in Chapter 4.

The split closure, or equivalently the MIR closure, was shown computationally to be a very tight approximation of the integer hull [13, 36]. On average it closes more than 75% of the integrality gap on MIPLIB 3.0 instances. This has since led to efforts on efficient generation of strong split cuts (see, for example, [7, 27, 35, 36]). We formally introduce split cuts in the next section.

1.3 Split disjunctions and split cuts

Consider an integer vector $\pi \in \mathbb{Z}^n$ where $\pi_j = 0$ for all $j \geq p+1$ and an integer $\pi_0 \in \mathbb{Z}$. We define a *split disjunction* for $P \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})$ as

$$\pi^\top x \leq \pi_0 \ \vee \ \pi^\top x \geq \pi_0 + 1. \tag{1.4}$$

Note that (1.4) is satisfied by every $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$. Therefore, let

$$\begin{aligned} \Pi_1^{(\pi,\pi_0)} &:= P \cap \{x \in \mathbb{R}^n : \pi^\top x \leq \pi_0\}, \\ \Pi_2^{(\pi,\pi_0)} &:= P \cap \{x \in \mathbb{R}^n : \pi^\top x \geq \pi_0 + 1\}, \end{aligned}$$

and $\bar{x}$ be any feasible integer solution of (1.1), then either $\bar{x} \in \Pi_1^{(\pi,\pi_0)}$ or $\bar{x} \in \Pi_2^{(\pi,\pi_0)}$.

Consider the convex hull of sets $\Pi_1^{(\pi,\pi_0)}$ and $\Pi_2^{(\pi,\pi_0)}$

$$P^{(\pi,\pi_0)} := \text{conv}\left(\Pi_1^{(\pi,\pi_0)} \cup \Pi_2^{(\pi,\pi_0)}\right),$$

where conv(S) denotes the smallest convex set that contains set S. Clearly $\bar{x} \in P^{(\pi,\pi_0)}$. We thus define split cuts as follows.

Definition 4. A **split cut** for $P \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})$ is a valid inequality for some $P^{(\pi,\pi_0)}$, where (π, π_0) is a split disjunction for $P \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})$.

Example 5. The cut $5x_1 + 6x_2 \leq 16$ in Example 2 is a split cut corresponding to the split disjunction $\pi = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $\pi_0 = 1$, as illustrated in Figure 1.2.

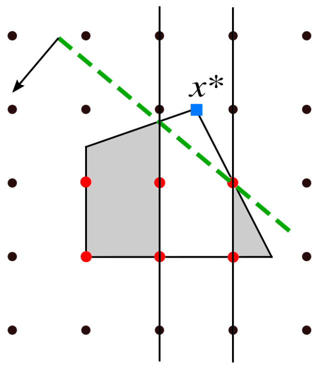
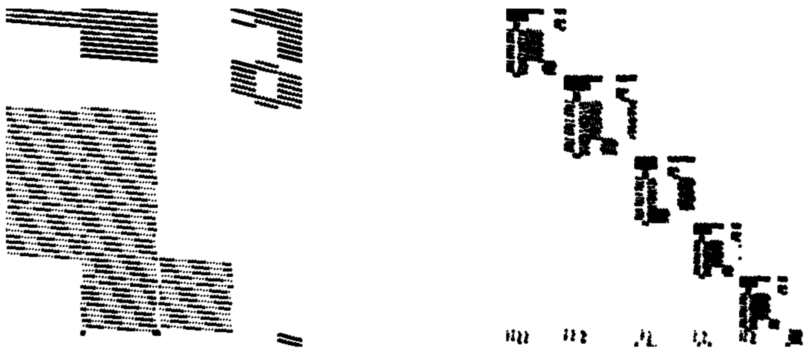


Figure 1.2: A split disjunction and a split cut

Obtaining effective violated cutting planes for MIPs is usually a nontrivial process. The problem of finding a cut that separates a fractional point from the convex hull of all feasible integer solutions, or showing that none exists, is called a *Separation Problem*. It has been shown that the separation problem for split cuts is $\mathcal{NP}$-hard [23] in general, and costly in practice [36].

MIPs are hard to solve in general, but there is an important property that underlies many real-world problems and plays in our favor: sparsity. We discuss it in the next section.



(a) less structured (b) more structured

Figure 1.3: Examples of sparse matrices

1.4 Sparsity

In numerical analysis and scientific computation, a matrix or a vector is considered *sparse* if most of the entires are zero, and it is *dense* if most of the entries are nonzero. Depending on the contexts, the term *sparsity* has been used in the literature with different meanings and emphases. Some authors refer to sparsity as the numeric quantity calculated from dividing the number of zero entries by the number of all entries, i.e., sparsity equals one minus density; some authors refer to sparsity as a general property of a matrix or vector being sparse. In this thesis, whenever we use the term sparsity, we mean the underlying matrix or vector is sparse. Sometimes the nonzero entries in a sparse matrix demonstrate a clear pattern. This is illustrated in Figure 1.3, where we take two sparse matrices, both have the same number of rows and columns, and plot their respective sparsity patterns by replacing each nonzero element with a black dot. Observe that most of the nonzero entries in the matrix on the right lie in five identifiable blocks, whereas the nonzero entries in the matrix on the left are distributed more arbitrarily. In fact, the more structured matrix in Figure 1.3b is obtained from the less structured one in Figure 1.3a by permuting its rows and columns. The advantage of having structured patterns in a sparse matrix will be made clearer in Chapter 2.

In general, sparsity helps when storing and manipulating matrices and vectors on a computer, as specialized algorithms and data structures may be designed to take advantage of their sparse structures. During a MIP solution process, sparsity may be exploited in several ways. In the following we mention a few that motivate the goal in this thesis.

First, many real-world MIP models have well-structured sparse constraint matrices that are amenable to decomposition-based solution approach. A recent result of Bergner et al. [15] shows 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 (e.g. Figure 1.3b). This shows that not only are these benchmark instances sparse (on average, MIPLIB 2010 [42] instances only have 1.62% density), but in many cases such sparsity has an identifiable structure that can be exploited.

Second, modern implementations of the simplex algorithm take advantage of sparsity in solving large linear systems [47], an approach credited as one of the two most notable improvements in the linear algebra routines of the simplex method [17]. Thus sparsity is a desirable property of cutting planes for MIPs. Indeed, in almost every cut generation and selection procedure described in the articles we mentioned in the previous sections, specific heuristics were 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 [33], applying a coefficient reduction algorithm to reduce the number of nonzero coefficients in the split cut [27], or discarding all dense cuts to ensure that only sparse cuts are added [35]. Additionally, in a recent computational study by Walter [48], it is shown that equivalent but denser versions of the same cuts negatively affect the 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 [30, 31].

Finally, in their computational study of the split closure [13], Balas and Saxena observed that most split disjunctions they produced were sparse, regardless of problem size. Although sparse split disjunctions do not necessarily lead to sparse split cuts, the two are not uncorrelated. On the other hand, although split closure provides a tight approximation to the integer hull, the time it takes to separate an arbitrary split cut makes it almost unrealistic to use in practice. Thus, given the possibility to exploit the sparsity underlying problem structures, and given the advantage of using sparser cuts, it is very interesting to evaluate the strength of split cuts based purely on sparse disjunctions, as a first step towards determining subsets of split cuts that are computationally more promising.

1.5 Motivations and outline

Split cuts have been shown to be strong theoretically—they dominate Chvatál-Gomory cuts, even on pure-integer sets [25]—and computationally [13, 29, 36]. Furthermore, even small subfamilies of split cuts, namely GMI and MIR cuts from tableaux, have proven extremely invaluable in practice [20]. On the other hand, finding a general split cut is hard both in theory [23] and in practice [36]. This motivates our search for subsets of split cuts, beyond GMI and MIR cuts from tableaux, with promising computational properties.

The main goal of this thesis is to study the strength of split cuts that exploit sparsity. Our contributions are the following. First, we implement an approximate separation routine based on the work on Balas and Saxena [13] that separates only split cuts whose split disjunctions are sparse and whose split coefficients are small. Second, we show empirically that in spite of these restrictions, the integrality gap closed by this subclass of split cuts is still quite significant compared to gap closed by the full split closure. Finally, we consider problem structures of individual instances and show that split cuts computed by considering only constraints and variables from a single block in an *arrowhead decomposition* [15,37] of the constraint matrix also largely preserve the strength of general split cuts, in terms of gap closed.

The focus of this work is computational, but the tools and results should be interesting to both practitioners and theoreticians. For example, although the separation problem for general split cuts is $\mathcal{NP}$-hard, finding a split cut arising from split disjunction with just one nonzero entry can be done in polynomial time [21]. Therefore, one might ask, how hard is it to optimize over all split cuts arising from split disjunction with support that has cardinality at most, say, two, five, ten? Furthermore, to begin with, is it even a meaningful question to think about? The results that we present in this thesis provide a positive answer: Split cuts from sparse disjunctions constitute a strong subclass of split cuts and indeed worth further study, both computationally and theoretically.

This thesis is organized as follows. In Chapter 2 we give a quick introduction to the cut generating linear program and the cut lifting procedure that are necessary for our separation routine. Then we lay out the basic approach of Balas and Saxena [13] for the separation of split cuts. Finally we introduce the automatic arrowhead decomposition of Bergner et al. [15]. In Chapter 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. Chapter 4 presents the results of our computational experiments.

Chapter 2

Background

In this chapter we review some basic results in linear and integer programming that will enable us to develop from first principles an approximate split cut separator for general MIPs. In particular, we begin by introducing the so-called Cut Generating Linear Program for split cuts, of which one variant plays a key role in our separator. Then we discuss a classical idea, called lifting, to obtain a cutting plane valid for a higher dimensional polyhedron from a given cutting plane valid for a lower dimensional one. It turns out that the lifting procedure significantly reduces computation time in practice and, as a result, makes the separation of split cuts realistic. With these basic technical tools, we present details of the approach of Balas and Saxena [13] for the separation of split cuts. Lastly, we give a concise overview of the method of Bergner et al. [15] to automatically detect structured sparsity in MIP instances.

2.1 Cut generating linear program

The idea that one can formulate the problem of finding a cutting plane valid for some given polyhedron as a linear program relies on Farkas' lemma. Farkas' lemma provides a simple characterization about the solvability of a system of linear inequalities. In the following we state three versions of Farkas' lemma that will be useful in our context. Proofs of Farkas' lemma can be found in various introductory texts, e.g., [24, Theorem 3.4] and [16, Theorem 4.6].

Theorem 6 (Farkas' lemma)**.** Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Then

$$\{x \in \mathbb{R}^n : Ax \le b\} = \emptyset \quad \iff \quad \{u \in \mathbb{R}^m : A^\top u = 0, b^\top u < 0, u \ge 0\} \neq \emptyset.$$

Corollary 7 (Farkas' lemma, other versions)**.** Let $A \in \mathbb{R}^{m\times n}, G \in \mathbb{R}^{l\times n}, b \in \mathbb{R}^m, d \in \mathbb{R}^l$. Then

$$\big\{x \in \mathbb{R}^n : Ax = b, x \geq 0\big\} = \emptyset \qquad \Longleftrightarrow \qquad \big\{u \in \mathbb{R}^m : A^\top u \leq 0, b^\top u > 0\big\} \neq \emptyset,$$

and

$$\begin{aligned} & \big\{x \in \mathbb{R}^n : Ax = b, Gx \geq d, x \geq 0\big\} = \emptyset \\ \Longleftrightarrow \quad & \big\{(u,v) \in \mathbb{R}^m \times \mathbb{R}^l : A^\top u + G^\top v \leq 0, b^\top u + d^\top v > 0, v \geq 0\big\} \neq \emptyset. \end{aligned}$$

Here is a direct consequence of Farkas' lemma.

Corollary 8. Suppose $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\} \neq \emptyset$, and $\min\{c^\top x : x \in P\} > -\infty$. Then $D = \{y \in \mathbb{R}^m : A^\top y \leq c\} \neq \emptyset$.

Proof. If $D = \emptyset$, then by Farkas' lemma, there is $\bar{u} \in \mathbb{R}^n$ such that $A\bar{u} = 0, c^\top \bar{u} < 0, \bar{u} \geq 0$. Let $\bar{x} \in P$, then $\bar{x} + \lambda\bar{u} \in P$ for all $\lambda \geq 0$, and $\lim_{\lambda\to+\infty} c^\top(\bar{x} + \lambda\bar{u}) = -\infty$, contradicting to our assumption that $\min\{c^\top x : x \in P\} > -\infty$. $\square$

Another useful result in linear programming is strong duality.

Theorem 9 (Strong duality)**.** Let $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ and $D = \{y \in \mathbb{R}^m : A^\top y \leq c\}$. If $P \neq \emptyset$ and $D \neq \emptyset$, then $\min\{c^\top x : x \in P\} = \max\{b^\top y : y \in D\}$, and there exist $x^* \in P$ and $y^* \in D$ such that $c^\top x^* = b^\top y^*$.

