Dissertation

Generating General-Purpose Cutting Planes for Mixed-Integer Programs

Dipl.-Wirt.-Inform. Franz Wesselmann

Schriftliche Arbeit zur Erlangung des akademischen Grades
doctor rerum politicarum (dr. rer. pol.)
im Fach Wirtschaftsinformatik

eingereicht an der
Fakultät für Wirtschaftswissenschaften der
Universität Paderborn

Gutachter:
1. Prof. Dr. Leena Suhl
2. Prof. Dr. Quentin Louveaux

Paderborn, im November 2010

And how is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?

Homer Simpson

Acknowledgements

This thesis is the result of the research work I carried out as a member of the Decision Support & Operations Research (DS&OR) Lab at the University of Paderborn. Completing this dissertation would not have been possible without the support of many people.

First and foremost, I would like to express my deepest gratitude to my supervisor Prof. Dr. Leena Suhl for giving me the opportunity to join her research group and to write this thesis. Her constant support and valuable guidance over the years contributed crucially to the completion of this work. I would like to thank Prof. Dr. Uwe Suhl for his support, many fruitful discussions and for sharing his long-time experience in developing software for solving linear and mixed-integer programming problems with me. I am very grateful to Prof. Dr. Quentin Louveaux for writing the second report on my thesis.

I furthermore wish to thank all my past and present colleagues at the DS&OR Lab for their close collaboration and for providing an excellent working atmosphere. I really enjoyed the stimulating discussions on scientific topics as well as the interesting and amusing discussions on non-work related topics during coffee and lunch breaks. In particular, I would like to thank my office colleagues Philipp Christophel, Achim Koberstein, Thomas Siebers and Daniel Rudolph for their support. I also wish to thank Viktor Dück, Stefan Kramkowski, Christian Temath and Jörg Wiese for their advice and encouragement during the time of writing this thesis.

Last but not least, I wish to thank my friends and family, especially my parents, my brothers and my godfather for believing in me and supporting me. Special thanks are due to Sonja for her encouragement, patience, understanding and for reminding me from time to time that there is more to life than a thesis.

Paderborn, November 2010 Franz Wesselmann

Contents

Chapter 1.

Introduction

Mixed-integer programming is a branch of mathematical programming concerned with optimization problems in which a linear objective function is maximized (or minimized) subject to linear constraints and integrality requirements on some of the variables. Early work on mixed-integer programming dates from the 1950s when, for instance, George B. Dantzig [67] demonstrated that various combinatorial optimization problems can be formulated as mixed-integer programs (MIPs). Dantzig [68] also noted that many problems involving complex logical conditions, non-linear separable functions and non-convex regions can be transformed into MIPs. Today mixed-integer programming is a widely used tool for modeling and solving real-world optimization problems. Applications originate in various domains such as telecommunication, public transport or production planning (see, for instance, Guéret et al. [110]). The versatility of mixed-integer programming also generated strong interest in solution methods for MIPs. A number of efficient software packages for solving MIPs were developed, including the commercial products Cplex [115], Xpress-MP [74] and Mops [139], the open-source MIP solver Cbc [1] and the constraint integer programming solver Scip [2] which is free for academic use.

Solving mixed-integer programs is an intricate task. From a theoretical point of view the computational complexity of solving MIPs is high. It is well known that mixed-integer programming is $\mathcal{NP}$-hard (cf. Schrijver [152]), which means that there is most likely no polynomial-time exact algorithm for solving MIPs. In practice there are MIP instances which take several hours or days to solve and others which can not be solved to optimality at all by today's state-of-the-art

MIP solvers. Such large computation times are often unacceptable in practical applications.