Recall that an inequality $\alpha^\top x \geq \beta$ is a valid split cut corresponding to the disjunction (π, π_0) only if it is satisfied by all points in $P^{(\pi,\pi_0)} = \operatorname{conv}\big(\Pi_1^{(\pi,\pi_0)} \cup \Pi_2^{(\pi,\pi_0)}\big)$, where $\Pi_1^{(\pi,\pi_0)} = P \cap \{x : \pi^\top x \leq \pi_0\}$, $\Pi_2^{(\pi,\pi_0)} = P \cap \{x : \pi^\top x \geq \pi_0 + 1\}$, and $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. Farkas' lemma enables us to certify the validity of $\alpha^\top x \geq \beta$ relative to $P^{(\pi,\pi_0)}$, as stated in Theorem 10. We give a proof that is similar to the proof of Theorem 3.22 in [24]. However, we note that the statement of Theorem 3.22 in [24] does not extend completely trivially to Theorem 10 that we present here. The slight technicality comes when one of $\Pi_1^{(\pi,\pi_0)}$ and $\Pi_2^{(\pi,\pi_0)}$ is empty. As we will see, it is easy to deal with.

Theorem 10. Suppose $P^{(\pi,\pi_0)} \neq \emptyset$. An inequality $\alpha^\top x \geq \beta$ is valid for $P^{(\pi,\pi_0)}$ if and only if there exist $y, z \in \mathbb{R}^m$, $s, t \in \mathbb{R}^n$, $y_0, z_0 \in \mathbb{R}$, such that $s, t \geq 0$, $y_0, z_0 \geq 0$, and the following conditions hold:

$$\begin{aligned} \alpha &= A^\top y + s - y_0 \pi \\ \alpha &= A^\top z + t + z_0 \pi \\ \beta &\leq b^\top y - y_0 \pi_0 \\ \beta &\leq b^\top z + z_0(\pi_0 + 1). \end{aligned} \tag{2.1}$$

Proof. Since $P^{(\pi,\pi_0)} \neq \emptyset$, we may assume without loss of generality that $\Pi_1^{(\pi,\pi_0)} \neq \emptyset$. Our proof considers $\Pi_1^{(\pi,\pi_0)}$ and $\Pi_2^{(\pi,\pi_0)}$ separately. The required result follows by simply observing that if $\alpha^\top x \geq \beta$ is valid for both $\Pi_1^{(\pi,\pi_0)}$ and $\Pi_2^{(\pi,\pi_0)}$, then it must be valid for $P^{(\pi,\pi_0)}$, too.

Let us first show that $\alpha^\top x \geq \beta$ is valid for $\Pi_1^{(\pi,\pi_0)}$ if and only if there exist $(y, y_0) \in \mathbb{R}^m \times \mathbb{R}$ and $s \in \mathbb{R}^n$ such that

$$\alpha = A^\top y + s - y_0 \pi, \quad \beta \leq b^\top y - y_0 \pi_0, \quad s \geq 0, \quad y_0 \geq 0. \tag{CRT-1}$$

Suppose $\alpha = A^\top y + s - y_0 \pi$ and $\beta \leq b^\top y - y_0 \pi_0$ for some $(y, y_0) \in \mathbb{R}^m \times \mathbb{R}$ and $s \in \mathbb{R}^n$ such that $s \geq 0$ and $y_0 \geq 0$. Then for all $x \in \Pi_1^{(\pi,\pi_0)}$,

$$\begin{aligned} \alpha^\top x \;&=\; y^\top A x + s^\top x - y_0 \pi^\top x \;=\; y^\top b + s^\top x - y_0 \pi^\top x \\ &\geq\; y^\top b - y_0 \pi^\top x \;\geq\; y^\top b - y_0 \pi_0 \;\geq\; \beta. \end{aligned}$$

Conversely, suppose $\alpha^\top x \geq \beta$ is valid for $\Pi_1^{(\pi,\pi_0)}$. Consider the linear programs

$$\begin{aligned} \min\ & \alpha^\top x \\ \text{s.t. } & Ax = b \\ & \pi^\top x + x_0 = \pi_0 \\ & x \geq 0, x_0 \geq 0 \end{aligned} \quad \text{(P)} \qquad \begin{aligned} \max\ & b^\top y + \pi_0 y_0 \\ \text{s.t. } & A^\top y + \pi y_0 \leq \alpha \\ & y_0 \leq 0 \end{aligned} \quad \text{(D)}.$$

The feasible region of (P) is $\left\{(x, x_0) \in \mathbb{R}^n \times \mathbb{R} : x \in \Pi_1^{(\pi,\pi_0)}\right\}$, and thus is nonempty. Furthermore, since $\alpha^\top x \geq \beta$ is valid for $\Pi_1^{(\pi,\pi_0)}$, (P) has a finite optimum $\beta^* \geq \beta > -\infty$. By Corollary 8, the feasible region of (D) is nonempty. Therefore, by strong duality, there exist $(y, y_0) \in \mathbb{R}^m \times \mathbb{R}$ such that $A^\top y + \pi y_0 \leq \alpha, y_0 \leq 0$, and $b^\top y + \pi_0 y_0 = \beta^* \geq \beta$, but upon flipping the sign of y_0 and adding slack variables $s \geq 0$, this is exactly the condition (CRT-1).

Let us now consider $\Pi_2^{(\pi,\pi_0)}$. The conditions that $\alpha^\top x \geq \beta$ is valid for $\Pi_2^{(\pi,\pi_0)}$ if and only if there exist $(z, z_0) \in \mathbb{R}^m \times \mathbb{R}$ and $t \in \mathbb{R}^n$ such that

$$\alpha = A^\top z + t + z_0\pi, \quad \beta \leq b^\top z + z_0(\pi_0 + 1), \quad t \geq 0, \quad z_0 \geq 0 \tag{CRT-2}$$

follows analogously if $\Pi_2^{(\pi,\pi_0)} \neq \emptyset$. So suppose $\Pi_2^{(\pi,\pi_0)} = \emptyset$. By Farkas' lemma applied to $\Pi_2^{(\pi,\pi_0)}$, there exist $(u, u_0) \in \mathbb{R}^m \times \mathbb{R}$ such that

$$A^\top u + \pi u_0 \leq 0, \quad b^\top u + (\pi_0 + 1)u_0 > 0, \quad u_0 \geq 0.$$

Note that we must have $u_0 > 0$ (otherwise, by Farkas' lemma, we would get a contradiction to $P \neq \emptyset$, and thus contradicting $P^{(\pi,\pi_0)} \neq \emptyset$). But then for any (y, y_0) and s that satisfy (CRT-1) and any appropriately scaled (u, u_0) such that

$$b^\top u + (\pi_0 + 1)u_0 \geq y_0 \quad \text{and} \quad u_0 \geq y_0,$$

we obtain (z, z_0) and t that satisfy (CRT-2) by setting

$$z := y + u, \quad z_0 := u_0 - y_0 \geq 0, \quad t := s - A^\top u - \pi u_0 \geq 0.$$

This completes the proof. □

Remark 11. As noted in [34], for technical reasons we usually assume that the trivial inequality $0^\top x \geq -1$ is contained (implicitly) in the system $Ax = b, x \geq 0$ that describes the constraint set. For problems with at least one bounded variable, say $l_j \leq x_j \leq u_j$, such trivial inequality may be obtained by adding $x_j \geq l_j$ and $-x_j \geq -u_j$ and dividing the resulting inequality by $u_j - l_j > 0$. Due to the presence of $0^\top x \geq -1$, we are allowed to require the $\leq$ inequalities for β in Theorem 10 to hold at equality. We will therefore make such an assumption in all of our subsequent discussions.

Theorem 10 immediately allows us to check whether a point $\hat{x} \in P$ lies in $P^{(\pi,\pi_0)}$.

Corollary 12. Let $\hat{x} \in P$. Then $\hat{x} \in P^{(\pi,\pi_0)}$ if and only if the following optimization problem, called *Cut Generating Linear Program* (CGLP), has a non-negative optimal objective value.

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

Therefore, given a point $\hat{x} \in P$ and a split disjunction $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}$, a most violated split cut $\alpha^\top x \geq \beta$ can be obtained by solving the corresponding (CGLP(π, π_0)). The optimal objective value is negative if such a cut exists. Otherwise, (CGLP(π, π_0)) proves $\hat{x} \in P^{(\pi, \pi_0)}$.

Remark 13 (Normalization)**.** The objective value of (CGLP(π, π_0)) is unbounded from below when it has a negative objective value. This is because one can scale up the solutions $(\alpha, \beta, y, z, s, t, y_0, z_0)$ while maintaining feasibility. In other words, the feasible region of (CGLP(π, π_0)) is a cone. Therefore, a normalization constraint $f(\alpha, \beta, y, z, s, t, y_0, z_0) = \kappa$, where κ is a positive constant normally set to be 1, is introduced to truncate the cone of feasible solutions. Many choices of f are possible, for example,

- β-normalization [11]
$$f(\cdot) := \beta \quad \text{or} \quad f(\cdot) := -\beta;$$
- α-normalization [11]
$$f(\cdot) := \sum_{j=1}^{n} |\alpha_j|;$$
- trivial normalization
$$f(\cdot) := y_0 + z_0;$$
- standard normalization [10]
$$f(\cdot) := \sum_{j=1}^{m} |y_j| + \sum_{j=1}^{m} |z_j| + \sum_{j=1}^{n} s_j + \sum_{j=1}^{n} t_j + y_0 + z_0;$$

- Euclidean normalization [34]

$$f(\cdot) := \sum_{j=1}^{m} \|A_j\||y_j| + \sum_{j=1}^{m} \|A_j\||z_j| + \sum_{j=1}^{n} s_j + \sum_{j=1}^{n} t_j + \|\pi\|y_0 + \|\pi\|z_0$$

  where A_j is the jth column of A and $\|\cdot\|$ is the Euclidean norm of a vector.

Normalization methods directly affect the quality of cut coefficients α and β as solutions of (CGLP(π, π_0)). The conditions of interest in our context are the trivial and the standard normalization conditions. The former is simple enough to make the separation of arbitrary split cuts possible, and the latter usually produces stronger cuts [34] and is used in our setting as a heuristic strengthening procedure for split cuts. We will return to this in Chapter 3.

We refer to [11, 12, 34] for in-depth discussions of different normalization conditions.

The following observation on the nonnegativity of Farkas' multipliers y and z in (CGLP(π, π_0)) is useful when imposing normalization conditions. It allows us to avoid unnecessary linearization of the absolute value constraint on y and z. While, in the literature, previous works use this observation as a fact, we couldn't find any proof. Here, we give a proof for the sake of completeness.

Proposition 14. Let $\hat{x} \in P$. If (CGLP(π, π_0)) has an optimal solution under the trivial, standard, or Euclidean normalization condition, then it has an optimal solution in which the Farkas' multipliers y and z are nonnegative.

Proof. Let $(\hat{\alpha}, \hat{\beta}, \hat{y}, \hat{z}, \hat{s}, \hat{t}, \hat{y}_0, \hat{z}_0)$ be an optimal solution to (CGLP(π, π_0)) under normalization $f(\cdot)$, where $f(\cdot)$ belongs to one of the conditions in the statement. Define $\hat{c} \in \mathbb{R}^m$ such that

$$\hat{c}_j := \begin{cases} 0 & \text{if } \hat{y}_j \geq 0 \text{ and } \hat{z}_j \geq 0, \\ -\hat{y}_j & \text{if } \hat{y}_j < 0 \text{ and } \hat{z}_j \geq 0, \\ -\hat{z}_j & \text{if } \hat{y}_j \geq 0 \text{ and } \hat{z}_j < 0, \\ -\hat{y}_j - \hat{z}_j & \text{if } \hat{y}_j < 0 \text{ and } \hat{z}_j < 0. \end{cases}$$

and consider

$$\begin{array}{llll} \overline{\alpha} := \hat{\alpha} + A^\top \hat{c} & \overline{\beta} := \hat{\beta} + b^\top \hat{c} & \overline{y} := \hat{y} + \hat{c} & \overline{z} := \hat{z} + \hat{c}, \\ \overline{s} := \hat{s}, & \overline{t} := \hat{t}, & \overline{y}_0 := \hat{y}_0, & \overline{z}_0 := \hat{z}_0. \end{array}$$

It is straightforward to see that

(i) $|\overline{y}_j| + |\overline{z}_j| = |\hat{y}_j| + |\hat{z}_j|$ for all $j = 1, 2, \ldots, m$;

(ii) $\overline{\alpha}^\top \hat{x} - \overline{\beta} = \hat{\alpha}^\top \hat{x} - \hat{\beta}$;

(iii) $\overline{y} \geq 0$ and $\overline{z} \geq 0$.

The proof is completed by noting that (i) gives feasibility of $(\overline{\alpha}, \overline{\beta}, \overline{y}, \overline{z}, \overline{s}, \overline{t}, \overline{y}_0, \overline{z}_0)$ under the same normalization $f(\cdot) = \kappa$; (ii) gives optimality; and (iii) gives nonnegativity as required. □

As a result of Proposition 14, we will require that $y \geq 0$ and $z \geq 0$ in (CGLP(π, π_0)) from now on.