The first algorithms for solving MIPs were proposed by Gomory [97] and Land and Doig [127]. The branch-and-bound algorithm of Land and Doig performs an implicit enumeration of the solution space which can be represented by a search tree. Since optimizing the objective function over the convex hull of the feasible solutions of an MIP is equivalent to solving the MIP itself, an alternative approach for solving MIPs is, in theory, to find a complete description of the convex hull of the feasible region. If we could compute the convex hull, we would be able to obtain an optimal MIP solution by solving a linear program. Linear programs are solvable in polynomial time (cf. Khachiyan [118]) and general solution algorithms for them, such as the simplex algorithm and interior point methods, are efficient in practice. Obtaining a complete description of the convex hull is, however, as hard as solving the MIP itself (cf. Grötschel et al. [106]). Gomory's algorithm thus approximates the convex hull of the feasible solutions of an MIP by adding so-called cutting planes to the problem formulation. Cutting planes (or cuts) are linear inequalities which are satisfied by all feasible solutions of an MIP but not by all feasible solutions of its linear programming (LP) relaxation, which is obtained by omitting the integrality restrictions on the integer-constrained variables.

Regardless of the theoretical complexity of MIP, the last two decades have brought enormous performance improvements in standard software for solving MIPs, and these for several reasons. Besides faster computers and improvements to the simplex method (cf. Koberstein [123]), enhanced cutting plane techniques brought about major reductions in the times needed to solve many MIPs to proven optimality (see Bixby et al. [38]). State-of-the-art MIP solvers use both the branch-and-bound algorithm and cutting planes. For each node in the branch-and-bound tree the LP relaxation is solved. Cutting planes are used to strengthen the LP relaxation by removing fractional solutions, thus reducing the search space of the branch-and-bound algorithm. In consequence, cutting planes can lead to a decrease in the number of enumeration nodes explored by the branch-and-bound algorithm.

There are various classes of cutting planes for mixed-integer programs which are generated from particular relaxations of MIPs with certain characteristics. Cutting planes can, for instance, be categorized by the amount of information about problem structure they use. There are cutting planes which can only be applied if the MIP has a special structure, i.e. if specific types of constraints are present. On the other hand, there are general-purpose cutting planes, also referred to as cutting planes for unstructured MIPs, which can be applied independently of any problem structure. Another possible categorization of cutting planes for mixed-integer programs is based on the number of constraints in the relaxation used for cut generation. Cutting planes can in particular also be derived from multiple-constraint relaxations of MIPs.

In this thesis we focus on general-purpose cutting planes for mixed-integer programs. We particularly concentrate on split cuts such as Chvátal-Gomory cuts [53] and Gomory mixed-integer cuts [99]. Split cuts are known crucially to affect the overall performance of MIP solvers. For instance, Bixby et al. [38] detected the Gomory mixed-integer cuts to be the most effective cutting planes in Cplex 8.0. We discuss several approaches in the literature which seek to improve the performance of the Gomory mixed-integer cuts; see Cornuéjols et al. [61], Ceria et al. [46], Andersen et al. [8] and Balas et al. [24]. We also propose a new heuristic algorithm which obtains improved Gomory mixed-integer cuts by performing a sequence of pivots on the simplex tableau. Moreover, we give a detailed description of our implementation of the discussed approaches and assess their practical usefulness based on computational experiments.

Multi-row cuts are known to play an important role in describing the convex hull of mixed-integer sets. Cook et al. [57] provide an example with a mixed-integer set whose convex hull of feasible solutions can not be obtained by repeated application of split cuts such as the Gomory mixed-integer cuts, i.e. there is a facet-defining inequality which does not have finite split rank. This facet-defining inequality can, however, be obtained as a multi-row cut. Multi-row cuts recently gained renewed interest; see, for instance, Andersen et al. [11], Dey and Wolsey [78–80] and Basu et al. [32, 33, 35, 36]. In this thesis we study cutting planes which are generated using more than one row of the simplex tableau simultaneously. We describe a separation algorithm and highlight important implementation details. We moreover evaluate the strength of multi-row cuts computationally and compare them with split cuts like the Gomory mixed-integer cuts.

Cut generation routines typically produce a large number of cutting planes violated by the current optimal solution of the LP relaxation. Adding all of these cutting planes to the problem formulation increases the size of the LP relaxation, making it more difficult to solve. In particular, the increased size of the LP relaxation repeatedly leads to higher node solution times during the branch-and-bound algorithm. In this thesis we discuss techniques for cutting plane selection and management to cope with the complexity introduced by the number of generated cutting planes.