2.2 Cut lifting

Lifting is a classical topic in integer programming, whose origin dates back to the 1960's and 70's in the contexts of the group problem [39] and the set packing problem [44]. A *lifting problem* can be stated as follows:

We are given

$$\begin{aligned}
\mathcal{I} &\subseteq \{1, 2, \ldots, n\}, \\
P &= \big\{x \in \mathbb{R}^n : Ax = b, x \geq 0\big\}, \\
Q &= \big\{x \in P : x_j \in \mathbb{Z}, \ \forall\, j \in \mathcal{I}\big\}, \\
P^R &= \left\{x^R \in \mathbb{R}^{n-q} : \begin{pmatrix} x^R \\ 0 \end{pmatrix} \in P\right\}, \\
Q^R &= \big\{x^R \in P^R : x^R_j \in \mathbb{Z}, \ \forall\, j \in \mathcal{I} \cap \{1, 2, \ldots, n-q\}\big\},
\end{aligned} \tag{2.2}$$

i.e., P^R is the restriction of P obtained by setting the last q variables to 0. Given an inequality $\big(\alpha^R\big)^\top x^R \geq \beta$ valid for Q^R, find $\alpha \in \mathbb{R}^n$ such that $\alpha^\top x \geq \beta$ is valid for Q and $\alpha_j = \alpha^R_j$ for $1 \leq j \leq n - q$.

When trying to obtain a split cut that separates $\hat{x} \in P$ from Q, a usual strategy is to work in the subspace, say $\mathbb{R}^{n-q}$, of variables $\hat{x}_j$ that are not at their bound (i.e., $\hat{x}_j > 0$), find a cut $\big(\alpha^R\big)^\top x \geq \beta$ in $\mathbb{R}^{n-q}$, and then lift it to $\mathbb{R}^n$. Note that since $x \geq 0$ for every $x \in P$, we would like the lifted coefficients as small as possible, in the sense that the lifted cut is potentially stronger. Therefore, an approach is to minimize cut coefficients subject to the lifted cut remaining valid. We give the details below.

A trivial lifting procedure for split cuts:

Suppose P, Q, P^R, Q^R are as in (2.2). Given a point $\hat{x} \in P$ such that, up to permutation of indices, $\hat{x}_j > 0$ for $1 \leq j \leq n-q$ and $\hat{x}_j = 0$ otherwise. Then the projection $\hat{x}^R$ of $\hat{x}$ onto $\mathbb{R}^{n-q}$ lies in P^R. Given $(\alpha^R, \beta) \in \mathbb{R}^{n-q} \times \mathbb{R}$ such that

(i) $(\alpha^R)^\top \hat{x}^R < \beta$;

(ii) $(\alpha^R)^\top x^R \geq \beta$ is a split cut for P^R corresponding to a split disjunction $(\pi^R, \pi_0) \in \mathbb{Z}^{n-q} \times \mathbb{Z}$.

Our goal is to obtain $\alpha \in \mathbb{R}^n$ such that $\alpha_j = \alpha_j^R$ for $1 \leq j \leq n-q$ and

(i') $\alpha^\top \hat{x} < \beta$;

(ii') $\alpha^\top x \geq \beta$ is a split cut for P corresponding to a split disjunction $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}$ where $\pi_j = \pi_j^R$ for $1 \leq j \leq n-q$ and $\pi_j = 0$ for $n-q+1 \leq j \leq n$;

(iii') α_j is minimized for $n-q+1 \leq j \leq n$.

Note that, in the above, (ii) implies $(\alpha^R)^\top x^R \geq \beta$ is valid for Q^R and (ii') implies $\alpha^\top x \geq \beta$ is valid for Q. The following Theorem 15 is a straightforward extension of the trivial lifting procedure of lift-and-project cuts described in [11].

Theorem 15. Given (α^R, β) that satisfies (i) and (ii). Let $A^R \in \mathbb{R}^{m \times (n-q)}$ be obtained from A by removing the last q columns. Let $y^R \in \mathbb{R}^m, z^R \in \mathbb{R}^m, s^R \in \mathbb{R}^{n-q}, t^R \in \mathbb{R}^{n-q}, y_0^R \in \mathbb{R}, z_0^R \in \mathbb{R}$ satisfy

$$\begin{aligned} \alpha^R &= (A^R)^\top y^R + s^R - y_0^R \pi^R = (A^R)^\top z^R + t^R + z_0^R \pi^R, \\ \beta &= b^\top y^R - y_0^R \pi_0 = b^\top z^R + z_0^R (\pi_0 + 1), \\ & y^R, z^R, s^R, t^R, y_0^R, z_0^R \geq 0. \end{aligned} \tag{2.3}$$

Define $\alpha \in \mathbb{R}^n$ such that $\alpha_j = \alpha_j^R$ for $1 \leq j \leq n-q$ and

$$\alpha_j = \max\left\{A_j^\top y^R, A_j^\top z^R\right\}, \quad n-q+1 \leq j \leq n, \tag{2.4}$$

where A_j is the jth column of A. Then (α, β) satisfies (i') and (ii'). Furthermore, if the set of multipliers (y^R, z^R) that satisfy (2.3) are unique, then (α, β) satisfies (iii').

Proof. Note that (i') is always satisfied since $\hat{x}_j = 0$ for $n - q + 1 \leq j \leq n$. To see that (ii') is satisfied, observe that α satisfies

$$\begin{aligned} \alpha_j &= \alpha_j^R, && 1 \leq j \leq n - q, \\ \alpha_j &\geq A_j^\top y^R, && n - q + 1 \leq j \leq n, \\ \alpha_j &\geq A_j^\top z^R, && n - q + 1 \leq j \leq n, \end{aligned} \tag{2.5}$$

and (y^R, z^R) satisfies (2.3). The validity of $\alpha^\top x \geq \beta$ corresponding to the split disjunction (π, π_0) defined in (ii') follows directly by applying Theorem 10.

If the multipliers (y^R, z^R) that satisfy (2.3) are unique, then the lifted cut $\alpha^\top x \geq \beta$ is valid if and only if (2.5) holds, and thus α_j is minimized by taking the maximum of $A_j^\top y^R$ and $A_j^\top z^R$. □

Remark 16. It is clear from our construction in this section that, given $\hat{x} \in P$, there is a split cut $\alpha^\top x \geq \beta$ for P that corresponds to disjunction (π, π_0) and cuts off $\hat{x}$ if and only if there is a split cut $(\alpha^R)^\top x^R \geq \beta$ for P^R that corresponds to disjunction (π^R, π_0) and cuts off $\hat{x}^R$.

Remark 17. There is no loss of generality in assuming that the feasible region of every MIP is of the form Q. Indeed, for a free variable x_j we can replace it with two bounded variables x_j^+ and x_j^- such that $x_j = x_j^+ - x_j^-$. For a bounded variable x_j with nonzero lower bound l_j and upper bound u_j, we can do either of the following:

- shift the lower bound, i.e., replace b with $b - l_j A_j$, then subsume the upper bound into constraints $Ax = b$; or
- complement the variable and shift the upper bound, i.e., replace A_j with $-A_j$ and replace b with $b - u_j A_j$, then subsume the lower bound into constraints $Ax = b$.

Of course, we can always require inequality constraints to hold at equality by adding slack variables.

Following Remark 17, the lifting procedure described here easily extends to MIPs whose variables have nonzero lower and/or upper bounds. In practice, depending on a given $\hat{x}$, for each bounded variable $\hat{x}_j$ we choose one of the transformations in Remark 17 so that, after the transformation, either $\hat{x}_j$ is not at the bound or $\hat{x}_j = 0$.

2.3 Optimizing over the split closure

The *split closure* $\mathcal{SC}$ for mixed-integer set $P \cap (\mathbb{Z}^p \cap \mathbb{R}^{n-p})$ is defined as

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

Given $\hat{x} \in P$, the problem of deciding wether $\hat{x} \in \mathcal{SC}$ is $\mathcal{NP}$-hard in general [23]. Balas and Saxena [13] implemented an iterative procedure that alternates between a *Master Problem* and a *Separation Problem* to find

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

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 split cut violated by $\hat{x}$, or proves that $\hat{x} \in \mathcal{SC}$. The Separation Problem is a mixed-integer nonlinear program obtained from (CGLP(π,π_0)) with trivial normalization $y_0 + z_0 = 1$, and allowing (π, π_0) to vary over $\mathbb{Z}^n \times \mathbb{Z}$, written 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 [13], (SP) is shown to be equivalent to a parametric MILP with a scalar parameter taking values between 0 and $\frac{1}{2}$. For completeness we state and prove their result here.

Theorem 18 ([13]). The optimum of (SP) is equal to the optimum of the following parametric mixed-integer linear program,

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

where each MILP(θ) is given by

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

Proof [13]. Apply the following modifications to (SP):

(a) substitute $\alpha = A^\top y + s - y_0 \pi$ and $\beta = b^\top y - y_0 \pi_0$ into the objective;

(b) eliminate α and β from the constraints;

(c) substitute $z_0 = 1 - y_0$ into the constraints;

(d) replace $y - z$ with w and replace y_0 with θ.

We see that (SP) is equivalent to

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

Now, fix $\theta = \hat{\theta}$ and suppose $(\hat{w}, \hat{s}, \hat{t}, \hat{\pi}, \hat{\pi}_0)$ is a feasible solution for MILP($\hat{\theta}$). Then

$$\overline{w} := -\hat{w}, \quad \overline{s} := \hat{t}, \quad \overline{t} := \hat{s}, \quad \overline{\pi} := -\hat{\pi}, \quad \overline{\pi}_0 := -\hat{\pi}_0 - 1$$

is a feasible solution for MILP($\overline{\theta}$) with $\overline{\theta} := 1 - \hat{\theta}$, and that

$$\begin{aligned} \overline{s}^\top \hat{x} - \overline{\theta}(\overline{\pi}^\top \hat{x} - \overline{\pi}_0) &= \hat{t}^\top \hat{x} - (1 - \hat{\theta})(-\hat{\pi}^\top \hat{x} + \hat{\pi}_0 + 1) \\ &= \hat{t}^\top \hat{x} + \hat{\pi}^\top \hat{x} - \hat{\pi}_0 - 1 + \hat{\theta} - \hat{\theta}(\hat{\pi}^\top \hat{x} - \hat{\pi}_0) \\ &= \hat{s}^\top \hat{x} - \hat{\theta}(\hat{\pi}^\top \hat{x} - \hat{\pi}_0), \end{aligned}$$

where the last line follows from $\hat{s}^\top \hat{x} = \hat{t}^\top \hat{x} + \hat{\pi}^\top \hat{x} - \hat{\pi}_0 - 1 + \hat{\theta}$, which in turn is obtained as a linear combination of constraints in MILP($\hat{\theta}$), i.e.,

$$\hat{x}^\top \left(A^\top \hat{w} + \hat{s} - \hat{t} - \hat{\pi}\right) - \left(b^\top \hat{w} - \hat{\pi}_0\right) = -1 + \hat{\theta}.$$

Therefore,

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

□

We immediately obtain a result similar to that of Corollary 12.

Corollary 19. Let $\hat{x} \in P$. Then $\hat{x} \in \mathcal{SC}$ if and only if

$$\min_{0 \le \theta \le \frac{1}{2}} \mathrm{MILP}(\theta) \ge 0.$$

Writing (SP) as a parametric mixed-integer program enables us to approximate its optimum by solving a finite sequence of problems MILP(θ) with varying values for θ using state-of-the-art MIP solvers.

2.4 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 (1.1) 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 $k \ge 1$. 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. [8] 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. [15] 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 2.1 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.

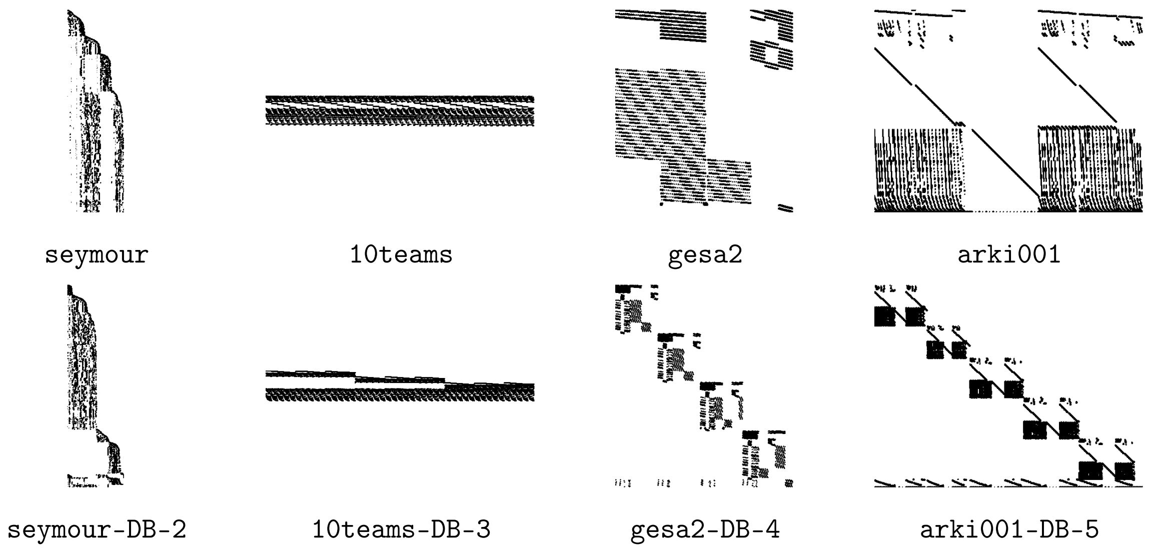


Figure 2.1: Original problem structure versus its DB-k forms

Chapter 3

Implementation Details

In this chapter we outline the computational details in our implementation of a split cut separator with (and without) structural information supplied by a given DB-k form. We follow the idea of Balas and Saxena [13] to approximate the optimal value of (SP) by solving a sequence of (MILP(θ))'s parametrized by $\theta \in [0, 1/2]$. The structure of this chapter is as follows. We begin by introducing some practical methods used in our implementation that are taken from or similar to those already presented in [13]. Then we discuss new features that we added to serve our goal of evaluating split cuts based on sparsity properties. Finally we lay out concise algorithmic descriptions of the entire computational process.

3.1 Existing methods for practical computation

Discretizing parameters. We denote by Θ the set of values of θ for which (MILP(θ)) will be solved. By Theorem 18, obtaining an exact solution of (SP) would require that $\Theta = [0, 1/2]$, which is prohibitive in practice. Therefore, we take Θ as a uniform parameter grid of finitely many points between 0 and 1/2. The initial size of Θ is t, and we increase the number of grid points whenever necessary, following the criteria in Algorithm 1.

Stabilizing objective. When some entries in an incumbent solution $\hat{x}$ are close to their bounds, say $\hat{x}_j \approx 0$ for $j \in \mathcal{J}$, the objective coefficients for s in (MILP(θ)) are close to zero. As a result, there may exist multiple feasible solutions $(w, s, t\pi, \pi_0)$ with differing values s_j and t_j for $j \in \mathcal{J}$, whose objective values are arbitrarily close to optimum. Some of these s_j's and t_j's can be unnecessarily large, which in turn leads to weak split directions and weak cuts. (This is because large s_j's or t_j's usually produce large π_j's, and when the disjunction coefficients are unnecessarily large, two undesirable numerical issues can happen. First, a split disjunction may be "squeezed" as its coefficients simultaneously become large, the two sides of the disjunction get close to each other in Euclidean norm, and thus we are not cutting off many points. Second, a split disjunction may be "tilted" away from the disjunction that produces most violated cut, and thus leading to weaker cuts. We illustrate this latter observation in Example 20.) However, due to numerical tolerances in a MIP solver, these poorly-scaled suboptimal solutions may be regarded as optimal. Therefore, to avoid obtaining unnecessarily weak cut coefficients, 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. We show the effect of this modification in a concrete example.

Example 20. Consider the integer program from Example 2. Let us write it in the general form of (1.1) by adding 2 continuous slacks variables x_3 and x_4. Given a point $x' \in \mathbb{R}^4$, we may find a split disjunction $(\pi, \pi_0) \in \mathbb{Z}^4 \times \mathbb{Z}$ and a split cut that separates x' from $P^{(\pi,\pi_0)}$ by substituting the data into (MILP(θ)),

$$\begin{aligned} \min \quad & \sum_{j=1}^{4} x'_j s_j - \theta \sum_{j=1}^{4} x'_j \pi_j + \theta \pi_0 \\ \text{s.t.} \quad & \pi = \begin{bmatrix} -2 & 2 \\ 6 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} w + s - t \\ & \pi_0 = \begin{pmatrix} 9 & 5 \end{pmatrix} w - 1 + \theta \\ & w \text{ free}, \ s, t \geq 0 \\ & (\pi, \pi_0) \in \mathbb{Z}^4 \times \mathbb{Z}, \ \pi_3 = \pi_4 = 0 \end{aligned} \tag{3.1}$$

and solve it for some $\theta \in [0, 1/2]$. Suppose we want to separate

$$x' = (0.000000001,\ 1.5,\ 0.000000002,\ 3.499999998)^\top.$$

Take $\theta = 1/10$. It is easy to verify that

$$w' = \begin{pmatrix} \frac{210}{31} \\ \frac{4}{35} \end{pmatrix}, s' = \begin{pmatrix} \frac{1}{15} \\ 0 \\ 0 \\ 0 \end{pmatrix}, t' = \begin{pmatrix} 0 \\ 0 \\ \frac{210}{31} \\ \frac{4}{35} \end{pmatrix}, \pi' = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \pi'_0 = 1 \tag{3.2}$$

is an optimal solution for (3.1) with optimal objective value $-\frac{749999999}{15000000000} \approx -0.049999$. We thus recover (α, β) from (s', t', π', π'_0) as

$$\alpha = s' - \theta\pi' = \begin{pmatrix} \frac{1}{15} \\ -\frac{1}{10} \\ 0 \\ 0 \end{pmatrix}, \quad \beta = -\theta\pi_0 = -\frac{1}{10}.$$

Since $x_3 = 9 + 2x_1 - 6x_2$ and $x_4 = 5 - 2x_1 - x_2$, we can eliminate cut coefficients for slack variables (in this case they are already 0) and obtain a cut in the space of (x_1, x_2) as

$$\frac{1}{15}x_1 - \frac{1}{10}x_2 \geq -\frac{1}{10}.$$

However, solving (3.1) with the same x' and $\theta = 1/10$ by CPLEX 12.71 we obtained a solution

$$\begin{aligned} &w'' \approx \begin{pmatrix} -\frac{23333279}{210} \\ \frac{4}{35} \end{pmatrix}, \; s'' \approx \begin{pmatrix} 0 \\ 0 \\ \frac{23333279}{210} \\ 0 \end{pmatrix}, \; t'' \approx \begin{pmatrix} \frac{18333329}{15} \\ 0 \\ 0 \\ \frac{4}{35} \end{pmatrix} \\ &\pi'' = \begin{pmatrix} -1000000 \\ -666665 \\ 0 \\ 0 \end{pmatrix}, \; \pi''_0 = -999998 \end{aligned} \tag{3.3}$$

whose objective value is approximately -0.049678. After eliminating slack variables, (3.3) gives the cut

$$322221.70476x_1 - 599998.61428x_2 \geq -899997.87142.$$

We note that lower bounding the objective coefficients by some appropriate δ, for example $\delta = 0.0001$, helped CPLEX find the optimal solution (3.2). Figure 3.1 shows the cuts obtained from CPLEX by solving (3.1) with and without our stabilizing objective.

Cut strengthening. Once a feasible solution $(\bar{w}, \bar{s}, \bar{t}, \bar{\pi}, \bar{\pi}_0)$ for MILP$(\bar{\theta})$ with a negative objective value is found, we feed $(\bar{\pi}, \bar{\pi}_0)$ into (CGLP(π, π_0)) with the standard normalization

$$\sum_{j=1}^{m} y_j + \sum_{j=1}^{m} z_j + \sum_{j=1}^{n} s_j + \sum_{j=1}^{n} t_j + y_0 + z_0 = \kappa$$

for a fixed positive constant κ. This normalization is shown in [34] to produce empirically stronger cuts than the normalization $y_0 + z_0 = 1$ used in deriving (MILP(θ)).

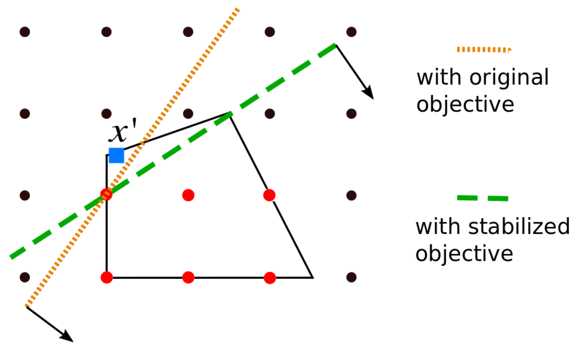


Figure 3.1: Effect of stabilizing objective

Cut lifting. We work in the subspace of the variables that are not at their bounds in the incumbent solution, and lift the resulting cuts to the full space following the lifting procedure described in Chapter 2.

Set covering. In an effort to impose some degree of 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 (MILP(θ)) in the next θ point. This modification ensures that (MILP(θ)) never returns a split disjunction whose support is identical to that of an already discovered one. In practice it also prevents us from obtaining the same π with varying θ values, which is an important property that makes our implementation more effective. Moreover, we observed that the computation time spent on solving set covering problems is almost negligible compared to the time spent on solving (MILP(θ)).

3.2 New features in our implementation

In addition to some modifications of (MILP(θ)) described in the last section, we added three more constraints to (MILP(θ)) in order to control the quality and structural properties of split disjunctions discovered by (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

$$\sigma \leq \pi^\top \hat{x} - \pi_0 \leq 1 - \sigma \tag{C1}$$

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

$$-Ur_j \leq \pi_j \leq Ur_j, \quad \forall\, j = 1, \dots, p, \quad \text{and} \quad \sum_{j=1}^{p} r_j \leq M \tag{C2}$$

where U is an artificial upper bound on the magnitude of the components of π. Note that imposing an artificial upper and lower bound on disjunction coefficients not only helps us control the sparsity, but also prevent us from obtaining weak splits due to unnecessarily large disjunction coefficients.

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

$$\begin{aligned} \pi_j = s_j = t_j = 0, &\quad \forall\, j \notin \mathcal{C}^i \\ w_j = 0, &\quad \forall\, j \notin \mathcal{R}^i \end{aligned} \tag{C3}$$

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

Certifying validity. For every split cut $\alpha^\top x \geq \beta$ generated from (CGLP(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, then the cut is invalid and we discard it. In theory, split cuts returned by (CGLP(π, π_0)) should always be valid as they are obtained from Farkas multipliers which already certify their validity; in practice, however, we may obtain invalid cuts due to either a numerical issue within the LP or MIP solver or an error in our own implementation. Having an independent procedure that checks the validity of cuts before adding them to the Master Problem ensures our computational results are reliable, in the sense that as long as our checker implementation is correct, not a single invalid cut is added during the entire computation.

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}.

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 (MP) for a while (see Algorithm 1), 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.

3.3 Algorithmic descriptions

Details of the iterative procedure are described in Algorithm 1. The cut generation procedure to separate a given $\hat{x} \in P$ is summarized in Algorithm 2.

Algorithm 1: 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
     Iter $= 0$. Denote $k$ the number of blocks in a given DB-$k$ from; if no decomposition
     is available, set $k = 1$.
2  TimeLimit $\leftarrow \tau_1$, Cutoff $\leftarrow \gamma_1$.
3  Iter $\leftarrow$ 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 $0 \le n \le 39$ then $s \leftarrow 2^{\lfloor 0.1n \rfloor} t$. else $s \leftarrow 16t$.
6  Set parameter grid $\Theta$ uniformly with $|\Theta| \leftarrow s$.
7  Separation.
     for $j = 1, \dots, k$ do Generate a set $\mathcal{K}^{(j)}$ of cuts following
     CutGen($\hat{x}, \Theta$, Cutoff, 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 TimeLimit$= \tau_1$, Cutoff$= \gamma_1$ then TimeLimit $\leftarrow \tau_2$, go to 7.
10 else if TimeLimit$= \tau_2$, Cutoff$= \gamma_1$ then Cutoff $\leftarrow \gamma_2$, go to 7.
11 else if TimeLimit$= \tau_2$, Cutoff$= \gamma_2$ then return $\hat{x}$.
```

Algorithm 2: CutGen($\hat{x}, \Theta, \gamma, \tau$)