The remainder of this thesis is structured in three parts. Part I comprises two chapters and introduces the basic concepts and notation used in this thesis. In Chapter 2 we present an introduction to mixed-integer programming and in Chapter 3 we discuss the basic algorithms for solving mixed-integer programs. Part II consists of three chapters and is concerned with the state-of-the-art in cutting plane methods. In Chapter 4 we consider single-row cuts such as Gomory mixed-integer cuts and Chvátal-Gomory cuts. In Chapter 5 we discuss the derivation of cutting planes from multi-row relaxations. Chapter 6 describes in detail the goals of this thesis. Part III comprises five chapters and is devoted to computational techniques and experiments. In Chapter 7 we introduce the MIP solver MOPS, on which our work is based and discuss some important aspects of our implementation. Chapter 8 concentrates on the separation of Chvátal-Gomory cuts and Gomory mixed-integer cuts. We describe implementations of a variety of algorithms designed to improve the performance of the latter class of cutting planes and compare their computational results. In Chapter 9 we present a scheme for generating cutting planes from multiple rows of a simplex tableau and again highlight some important implementation details. We in addition report on computational experiments assessing the effectiveness of multi-row cuts and compare them with split cuts. In Chapter 10 we describe a cut selection scheme. Chapter 11 summarizes the main results of this thesis, offers some conclusions and suggests possibilities for further research.

Part I.

Foundations

Chapter 2.

Integer Programming Preliminaries

In this chapter we present an introduction to mixed-integer programming and discuss the basic concepts and notation we work with in the remainder of this thesis. We formally define the mixed-integer programming problem and introduce its linear programming relaxation and disjunctive relaxation. We also highlight the role these relaxations play in solving mixed-integer programs. We discuss in particular the concept of a tight LP relaxation.

This chapter is structured as follows. Section 2.1 defines mixed-integer programs. The linear programming relaxation is discussed in Section 2.2. Section 2.3 treats of disjunctive relaxations of mixed-integer programs.

2.1. Mixed-Integer Programs

Consider the *mixed-integer program*

$$\text{(MIP)} \quad \min \{cx : Ax \geq b, x \geq 0, x_j \in \mathbb{Z}, \forall j \in N_I\} \tag{2.1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and $N_I \subseteq N = \{1, \dots, n\}$. An MIP consists of a *linear objective function*, *linear constraints* and *lower and upper bounds* on the variables. Moreover, some of the variables are *constrained to take integer values.* The form in which the MIP (2.1) is given is referred to as the *standard inequality form* as all variables have a lower bound of zero. If $N_I = \emptyset$ we obtain a *linear program.* On the other hand, if $N \setminus N_I = \emptyset$ we have a *pure integer program.* The set of *feasible solutions* to (2.1) is given by

$$X_{MIP} = \{x \in \mathbb{R}^n : Ax \geq b, x \geq 0, x_j \in \mathbb{Z}, \forall j \in N_I\}\,. \tag{2.2}$$

Let $\bar{c}$ be the optimal (minimal) objective value of (2.1). A feasible solution $\bar{x} \in X_{MIP}$ is called *optimal* if $c\bar{x} = \bar{c}$. Finding a feasible solution for an MIP generally is $\mathcal{NP}$-hard (cf. Schrijver [152]).

It is sometimes more natural to consider an MIP in *standard equality form* or *equality form*

$$\text{(MIP)} \quad \min\{cx : Ax = b, x \geq 0, x_j \in \mathbb{Z}, \forall j \in N_I\} \tag{2.3}$$

where c, b, A, and N_I are defined as above. We assume that the matrix A has *full row rank*. This assumption is always satisfied if (2.3) arises from (2.1). As before, we denote by X_{MIP} the feasible region of the MIP (2.3). In Parts I and II of this thesis we use MIPs in the standard forms to discuss theory and algorithms. It will be noted or should become clear from the context whether an MIP of the form (2.1) or (2.3) is used. These representations of an MIP are equivalent, i.e. any MIP can be transformed from the form (2.1) to the form (2.3) and vice versa.