```
Input: Incumbent solution $\hat{x}$, parameter grid $\Theta$, upper cutoff limit $\gamma < 0$, time limit
       $\tau$, minimum cut violation $\epsilon > 0$, required properties of split disjunctions
       (C1),(C2),(C3). Polyhedron $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ describing the
       constraint set of (1.2).
Output: A set $\mathcal{K}$ of split cuts violated by $\hat{x}$.
1  $\mathcal{K} \leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$.
2  for $\theta \in \Theta$ do
3      Solve StCvIP($\hat{x}, \mathcal{S}$) and impose partial orthogonality if needed. Add
         fractionality (C1), sparsity (C2), and structure (C3) constraints to (MILP($\theta$))
         as indicated.
4      Solve (MILP($\theta$)) with time limit $\tau$.
5      if Found a feasible solution $(\pi, \pi_0)$ to (MILP($\theta$)) with objective value $\leq \gamma$. then
6          Perform cut strengthening to get cut $\alpha^\top x \geq \beta$.
7          Perform cut lifting on cut $\alpha^\top x \geq \beta$.
8          Perform cut cleaning on cut $\alpha^\top x \geq \beta$.
9          if $\alpha^\top \hat{x} - \beta \leq -\epsilon$ then
10             $\beta_l \leftarrow \min_{x \in P}\{\alpha^\top x : \pi^\top x \leq \pi_0\}$, $\beta_u \leftarrow \min_{x \in P}\{\alpha^\top x : \pi^\top x \geq \pi_0 + 1\}$.
                $\beta^* \leftarrow \min\{\beta_l, \beta_u\}$.
11             if $\beta \leq \beta^*$ then
12                 $\mathcal{K} \leftarrow \mathcal{K} \cup \{\alpha^\top x \geq \beta\}$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{\pi\}$.
13 return $\mathcal{K}$.
```

Chapter 4

Computational Experiments

In this chapter we present and discuss our computational results. In order to evaluate the strength of a certain (sub-)family of cuts, we use the amount of integrality gap closed by these cuts (c.f. Definition 3) as a measure of how strong that (sub-)family of cuts is.

While it is tempting to argue that more integrality gap closed by some cuts does not necessarily mean that these cuts are indeed more effective in practice, the amount of gap closed does provide a more stable measure of the cut strength than other possible candidates, e.g., the amount of time to find an optimal solution. Intuitively the amount of gap closed gives us a rough idea on how closer we are to the optimal solution. On the other hand, we note that the gap closed depends on the objective function in the MIP formulation, that is, it may happen that with one objective function a family of cuts closes 100% integrality gap but with a different objective function the same family closes 0% gap. This is not a concern for us because all MIP formulations have a clearly defined objective function and in practice we need only care about how close we can get to the optimal solution in the direction of the objective.

Previous results on the integrality gap closed by the split closure are only known on MIPLIB 3.0 instances, therefore, for comparison purposes we first carry out the computations on MIPLIB 3.0 instances. These instances are older and tend to be smaller than instances in the more recent MIPLIB 2003 and MIPLIB 2010, but nonetheless they represent a diverse selection of test instances (there are still unsolved problems in MIPLIB 3.0). Then in order to see whether we can obtain similar computational results on newer and possibly larger instances, we run the same experiments again on MIPLIB 2003 instances. We did not run our experiments on MIPLIB 2010 instances due to time constraint, but we expect that the conclusions we are able to draw from these computational tests would stay the same.

This chapter is organized as follows. Section 4.1 deals with the practical setup for the experiments: we explain how and why each parameter is chosen in our actual computation. Section 4.2 consists of three sets of experiments. We first show that our implementation is reasonable in the sense that our computational results are consistent with known results under the same setting. Then we evaluate the strength of split cuts under sparsity and structured sparsity constraints and demonstrate the potential advantages of exploiting sparsity. Finally in Section 4.3 we present results on MIPLIB 2003 instances.

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 4.1. 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	1 or 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	10^{-6} or 0.025

Table 4.1: Model parameter values used in computation

When only one nonzero element is allowed for split disjunctions, the corresponding split cuts are essentially lift-and-project cuts[1]. Therefore, in a way to test the reliability of our implementation, we first set $M = 1$ and compare the computational results with those of the lift-and-project closure by Bonami et al [21].

In their experimental analysis, Balas and Saxena [13] 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 actually set $U = 100$ (i.e., $-100 \leq \pi_j \leq 100$, for all $1 \leq j \leq p$). This allows for splits with somewhat larger coefficients, but we still need U to be finite for practical reasons.

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)'s are processed; if a decomposition is given, then we set $t = 20$ for each of the k blocks, and therefore $20k$ MILP(θ)'s 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}. In general, more cuts may be obtained by using such a loose bound, and we thus set $\sigma = 10^{-6}$ when trying to reproduce lift-and-project results of Bonami [21]. In all the other experiments, however, we impose a rather strict bound $\sigma = 0.025$. 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)'s from yielding unviolated split disjunctions due to numerical errors.

[1]Original ideas of the lift-and-project approach dates back to the early 1970's in the context of disjunctive programming, see Balas [9]. More explicit reformulation techniques for integer programs with all binary variables are proposed by Sherali and Adams [46], and by Lovász and Schrijver [43]. When lift-and-project is applied to a single variable x_j, Balas et al. [11] showed that it is equivalent to considering $P^{(e_j,0)}$ (c.f. Section 1.3), i.e., a disjunctive programming problem. Therefore, for general MIPs, we say that any inequality valid for $P^{(e_j,\pi_0)}$ for some integer variable x_j and $\pi_0 \in \mathbb{Z}$ is a lift-and-project cut.

4.2 Three experiments

4.2.1 How does our implementation compare with known results?

To check whether our implementation was reasonable, we performed two tests.

First, we tested the code on both MIPLIB 3.0 [19] and MIPLIB 2003 [5] instances, in a configuration where it approximates[2] a lift-and-project cut separator. We restricted the sparsity parameter $M = 1$ in order to allow only one nonzero element in split disjunctions, thus forcing our split cut separator to return lift-and-project cuts only. Since $M = 1$, we upper bounded, without loss of generality, the magnitude of disjunction coefficients by $U = 1$. The entire computation time for each instance is limited to one week, including the time taken to check cut validity. At termination, we measure the final percentage of integrality gap closed (GAPnew), which is then compared with the bounds (GAPknown) given in [21]. For each instance, we look at the percentage of gap closed divided by the analogous results in [21], i.e., we compute

$$\text{relative gap closed} := \frac{\text{GAPnew}}{\text{GAPknown}}.$$

Therefore, a 50% relative gap closed means that our result is equal to 50% of the known gap, and a 100% relative gap closed means that our result is exactly the same as the known gap. Note that, however, due to numerical tolerances, a 99% relative gap closed would usually mean the results are identical.

We do this comparison on the 57 MIPLIB 3.0 instances where the gaps in [21] are strictly positive. Table 4.2 shows the number of instances that fall within various categories based on this ratio. In particular, on 49 (out of 57) instances we closed at least 99% relative gap, and on 54 (out of 57) instances we closed at least 90% relative gap. The distribution of relative gaps for MIPLIB 2003 instances is similar. Appendix A.1 and A.2 contain the details of gap closed for each of MIPLIB 3.0 and MIPLIB 2003 instances, respectively.

Note that our implementation, which involves solving a large sequence of MILP(θ)'s, is not designed for lift-and-project cut, as the separation problem for lift-and-project cuts can be done by solving linear programs only [21]. Therefore, it is reasonable to expect that on some instances we could obtain a low percentage on the relative gap closed. It is somewhat striking to observe how close our results are to the results given in [21], considering that our approach is completely different, yet on many instances the gaps are identical within a $\pm 0.01\%$ tolerance.

[2] Here "approximates" is in a sense that our cut separator involves discretization of $\theta \in [0, 1/2]$.

relative gap closed	# instances
$\geq$ 99%	49
$\geq$ 90%	54
$\geq$ 50%	57

Table 4.2: Gap closed as a percentage of the known gap for the lift-and-project closure (from [21])

relative gap closed	# instances
$>$ 100%	8
$\geq$ 99%	25
$\geq$ 90%	30
$\geq$ 50%	31
$<$ 1%	23

Table 4.3: Gap closed as a percentage of the best known gap for the split closure (from [13] and [29])

Second, we tested the code on the MIPLIB 3.0 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. As in our first test, we limited the entire computation time for each instance to one week, including the time taken to check cut validity[3]. At termination, we measure the final percentage of integrality gap closed, which is then compared with the best of the bounds given in [13] and [29]. For each instance, we look at the percentage of gap closure divided by the best of the analogous results in [13] and [29]. We do this comparison on the 57 instances where the best known gaps are strictly positive. Table 4.3 shows the number of instances that fall within various categories based on this ratio. On 8 (out of 57) instances we closed more gap than the best available result, and on 30 (out of 57) instances we closed at least 90% relative gap.

On the other hand, we closed less than 1% relative gap on 23 (out of 57) instances. As shown in Figure 4.1 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

[3]Note that neither [13] nor [29] has any cut validity checking procedure and also that [13] mention no time limit on their experiments