In the following sections we discuss two *relaxations*[1] of MIPs. Relaxations are primarily constructed in the hope that they are easier to analyze and solve than the original MIPs. As all feasible solutions to an MIP are also feasible to a relaxation, information about the solution space of a relaxation is also valid for the original MIP, and can be used to improve the solution process. For example, *valid inequalities*[2] generated from a relaxation are also valid for an MIP. Moreover, the optimal value of the objective function of a relaxation provides a dual bound on the value of the objective function of an MIP (see Section 3.2).

The most prevalent technique to solve MIPs is the branch-and-bound algorithm which recursively subdivides the problem into smaller subproblems (see Section 3.2). For each of these subproblems the linear programming relaxation (see Section 2.2) is solved in order to obtain dual bounds. Disjunctive relaxations are used to derive valid inequalities (see Section 2.3).

[1]The set of feasible solutions of a relaxation is required properly to contain the set of feasible solutions to the original optimization problem. Moreover, the objective value of a relaxation has to be no worse than the value of the original objective function for all feasible solutions to the original problem. For a more detailed discussion we refer the reader to Wolsey [175].

[2]Given a set X an inequality is referred to as a valid inequality if it is satisfied by all $x \in X$. With respect to a solution $x^* \notin X$ a valid inequality for X is called a cutting plane (or cut) if it is not satisfied by x^*. More formal definitions for MIPs are given in Chapter 3.

2.2. Linear Programming Relaxation

A *linear program* is an optimization problem in which the objective function and the constraints are linear and all variables are continuous. Dantzig [66] proposed the simplex algorithm to solve linear programs. Later, Khachiyan [118] presented the *ellipsoid algorithm* and showed that linear programs are solvable in polynomial time.

A linear program that is closely related to an MIP is its *linear programming (LP) relaxation.* The LP relaxation of (2.1) is obtained by omitting the integrality conditions on the integer-constrained variables

$$\text{(LP)} \quad \min\left\{cx : Ax \geq b, x \geq 0\right\}. \tag{2.4}$$

The *feasible region* of the LP relaxation is given by

$$X_{LP} = \{x \in \mathbb{R}^n : Ax \geq b, x \geq 0\}. \tag{2.5}$$

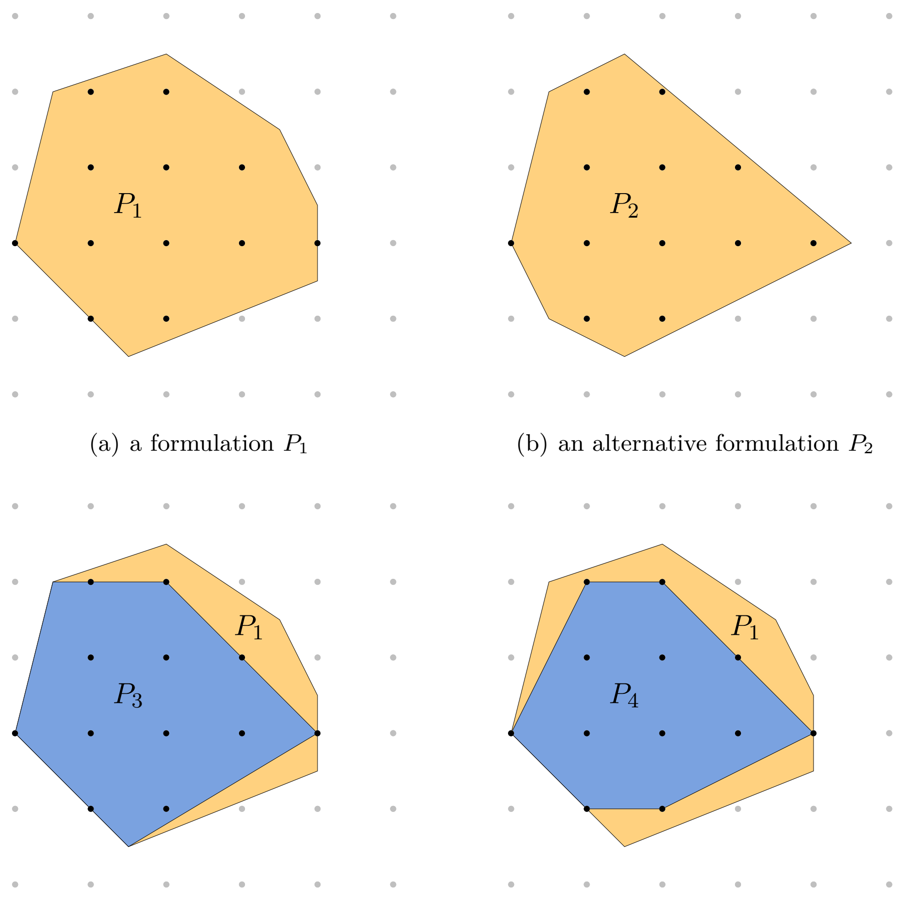
Let c^* be the optimal (minimal) objective value of (2.4). A feasible solution $x^* \in X_{LP}$ is called *LP-optimal* if $cx^* = c^*$. The set (2.5) is defined by $m + n$ inequalities which in turn define $m + n$ *half-spaces.* The set $P = X_{LP}$ is the *polyhedron* which lies at the intersection of these half-spaces. We can write the set X_{MIP} of feasible solutions to (2.1) in the form

$$X_{MIP} = P \cap \left(\mathbb{Z}^{N_I} \times \mathbb{R}^{N \setminus N_I}\right). \tag{2.6}$$

We have $P_I = \text{conv}(X_{MIP}) \subseteq P$, i.e. the polyhedron P contains the polyhedron P_I which is the *integer hull* or the *convex hull of feasible solutions* to (2.1). Concerning the MIP in standard equality form (2.3), X_{LP} and P will denote the feasible domain of the LP relaxation and the associated polyhedron.

2.2.1. Tight LP Relaxation

The set X_{MIP} of feasible solutions to the MIP (2.1) or (2.3) is contained in the polyhedron P which is the feasible domain of the associated LP relaxation. Typically there exist a large number of different polyhedra that all contain exactly the solutions from X_{MIP} but no additional feasible (integral) solutions. Each



(a) a formulation P_1

(b) an alternative formulation P_2

(c) a stronger formulation P_3

(d) the ideal formulation P_4 (convex hull)

Figure 2.1. Formulations

of these polyhedra is called a *formulation* for an MIP. For instance, Figure 2.1 presents different formulations for the same integer program.

A standard technique for solving MIPs is to use an enumerative approach which solves a series of LP relaxations. Concerning this solution approach, not all formulations for an MIP are of the same quality. By linear programming theory, we know that an optimal solution to an LP in standard form is an extreme point of the solution space. Thus all fractional extreme points of a formulation P for an MIP are potential optimal solutions for the LP relaxation (2.4). However, these solutions are not feasible to the MIP. Ideally, the formulation for an MIP is the integer hull $P_I = \operatorname{conv}(X_{MIP})$, i.e. a formulation P having only feasible (integral) extreme points. The strength of an LP relaxation depends heavily on how closely P_I is approximated. A formulation P that well approximates P_I has a *tight LP relaxation* which gives strong dual bounds (see Section 3.2).