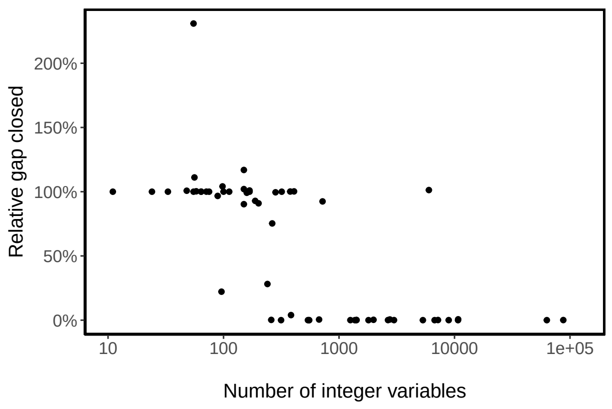


Figure 4.1: Gap closed as a percentage of the best known gap closure (from [13] and [29]), vs. number of integer variables

the experiments considering 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 23 instances.

The purpose of this experiment was to determine if our implementation was reliable, compared to other ones. Our highly consistent lift-and-project results ($M = 1$) demonstrate that the implementation was indeed reasonable. Although the results on general split cuts were not as convincing, it is sensible to expect that they would match with the best known results had we chosen a longer computation time limit and fine-tuned the parameters (this is omitted because the fine-tuning of the parameters is not the primary goal of this work). Finally, we note that the reliability of our implementation is further evidenced by the experiments in subsequent sections.

4.2.2 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 one week was set for all computations. Table 4.4 shows the details of our results. The first column of the table shows the best of the gaps given in [13] and [29], 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 `fast0507`, of the 168 hours spent, only 38% contributed to the actual computation. The remaining 62% 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 4.4, 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, an interesting related effect to observe is the sparsity of the cuts produced. Although sparse disjunctions do not necessarily lead to sparse cuts, Figure 4.2 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 4.4. 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 4.2. The horizontal lines in the figure 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 criterion. 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 0.420 with arbitrary disjunctions, it dropped to 0.140 with sparse ones, and 0.124 with sparse ± 1 disjunctions.

The resulting average integrality gap closed by the split cuts in the most restricted experiment (disjunctions with at most 10 nonzero ± 1 coefficients) is 69.0%, accounting for 92% of the 75.2% average for the best in [13] and [29]. Figure 4.3 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	Instance	$M = +\infty,\ U = 100$				$M = 10,\ U = 100$				$M = 10,\ U = 1$			
gap		Gap	# cuts	Time	% time	Gap	# cuts	Time	% time	Gap	# cuts	Time	% time
[13], [29]		closed	binding	(h)	checking	closed	binding	(h)	checking	closed	binding	(h)	checking
100.00	10teams	0.00	6700	28.73	27.03	84.91	4643	168.00	20.09	93.76	1245	168.00	14.50
100.00	air03	0.66	531	168.00	97.36	100.00	749	0.52	89.70	100.00	324	0.05	86.47
91.23	air04	0.02	387	168.00	98.33	49.12	350	168.00	74.98	89.57	525	168.00	67.36
61.98	air05	0.07	587	168.00	93.18	54.67	381	168.00	47.25	63.53	370	168.00	16.50
83.95	arki001	0.01	4212	168.00	94.78	77.11	214	168.00	0.05	42.61	236	32.54	0.24
99.60	bell3a	99.64	87	22.44	0.03	75.50	100	1.72	0.03	75.73	81	0.08	0.20
92.95	bell5	93.24	185	15.23	0.01	91.18	75	28.07	0.00	92.57	222	0.29	0.03
46.52	blend2	35.03	453	3.16	1.99	39.71	76	1.48	0.04	45.56	72	0.86	0.12
65.17	cap6000	66.00	1068	168.00	0.19	63.83	61	0.40	1.77	64.69	119	2.12	0.65
0.22	dano3mip	0.00	1217	168.00	88.22	0.16	428	168.00	17.00	0.24	492	168.00	36.16
8.20	danoint	9.11	515	168.00	2.09	8.15	294	168.00	0.76	8.98	393	168.00	0.74
100.00	dcmulti	100.00	1069	1.21	6.62	99.80	252	33.00	0.02	99.84	309	11.71	0.01
100.00	egout	100.00	281	0.17	0.43	100.00	223	0.18	0.12	98.64	230	0.00	2.22
19.08	fast0507	0.00	5034	168.00	55.62	0.55	360	168.00	55.87	7.50	401	168.00	61.66
99.68	fiber	0.03	445	168.00	89.91	37.38	232	168.00	0.01	72.99	263	168.00	0.02
99.75	fixnet6	99.87	562	157.75	0.01	99.83	558	168.00	0.01	99.86	342	168.00	0.01
100.00	flugpl	100.00	86	0.05	0.05	100.00	86	0.05	0.06	98.49	124	0.00	0.59
100.00	gen	90.28	510	12.91	11.97	100.20	281	136.25	0.04	100.20	378	0.08	6.95
99.70	gesa2	99.93	255	168.00	0.15	99.87	168	168.00	0.01	100.00	277	36.31	0.01
99.97	gesa2_o	92.41	336	168.00	0.38	87.41	303	168.00	0.02	99.99	274	37.79	0.01
95.81	gesa3	3.77	378	6.50	8.77	95.08	240	168.00	0.04	96.01	292	54.24	0.01
95.20	gesa3_o	0.45	482	2.47	13.39	95.38	238	168.00	0.03	96.07	242	122.70	0.01
98.38	gt2	91.35	2528	168.00	0.57	94.18	105	168.00	0.00	98.89	175	75.47	0.01
58.48	harp2	0.02	142	0.13	15.46	12.93	90	1.95	0.07	45.21	161	70.59	0.02
100.00	khb05250	100.00	308	0.21	0.66	100.00	386	0.24	0.47	100.00	451	0.01	12.50
95.20	l152lav	0.26	2434	168.00	59.31	32.33	202	168.00	1.83	48.93	199	168.00	0.33
93.75	lseu	90.66	116	168.00	0.00	70.18	52	74.19	0.00	76.16	103	65.68	0.00
14.02	mas74	16.39	117	168.00	0.23	10.65	39	149.46	0.00	11.81	45	56.51	0.00
26.52	mas76	27.06	108	168.00	0.04	13.67	38	82.92	0.00	14.94	63	84.85	0.00
51.70	misc03	51.26	224	168.00	1.09	51.17	188	57.57	0.02	51.49	216	57.13	0.01
100.00	misc06	100.00	279	0.03	8.07	100.00	325	0.09	4.48	100.00	168	0.02	11.82
20.11	misc07	0.04	2020	168.00	25.33	16.92	226	168.00	0.07	16.62	322	168.00	0.01
100.00	mitre	0.00	6421	168.00	2.49	0.06	4466	168.00	0.19	23.52	2195	168.00	0.28
36.16	mkc	0.00	6530	128.30	7.86	55.96	534	168.00	1.13	65.11	942	168.00	1.27
99.98	mod008	99.97	315	168.00	0.51	54.71	74	39.72	0.00	55.51	128	52.65	0.00
100.00	mod010	0.06	5807	50.50	40.61	84.43	206	168.00	3.37	100.00	384	2.78	9.36
72.44	mod011	16.06	6403	43.39	9.48	83.78	1155	168.00	0.73	85.35	1115	168.00	0.78
92.18	modglob	95.97	191	7.45	0.03	96.52	181	11.07	0.03	95.49	164	132.90	0.00
100.00	nw04	0.07	491	168.00	96.81	57.30	325	168.00	98.41	85.08	380	168.00	2.59
87.42	p0033	87.42	123	9.05	0.00	81.59	55	29.99	0.00	82.43	66	31.20	0.00
74.93	p0201	68.15	1570	168.00	4.54	69.94	518	168.00	0.02	71.31	162	168.00	0.01
99.99	p0282	99.52	130	96.74	0.03	98.97	130	168.00	0.00	98.45	132	149.50	0.00
99.42	p0548	0.00	6430	4.37	1.26	93.14	373	168.00	0.00	96.37	369	168.00	0.00
99.90	p2756	0.52	6433	14.95	1.52	85.95	444	168.00	0.01	87.29	467	119.63	0.01
0.00	pk1	0.00	4193	168.00	0.03	0.00	269	168.00	0.00	0.00	265	168.00	0.00
97.03	pp08a	97.04	186	152.16	0.01	97.04	175	150.58	0.00	97.04	184	134.58	0.00
95.81	pp08aCUTS	95.84	176	168.00	0.03	95.80	132	125.56	0.00	95.82	196	83.59	0.00
77.51	qiu	78.09	617	168.00	0.67	78.09	385	168.00	0.64	78.09	419	168.00	0.24
100.00	qnet1	0.09	989	168.00	74.80	97.09	266	168.00	0.04	100.00	230	2.61	0.16
100.00	qnet1_o	0.11	928	168.00	12.32	99.97	208	168.00	0.01	100.00	369	3.51	0.05
23.40	rentacar	54.03	326	23.48	9.51	50.86	307	28.93	8.67	6.93	265	8.66	15.33
100.00	rgn	100.00	438	14.51	0.10	74.55	120	168.00	0.00	75.15	616	107.75	0.00
70.70	rout	0.00	6416	18.34	5.13	50.29	215	168.00	0.05	68.99	293	168.00	0.08
89.74	set1ch	25.26	470	51.64	1.73	89.76	411	12.20	0.01	89.75	342	6.41	0.01
61.52	seymour	0.00	5020	168.00	34.97	7.89	816	168.00	21.73	57.96	1039	168.00	18.67
0.00	stein27	0.00	156	3.88	0.01	0.00	169	1.71	0.01	0.00	125	3.06	0.01
0.00	stein45	0.00	1438	168.00	0.14	0.00	3751	139.54	0.15	0.00	4882	99.76	0.21
33.93	swath	0.00	6400	20.75	67.02	13.00	339	168.00	1.67	28.93	305	168.00	0.54
100.00	vpm1	100.00	297	21.42	0.04	100.00	357	1.68	0.06	100.00	488	0.11	0.34
81.05	vpm2	81.81	204	90.73	0.02	81.31	192	39.05	0.00	81.46	217	22.41	0.00
75.17	average	42.79	1713	97.91	19.55	64.33	476	111.54	7.53	68.95	431	86.64	6.15

Table 4.4: Gap closed for the (i) full split closure, (ii) sparse split cuts only, and (iii) sparse ± 1 split cuts only.

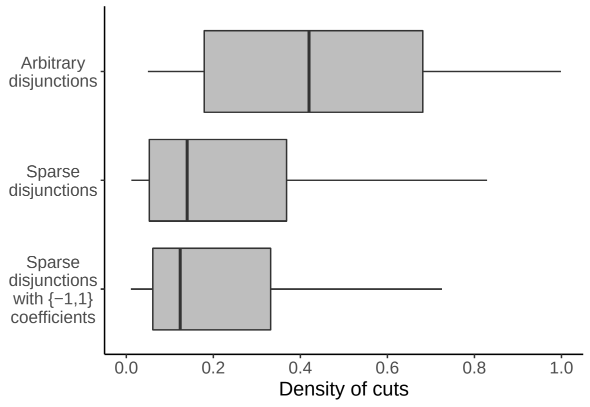


Figure 4.2: Distribution of cut densities with different experimental settings.

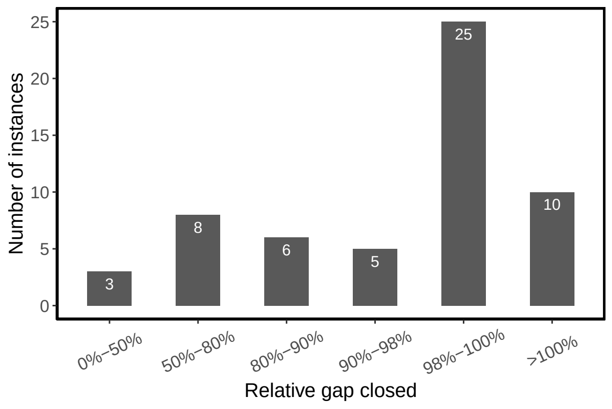


Figure 4.3: Distribution of gap closed with time limit of one week.

4.2.3 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 [37] as a black-box tool to generate the required DB-k forms on MIPLIB 3.0 instances, and then implement our model with the additional structure constraint (C3) on the disjunctions, as described in Chapter 3. Furthermore, for comparison purposes,

- we have kept the sparsity parameter $M = 10$ and coefficient bound $U = 1$ on all 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 4.5 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 last column of Table 4.5 shows 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 second 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 seven instances where no DB-k form was found for any $k \in \{2, 3, 4, 5\}$ (`air03`, `mas74`, `mas76`, `mod008`, `nw04`, `p0033`, `rentacar`).

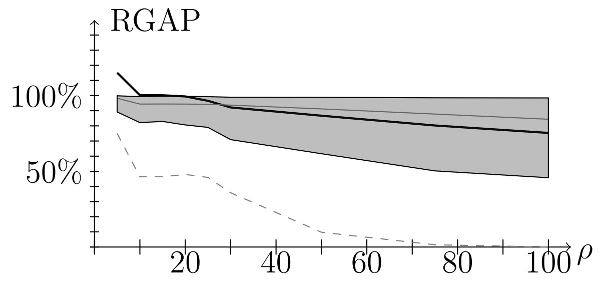
Note that the set of split cuts we used to obtain the results on Table 4.5 is extremely restrictive: (i) the corresponding disjunctions have at most 10 nonzero coefficients which are either 1 or -1, and (ii) the cuts are obtained by aggregating only rows and columns that belong to a single block in a DB-k form. Despite being so selective, these cuts close a significant amount of gap in most cases. In fact, of the 50 instances left, the average gap closed without DB-k is 75.6% and the best gap closed among all DB-k is 56.7%.

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 4.5. The results are summarized in Figure 4.4. 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:

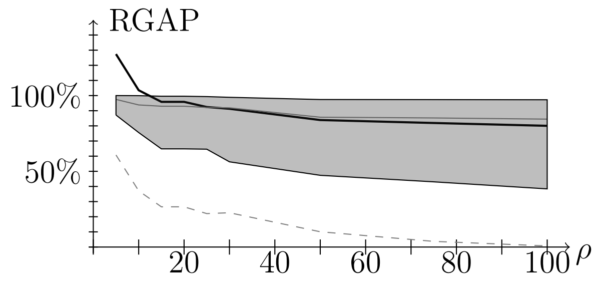
$$\frac{\text{GAP}k}{\text{GAPnodb}}. \tag{RGAP}$$

Instance	DB-2				DB-3				DB-4				DB-5				Best gap with DB-k	Gap without DB-k
	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		
10teams	96.99	1096	168.00	6.97	100.00	807	168.00	7.93	100.00	1415	168.00	3.32	100.00	1295	168.00	1.17	100.00	93.76
air04	45.61	171	110.02	7.45	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	45.61	89.57
air05	0.00	0	1.01	0.00	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00	63.53
arki001	33.03	145	8.76	0.13	33.38	171	42.03	0.05	32.31	173	0.77	0.94	32.31	186	2.89	0.66	33.38	42.61
bell3a	70.74	83	0.00	10.00	70.74	64	0.00	10.00	70.74	59	0.00	0.00	70.66	56	0.00	10.00	70.74	75.73
bell5	92.01	90	0.11	0.05	91.94	84	0.00	2.22	88.99	53	0.01	0.57	86.83	58	0.00	1.25	92.01	92.57
blend2	19.79	14	0.00	0.00	19.79	16	0.00	3.33	19.79	17	0.00	0.00	19.79	13	0.00	0.00	19.79	45.56
cap6000	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	64.69
dano3mip	0.21	205	168.00	26.47	0.25	267	168.00	25.20	0.22	214	168.00	27.97	0.25	246	168.00	36.12	0.25	0.24
danoint	0.77	72	3.68	0.80	0.80	63	0.32	4.28	0.30	30	0.05	6.95	0.72	83	0.05	37.81	0.80	8.98
dcmulti	95.57	196	17.34	0.01	73.24	138	2.09	0.01	69.56	137	5.56	0.02	60.28	130	1.22	0.02	95.57	99.84
egout	95.63	132	0.01	0.70	92.14	90	0.00	2.00	86.56	98	0.00	3.33	93.86	93	0.00	2.31	95.63	98.64