As the convex hull P_I is the ideal formulation, we have

$$X_{MIP} \subseteq P_I \subseteq P \tag{2.7}$$

for all formulations P. Solving an LP over P_I is equivalent to solving the MIP (2.1) or (2.3). More precisely, instead of solving the MIP $\min\{cx : x \in X_{MIP}\}$ we can solve the LP

$$\min\{cx : x \in P_I\}. \tag{2.8}$$

This, however, is primarily a theoretical result. Finding a complete description of the convex hull of general $\mathcal{NP}$-hard MIPs is a hard if not impossible task. It is not only difficult to characterize which inequalities describe the convex hull; their number is often exponential in the size of the problem. We therefore concentrate on approximating the convex hull of such problems rather than computing it completely. More precisely, we seek to find a formulation P' such that the optimal extreme point of the LP $\min\{cx : x \in P'\}$ is integer. This solution is then also an optimal solution to the associated MIP. There are two basic ways of obtaining formulations better approximating the convex hull. The first approach creates extended formulations by adding additional variables. The second approach adds additional constraints to the formulation. These constraints are called valid inequalities.

2.2.2. Bases of the LP Relaxation and the Simplex Tableau

The polyhedron P associated with the LP relaxation of (2.1) or (2.3) respectively is defined by $m + n$ constraints, including m constraints from the system $Ax \geq b$ or $Ax = b$ respectively, and n lower bound constraints from $Ix \geq 0$. Each extreme point (vertex) x^* of P corresponds to at least one *basis*. Concerning the LP relaxation of (2.1), a basis B is a set of n constraints such that the submatrix which consists of these constraints is non-singular. The solution x^* that satisfies all n constraints at equality is called the *basic solution*. Geometrically, x^* is a vertex that lies at the intersection point of n hyperplanes. With respect to the LP relaxation of (2.3), a basis B is a set of m linearly independent columns of A. The polyhedron P is said to be *non-degenerate*, if there is a one-to-one correspondence between bases and basic solutions (vertices).

Assume that an MIP in the form (2.3) is given. Let B be a basis of the LP relaxation, i.e. a set of m linearly independent columns of A, and denote the associated basic solution by x^*. Moreover, let $J = N \setminus B$ index the non-basic variables, i.e. the remaining columns of A. With the partition $N = (B, J)$ of the variables, we have $A = (A_B, A_J)$ and $x = (x_B, x_J)$. The matrix A_B is called the *basis matrix.* We can write

$$Ax = b, \tag{2.9a}$$

$$A_B x_B + A_J x_J = b, \tag{2.9b}$$

$$x_B = A_B^{-1} b - \left(A_B^{-1} A_J\right) x_J. \tag{2.9c}$$

The vectorial Equation (2.9c) is known as the *simplex tableau.* Each row of the simplex tableau is associated with a specific basic variable. Let $e_i A_B^{-1}$ be the i^{th} row of the basis inverse where e_i is the i^{th} unit vector. For simplicity, we assume this row of the basis inverse to be associated with the basic variable x_i. Also let $(A_J)_j$ be the j^{th} column of A_J. We obtain

$$x_i = x_i^* - \sum_{j \in J} \bar{a}_{ij} x_j, \quad i \in B, \tag{2.10}$$

where $x_i^* = (e_i A_B^{-1}) b$ and $\bar{a}_{ij} = (e_i A_B^{-1})(A_J)_j$ with $i \in B$ and $j \in J$. We note, however, that in practice the relation between basic variables and rows of the basis inverse is dependent on the order of the columns in the basis matrix A_B. We will sometimes substitute $\bar{a}_{i0}$ for x_i^* in the tableau row (2.10). To simplify the notation the basic and non-basic integer constrained variables will be denoted by $B_I = B \cap N_I$ and $J_I = J \cap N_I$ respectively.

2.3. Disjunctive Relaxation

Disjunctive programming [19] can be seen as an extension of integer programming. More precisely, disjunctive programs are allowed to contain disjunctions of linear constraints. This is a highly intuitive modeling approach since it captures the idea of choosing from a number of alternatives. For example, any combinatorial optimization problem can be modeled as a disjunctive program where each of the disjunctions is one of the feasible solutions. In general, integer programs and other non-convex optimization problems can also be formulated as disjunctive programs, i.e. statements about linear inequalities connected by the logical operator "or".