fast0507	0.00	0	0.01	0.00	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00	7.50
fiber	65.50	200	168.00	0.01	73.18	177	168.00	0.01	24.22	90	48.54	0.01	15.77	52	13.48	0.01	73.18	72.99
fixnet6	84.26	274	1.54	0.08	83.56	319	0.08	0.49	84.43	366	0.05	1.05	84.86	413	0.03	3.80	84.86	99.86
flugpl	90.16	18	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	98.49
gen	100.20	225	0.12	3.08	100.20	203	0.54	0.47	100.20	380	0.04	9.47	100.20	392	0.02	9.19	100.20	100.20
gesa2	99.97	192	20.38	0.03	99.95	214	8.28	0.02	99.95	253	0.43	0.42	99.94	209	21.69	0.01	99.97	100.00
gesa2_o	44.60	125	19.68	0.00	38.67	166	2.50	0.02	38.59	161	10.73	0.01	38.60	155	4.63	0.01	44.60	99.99
gesa3	95.87	190	30.06	0.01	95.87	119	28.00	0.01	95.84	206	47.65	0.00	96.03	290	32.57	0.01	96.03	96.01
gesa3_o	95.85	169	66.91	0.01	96.08	184	119.44	0.00	46.56	238	2.59	0.04	60.21	208	8.91	0.01	96.08	96.07
gt2	0.00	0	0.00	NA	0.00	0	0.00	NA	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	98.89
harp2	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	45.21
khb05250	53.48	19	0.00	2.00	53.48	19	0.00	1.67	53.48	19	0.00	1.67	53.48	19	0.00	1.11	53.48	100.00
l152lav	0.24	23	0.84	0.04	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.24	48.93
lseu	52.62	64	0.00	1.00	19.94	52	0.00	2.00	38.58	37	0.00	0.00	4.21	5	0.00	0.00	52.62	76.16
misc03	0.00	6	0.00	NA	0.00	0	0.00	NA	0.00	0	0.00	NA	0.00	0	0.00	NA	0.00	51.49
misc06	94.18	45	0.00	13.33	26.55	27	0.00	5.00	68.14	39	0.00	10.00	97.51	52	0.00	9.29	97.51	100.00
misc07	0.00	54	0.25	0.40	0.00	0	0.00	0.00	0.00	0	0.00	NA	NA	NA	NA	NA	0.00	16.62
mitre	55.23	2098	168.00	0.63	63.34	3758	168.00	0.46	64.02	2756	168.00	0.59	65.93	2160	168.00	0.56	65.93	23.52
mkc	62.90	467	168.00	0.35	69.08	198	168.00	0.22	69.84	543	168.00	0.20	72.91	353	168.00	0.19	72.91	65.11
mod010	0.00	0	2.41	0.00	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.00	100.00
mod011	89.12	1129	168.00	0.92	92.51	1256	168.00	0.85	89.01	1388	168.00	1.03	90.61	1217	168.00	1.00	92.51	85.35
modglob	93.08	161	4.30	0.02	85.58	117	0.68	0.05	85.87	128	0.10	0.38	83.64	119	0.05	0.65	93.08	95.49
p0201	54.14	93	78.58	0.00	50.00	103	10.39	0.01	0.00	0	0.00	NA	0.00	0	0.00	NA	54.14	71.31
p0282	3.51	79	0.01	0.26	83.88	52	0.00	0.59	1.52	10	0.00	0.00	83.87	35	0.01	0.80	83.88	98.45
p0548	90.20	278	114.73	0.00	89.83	287	167.93	0.00	89.73	313	168.00	0.00	88.20	344	146.88	0.00	90.20	96.37
p2756	88.29	408	62.77	0.01	86.99	427	17.61	0.02	85.17	540	13.44	0.02	86.04	478	17.91	0.01	88.29	87.29
pp08a	95.53	188	0.08	0.10	94.28	169	0.03	0.53	95.59	184	0.02	0.57	95.18	201	0.01	2.07	95.59	97.04
pp08aCUTS	93.73	159	0.43	0.10	92.80	186	0.03	0.98	93.37	214	0.02	3.11	89.06	185	0.02	2.00	93.73	95.82
qiu	78.09	262	168.00	0.15	0.00	0	0.16	0.00	78.09	946	75.38	0.31	0.00	0	0.16	0.00	78.09	78.09
qnet1	2.44	23	2.14	0.00	2.45	42	1.30	0.01	2.45	42	3.48	0.00	1.90	29	2.33	0.00	2.45	100.00
qnet1_o	20.96	9	0.00	NA	20.96	9	0.00	NA	20.96	6	0.00	NA	20.96	6	0.00	0.00	20.96	100.00
rgn	68.40	187	2.18	0.03	48.20	115	0.01	1.74	35.86	116	0.00	3.75	NA	NA	NA	NA	68.40	75.15
rout	42.00	319	107.82	0.05	44.14	393	33.88	0.08	41.75	282	33.04	0.05	71.42	513	164.45	0.03	71.42	68.99
set1ch	89.73	332	0.17	0.40	89.73	349	0.26	0.54	89.73	452	0.15	0.89	89.73	427	0.08	1.07	89.73	89.75
seymour	35.94	439	168.00	21.80	53.89	708	168.00	23.00	54.63	481	168.00	23.02	60.48	918	168.00	24.04	60.48	57.96
swath	19.08	107	0.05	10.53	16.65	73	0.06	3.39	11.26	87	0.01	31.59	12.78	120	0.01	31.11	19.08	28.93
vpm1	78.18	189	0.01	1.90	34.55	115	0.00	2.22	34.55	101	0.03	0.18	42.42	85	0.00	2.86	78.18	100.00
vpm2	73.85	135	0.92	0.02	74.59	157	0.16	0.21	57.52	132	0.02	0.55	66.38	182	0.01	1.60	74.59	81.46
average	53.23	217	40.03	2.34	53.96	260	35.82	2.37	49.48	282	31.51	3.22	53.09	264	33.15	4.41	56.73	75.60

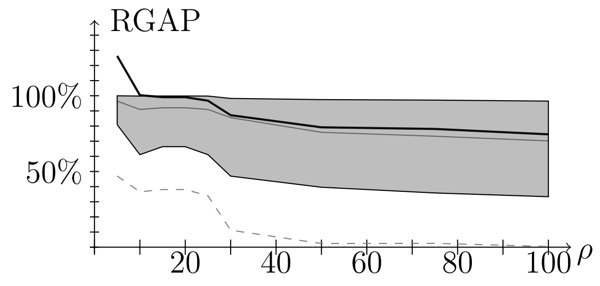
Table 4.5: Gap closed with sparse ±1 splits and DB-k structures



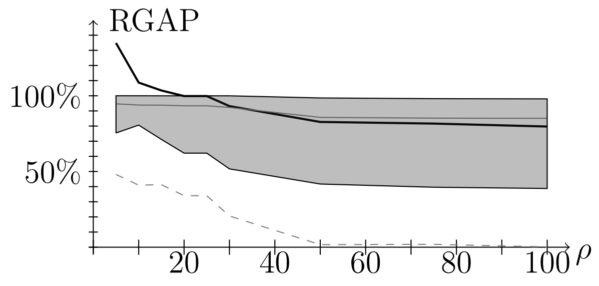
(a) $k = 2$



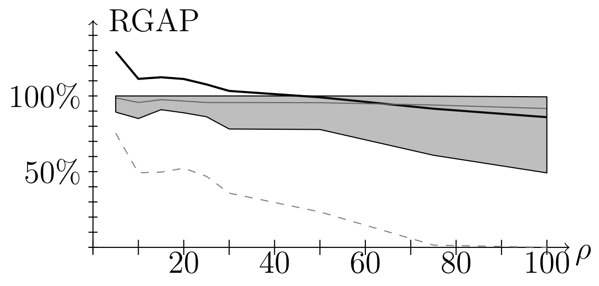
(b) $k = 3$



(c) $k = 4$



(d) $k = 5$



(e) best

Figure 4.4: Distribution of relative gap closed for DB-k forms for several values of ρ

The following statistics are shown in Figure 4.4 for the instances remaining after filtering:

- 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 4.4 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, especially when ρ is small. We now try to understand how structured sparsity helps to produce more effective cuts. Table 4.6 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 (8.3 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 4.6 shows that disjunctions with arbitrarily many nonzero entries that are much denser still lead to sparse cuts when exploiting problem structure.

In Figure 4.5 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 4.5, 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 0.067 to 0.047, and the 75 percentile from 0.115 to 0.080.

Finally, we illustrate one more potential advantage of using split cuts based on DB-k forms. Figure 4.6 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 4.6a) and in terms of number of cuts (Figure 4.6c). 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 4.6b). The grey lines labelled "DB-k*" in Figure 4.6 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[4].

Instance	Disjunction type	Average support size of disjunctions (#)	Average cut density (%)
`10teams`	$M = 10,\ U = 1$	8.3	95.5
`10teams`	$M = 10,\ U = 1$, with `DB-2`	8.2	54.6
`mkc`	$M = 10,\ U = 1$	9.4	12.6
`mkc`	$M = 10,\ U = 1$, with `DB-2`	9.2	3.8
`seymour`	$M = 10,\ U = 1$	9.8	9.6
`seymour`	$M = 10,\ U = 1$, with `DB-2`	8.2	3.7
`seymour`	$M = +\infty,\ U = 1$, with `DB-2`	255.2	10.3

Table 4.6: Disjunction and cut density for three example instances.

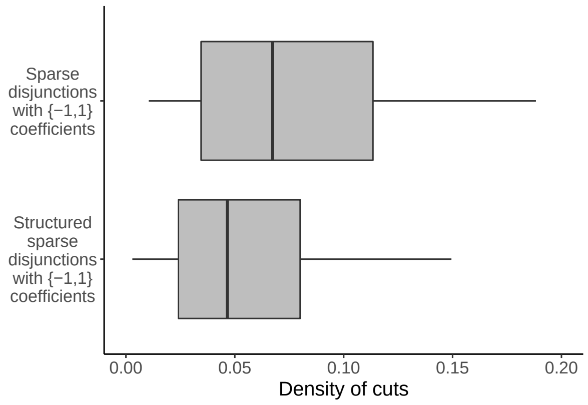
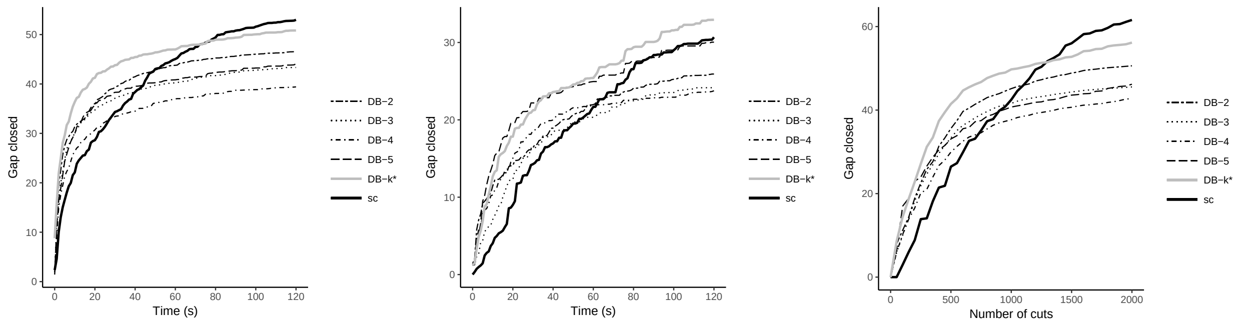


Figure 4.5: Distribution of cut densities for `DB-2`

[4]It is clear from CGLP(π, π_0) and the shape of `DB-`k forms that, if a split disjunction has a support that lies entirely within a block, then the support of the corresponding split cut lies entirely in that block, too. Therefore, if two disjunctions each has a support in a different block, then the split cuts from these disjunctions will be orthogonal (in the sense that the dot product of their coefficients are zero).



(a) Average gap closed over time

(b) Average gap closed over time for large instances

(c) Average gap closed per cuts added

Figure 4.6: Evolution of average gap closed

4.3 Computing (restricted) split closure 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 [5] 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 4.7 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 [21]. 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 integrality gap.

L&P	Str. L&P	Instance	Without DB-k				DB-2				DB-3				DB-4				DB-5				Best
gap	gap		Gap	# cuts	Time	% time	Gap	# cuts	Time	% time	Gap	# cuts	Time	% time	Gap	# cuts	Time	% time	Gap	# cuts	Time	% time	gap
[21]	[21]		closed	binding	(h)	checking	closed	binding	(h)	checking	closed	binding	(h)	checking	closed	binding	(h)	checking	closed	binding	(h)	checking	(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 4.7: Gap closed in MIPLIB2003 instances.

Chapter 5

Conclusion

The family of split cuts in general provides very tight relaxations to the feasible sets of MIPs [35], yet the computational cost of separating an arbitrary split cut is prohibitive both in theory [23] and in practice [36]. The main motivation for this thesis was to search for subsets of split cuts with promising computational properties. In particular, our goal is to address the following questions:

- How much can we restrict the set of split cuts that we separate over, while retaining enough of the strength of the full split closure?
- Is there a systematic way to obtain split cuts with desirable computational characteristics?

In this thesis we developed a tool, which builds on the work of Balas and Saxena's separation algorithm [13], to empirically evaluate the strengths of several restricted classes of split cuts. 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. Second, 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 [13, 36] 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 92% as effective as all split cuts together (in terms of gap closed, and compared to the best known results for the split closure [13,29]).

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 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. We test this approach with arrowhead decompositions [15,37] 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. Therefore, a possible continuation of this work is to look at which subsets of split cuts can be computed more quickly in practice. For example, lift-and-project cuts are computed much more quickly in [21] than in our setting where we simply set $M = 1$ and $U = 1$. However, even a tiny step up to $M = 2$ in the number of nonzero elements in split disjunctions is shown to produce a significantly tighter relaxation (c.f. Appendix A.3 and A.4). Thus, it is interesting to determine if this small class of split cuts are easy to separate over, both in theory and in practice. Other possible research directions are:

- determine a more accurate method, possibly more sophisticated than simply looking at the number of linking constraints and linking variables, to classify MIP instances that are suitable or not so suitable for decomposition-based separation of split cuts; and moreover,
- dynamically find the "best" decomposition (or the "best" combination of decompositions) that will lead to strong split cuts.

We expect that for example tools from machine learning may help addressing these questions. There are already some recent work on applying supervised learning to MIP solution processes, e.g., [6] and [41].

References

[1] FICO Xpress Optimization. http://www.fico.com/en/products/fico-xpress-optimization. Accessed: 2018-09-10.

[2] Gurobi Optimization. http://www.gurobi.com. Accessed: 2018-09-10.

[3] IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/analytics/cplex-optimizer. Accessed: 2018-09-10.

[4] MIPLIB. http://miplib.zib.de. Accessed: 2018-10-30.

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

[6] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A supervised machine learning approach to variable branching in branch-and-bound. In *IN ECML*, 2014.

[7] 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.

[8] 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.

[9] Egon Balas. Disjunctive programming. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 3 – 51. Elsevier, 1979.

[10] Egon Balas. A modified lift-and-project procedure. *Mathematical Programming*, 79(1):19–31, Oct 1997.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[17] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

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

[19] 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.

[20] 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.

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

[22] 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.

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

[24] Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming*. Springer Publishing Company, Incorporated, 2014.

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

[26] Gérard Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, Mar 2008.

[27] 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.

[28] Gérard Cornuéjols and Yanjun Li. Elementary closures for integer programs. *Operations Research Letters*, 28(1):1 – 8, 2001.

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

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

[31] 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.

[32] Friedrich Eisenbrand. Note – on the membership problem for the elementary closure of a polyhedron. *Combinatorica*, 19(2):297–300, Feb 1999.

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

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

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

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

[37] 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.

[38] Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin, July 2018.

[39] Ralph E. Gomory. Some polyhedra related to combinatorial problems. *Linear Algebra and its Applications*, 2(4):451 – 558, 1969.

[40] C. Guéret, C. Prins, and M. Servaux. *Applications of optimization with Xpress - MP*. Dash Optimization Ltd., London, 2002.

[41] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 724–731. AAAI Press, 2016.

[42] 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.

[43] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0–1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[44] Manfred W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1):199–215, Dec 1973.

[45] A. Schrijver. On cutting planes. In Peter L. Hammer, editor, *Combinatorics 79*, volume 9 of *Annals of Discrete Mathematics*, pages 291 – 296. Elsevier, 1980.

[46] H. Sherali and W. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[47] 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.

[48] 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.

APPENDICES

Appendix A

Additional computational results

A.1 $U = 1$, $M = 1$, MIPLIB 3.0

	L&P	$U = 1, M = 1$			
Instance	gap	Gap	# cuts	Time	% time
	[21]	closed	binding	(s)	checking
`10teams`	30.41	32.26	2260	604800	1.05
`air03`	100.00	100.00	54	2	60.00
`air04`	87.38	90.13	551	604800	21.58
`air05`	85.81	61.88	441	604800	9.51
`arki001`	20.34	19.59	328	3178	0.93
`bell3a`	64.46	64.56	76	3	0.00
`bell5`	86.25	86.25	110	20	0.00
`blend2`	21.82	21.82	47	58	0.69
`cap6000`	50.00	46.67	41	593	1.32
`dano3mip`	0.52	0.47	639	604800	7.72
`danoint`	5.53	5.63	1147	131888	1.07
`dcmulti`	98.15	98.15	476	1060	0.43
`egout`	93.85	93.85	203	33	0.61
`fast0507`	9.95	16.50	354	604800	22.94
`fiber`	20.63	20.63	646	7613	0.03
`fixnet6`	86.36	86.37	876	2014	0.46
`flugpl`	11.72	11.74	25	2	0.00
`gen`	70.49	70.41	326	407	1.33
`gesa2`	59.10	59.08	268	623	0.30
`gesa2_o`	59.80	59.78	312	1032	0.21
`gesa3`	79.89	79.75	513	1293	0.36
`gesa3_o`	82.88	82.68	680	1382	0.25
`gt2`	92.38	92.01	74	4	0.00
`harp2`	21.28	20.66	283	196356	0.03
`khb05250`	99.86	99.86	325	98	4.49
`l152lav`	34.46	24.16	345	15612	1.27
`lseu`	16.58	16.58	127	12	0.83
`mas74`	5.47	5.45	86	80	1.12
`mas76`	3.68	3.68	99	61	0.66
`misc03`	40.31	40.21	313	184	0.92
`misc06`	86.21	81.93	121	35	4.00
`misc07`	11.44	11.44	358	8338	0.20
`mitre`	100.00	100.00	6674	44523	0.36
`mkc`	63.18	63.48	26109	604800	0.05
`mod008`	9.02	9.03	87	19	1.58
`mod010`	52.51	52.49	418	5663	2.85
`mod011`	55.39	79.65	1355	604800	0.77
`modglob`	57.09	57.08	182	350	0.31
`nw04`	38.85	40.02	825	146178	6.77
`p0033`	8.19	8.19	79	6	1.67
`p0201`	46.85	46.85	171	115	0.52
`p0282`	93.90	93.90	332	698	0.30
`p0548`	91.36	91.48	821	604800	0.00
`p2756`	81.60	82.29	846	34841	0.02
`pk1`	NA	0.00	1054	4657	0.13
`pp08a`	79.29	79.29	243	86	0.23
`pp08aCUTS`	68.81	68.82	282	242	0.25
`qiu`	78.09	78.09	2815	604800	1.03
`qnet1`	94.28	94.28	307	154	1.62
`qnet1_o`	87.59	87.59	309	154	0.45
`rentacar`	91.35	58.71	404	604800	9.65
`rgn`	11.88	11.88	255	26	1.54
`rout`	28.01	27.98	594	40163	0.09
`set1ch`	39.88	39.88	221	1841	0.03
`seymour`	55.94	61.77	860	604800	5.30
`stein27`	0.00	0.00	84	48	1.04
`stein45`	0.00	0.00	598	10672	0.31
`swath`	2.77	3.68	316	629	7.54
`vpm1`	31.42	31.42	159	79	0.38
`vpm2`	54.29	54.29	240	211	0.28
average	50.98	50.44	986	122743	3.16

A.2 $U = 1$, $M = 1$, MIPLIB 2003

Instance	L&P gap [21]	$U = 1, M = 1$ Gap closed	# cuts binding	Time (s)	% time checking
a1c1s1	78.76	78.77	831	76709	0.07
aflow30a	42.41	42.45	856	10356	0.11
aflow40b	34.29	34.29	1363	22829	0.17
atlanta-ip	1.09	0.22	29	200108	3.31
ds	NA	7.88	967	604800	81.23
glass4	0.00	0.00	102	10527	0.05
manna81	NA	100.00	456	3736	0.14
momentum1	44.88	65.67	1305	604800	14.63
momentum2	41.47	68.59	1181	604800	24.60
msc98-ip	42.23	0.00	0	4885	0.00
mzzv11	56.47	68.33	803	604800	53.82
mzzv42z	87.73	81.97	616	604800	63.93
net12	22.73	25.46	3298	604800	17.88
nsrand-ipx	36.88	36.88	1008	177603	0.04
opt1217	0.19	0.19	603	3531	2.28
protfold	10.29	40.32	556	604800	19.70
rd-rplusc-21	0.00	0.00	476	479317	0.11
roll3000	16.31	16.31	613	604800	1.08
sp97ar	42.06	47.04	761	541947	0.78
timtab1	26.99	26.99	233	697	0.07
timtab2	20.98	21.65	272	8132	0.02
tr12-30	64.12	64.12	738	26168	0.01
average	35.35	37.60	776	291134	12.91

A.3 $U = 1$, $M = 2$, MIPLIB 3.0

	L&P	$U = 1, M = 2$			
Instance	gap	Gap	# cuts	Time	% time
	[21]	closed	binding	(s)	checking
10teams	30.41	49.52	764	604800	2.97
air03	100.00	100.00	51	3	90.00
air04	87.38	90.54	508	604800	35.28
air05	85.81	63.13	299	604800	6.68
arki001	20.34	29.02	409	21342	0.15
bell3a	64.46	69.09	80	12	0.83
bell5	86.25	87.54	159	75	0.53
blend2	21.82	24.27	109	251	0.24
cap6000	50.00	58.05	39	925	2.00
dano3mip	0.52	0.52	409	604800	14.99
danoint	5.53	7.49	607	604800	0.13
dcmulti	98.15	100.00	651	341	1.00
egout	93.85	100.00	175	40	0.75
fast0507	9.95	16.92	494	604800	36.39
fiber	20.63	33.82	472	604800	0.00
fixnet6	86.36	89.73	678	25831	0.02
flugpl	11.72	13.34	44	5	0.00
gen	70.49	83.71	291	10878	0.08
gesa2	59.10	77.21	317	19523	0.03
gesa2_o	59.80	77.41	342	20924	0.01
gesa3	79.89	96.02	663	33012	0.05
gesa3_o	82.88	96.02	592	22314	0.03
gt2	92.38	92.62	90	127	0.08
harp2	21.28	34.20	159	74382	0.03
khb05250	99.86	100.00	410	675	1.14
l152lav	34.46	32.44	330	582244	0.07
lseu	16.58	34.96	144	630	0.05
mas74	5.47	7.31	94	459	0.17
mas76	3.68	5.22	80	369	0.14
misc03	40.31	40.21	257	4518	0.04
misc06	86.21	100.00	217	77	8.96
misc07	11.44	13.36	326	229485	0.01
mitre	100.00	100.00	17973	365986	0.28
mkc	63.18	66.46	1758	604800	0.18
mod008	9.02	16.62	109	249	0.32
mod010	52.51	82.32	373	209606	0.11
mod011	55.39	81.88	1193	604800	0.75
modglob	57.09	90.94	245	8584	0.02
nw04	38.85	55.40	497	604800	2.94
p0033	8.19	15.23	82	99	0.10
p0201	46.85	69.37	291	15323	0.02
p0282	93.90	96.54	246	6159	0.03
p0548	91.36	95.49	439	604800	0.00
p2756	81.60	86.96	717	239467	0.00
pk1	0.00	0.00	752	98794	0.01
pp08a	79.29	94.42	207	5711	0.01
pp08aCUTS	68.81	88.72	288	2246	0.09
qiu	78.09	78.09	746	604800	0.17
qnet1	94.28	98.96	375	190146	0.01
qnet1_o	87.59	93.00	371	47003	0.00
rentacar	91.35	26.33	132	1367	20.72
rgn	11.88	22.16	232	514	0.14
rout	28.01	43.99	336	604800	0.01
set1ch	39.88	65.61	225	29366	0.00
seymour	55.94	62.12	783	604800	8.78
stein27	0.00	0.00	179	297	0.13
stein45	0.00	0.00	2199	565672	0.15
swath	2.77	10.15	305	195594	0.14
vpm1	31.42	43.63	232	1699	0.04
vpm2	54.29	70.57	230	7137	0.01
average	50.98	57.98	696	191778	3.97

A.4 $U = 1$, $M = 2$, MIPLIB 2003

Instance	L&P gap [21]	$U = 1, M = 2$ Gap closed	# cuts binding	Time (s)	% time checking
a1c1s1	78.76	91.62	792	490411	0.02
aflow30a	42.41	58.86	577	604800	0.00
aflow40b	34.29	46.45	694	604800	0.01
atlanta-ip	1.09	8.19	626	578238	5.18
ds	NA	9.23	1142	604800	68.75
glass4	0.00	0.28	206	21661	0.16
manna81	NA	100.00	1579	22549	0.08
momentum1	44.88	65.81	1245	604800	18.85
momentum2	41.47	67.93	914	604800	32.35
msc98-ip	42.23	0.00	0	268	0.00
mzzv11	56.47	60.44	337	604800	69.81
mzzv42z	87.73	76.02	850	604800	65.04
net12	22.73	23.35	2311	604800	16.33
nsrand-ipx	36.88	48.84	1001	604800	0.03
opt1217	0.19	0.43	682	166681	0.04
protfold	10.29	38.99	786	604800	26.92
rd-rplusc-21	0.00	0.00	669	522527	0.62
roll3000	16.31	26.33	567	604069	0.58
sp97ar	42.06	55.19	282	604800	0.97
timtab1	26.99	53.80	260	37060	0.00
timtab2	20.98	43.92	309	102548	0.00
tr12-30	64.12	85.49	773	574047	0.00
average	35.35	43.70	755	444221	13.90

Appendix B

Effect of heuristics in computation

In order to examine the effect of some features and modifications we introduced to our separation routine, we ran the code on MIPLIB 3.0 instances with $M = +\infty$ and $U = 100$. We set the global time limit to an hour, and disabled the following features one at a time:

- cut strengthening (`cut_str_off`);
- stabilizing objective (`stb_obj_off`);
- set covering (`set_cov_off`).

We then plotted, for each scenario, the number of instances on which at least $x\%$ integrality gap was closed, for $x \in \{10, 20, \ldots, 90\}$. As shown in Figure B.1, the additional features indeed helped to obtain better results.

Among the three heuristics compared, using set covering to impose partial orthogonality between split cuts plays the most important role. The advantage of stabilizing objective is also evident. Note that, these results were obtained with a bound $U = 100$ already imposed on the disjunction coefficients. We thus expect to observe a more dramatic gain from stabilized objective, had we chosen a larger bound U to begin with. Table B.1 shows the average gap closed in each computational setting. Note that we were able to close significantly more integrality gap when all features were used (`default`).

Setting	Average gap closed
`default`	31.04%
`cut_str_off`	28.35%
`stb_obj_off`	21.32%
`set_cov_off`	12.41%

Table B.1: Effect of heuristics

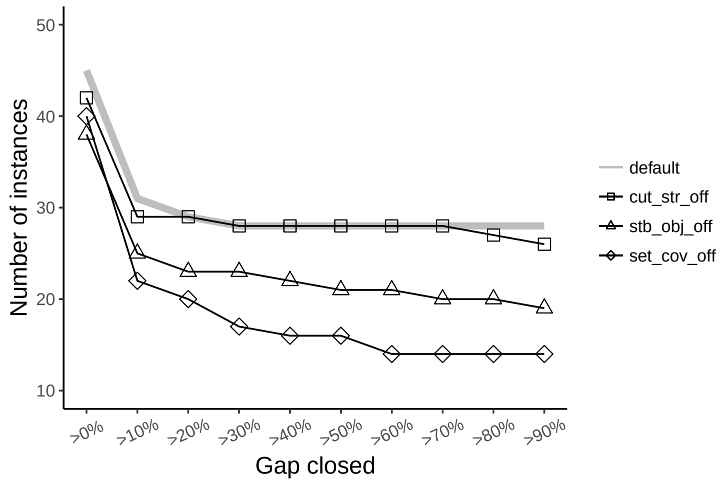


Figure B.1: Effect of heuristics