A *disjunctive program* is of the form

$$\text{(DP)} \quad \min \left\{ cx : \bigvee_{i \in Q} \begin{pmatrix} A^i x \geq b^i \\ x \geq 0 \end{pmatrix} \right\}, \tag{2.11}$$

where $c \in \mathbb{R}^n$, and $b^i \in \mathbb{R}^{m_i}$, $A^i \in \mathbb{R}^{m_i \times n}$, $\forall i \in Q$. The feasible solutions of this disjunctive program are given by the disjunctive set

$$X_{DP} = \left\{ x \in \mathbb{R}^n : \bigvee_{i \in Q} \begin{pmatrix} A^i x \geq b^i \\ x \geq 0 \end{pmatrix} \right\}. \tag{2.12}$$

Here we assume that (2.11) is a *disjunctive relaxation* of an MIP in the form (2.1). More precisely, we assume that

$$A^i = \begin{pmatrix} A \\ D^i \end{pmatrix} \quad \text{and} \quad b^i = \begin{pmatrix} b \\ d^i \end{pmatrix}, \quad \forall i \in Q, \tag{2.13}$$

and that $X_{MIP} = X_{DP} \cap (\mathbb{Z}^{N_I} \times \mathbb{R}^{N \setminus N_I})$, i.e. the disjunctive set contains all of the feasible solutions to (2.1). In addition, let

$$P^i = \left\{ x \in \mathbb{R}^n : A^i x \geq b^i, x \geq 0 \right\}, \quad \forall i \in Q. \tag{2.14}$$

Each of the sets P^i is a polyhedron. We can restate the disjunctive set (2.12) as

$$X_{DP} = \bigcup_{i \in Q} P^i. \tag{2.15}$$

This shows that disjunctive programming is in fact the optimization over the union of (convex) polyhedra. The union of these polyhedra is not, however, necessarily convex.

2.3.1. Disjunctive Inequalities

A disjunctive inequality is a valid inequality which is derived from a disjunction. The idea is to find an inequality that is valid for each term of a disjunction, or in other words for each polyhedron P^i. It then follows that this inequality is also valid for the disjunction itself, i.e. for the union X_{DP}.

Proposition 2.1 ([19])**.** *Let $a^i \in \mathbb{R}^n$ and $b^i \in \mathbb{R}$. Suppose the disjunctive set*

$$S = \left\{ x \in \mathbb{R}^n : \bigvee_{i \in Q} \begin{pmatrix} a^i x & \geq & b^i \\ x & \geq & 0 \end{pmatrix} \right\} \tag{2.16}$$

is given. An inequality $\alpha x \geq \beta$ is valid for S if and only if

$$\alpha_j \geq \max_{i \in Q} \left\{ a^i_j \right\}, \quad j = 1, \ldots, n, \tag{2.17a}$$

$$\beta \leq \min_{i \in Q} \left\{ b^i \right\}. \tag{2.17b}$$

Proposition 2.1 plays a central role in deriving valid inequalities for both disjunctive and mixed-integer programs. However, in our case, each polyhedron P^i (see Equation (2.14)) is described by a set of inequalities. In order to be able to derive a disjunctive inequality using Proposition 2.1 the systems (A^i, b^i) need to be subsumed by single inequalities.

To this end linear combinations $(uA^i)x \geq u^i b^i$ with $u^i \geq 0$, $\forall i \in Q$ are considered. These linear combinations are called *surrogates*. For an inequality to be valid for a set it has to be dominated by a surrogate of the corresponding constraint system. One can think of a surrogate as a reformulation in the sense that it brings out information about feasible solutions that is implicitly contained in the original problem description. Surrogates are particularly helpful in detecting inconsistency of a constraint system. By first constructing surrogates and then applying Proposition 2.1, we obtain the following result.

Theorem 2.2 ([19, 21])**.** *Consider the polyhedra*

$$P^i = \left\{x \in \mathbb{R}^n : A^i x \geq b^i, x \geq 0\right\} \neq \emptyset, \quad \forall i \in Q. \tag{2.18}$$

An inequality $\alpha x \geq \beta$ is valid for $X_{DP} = \bigcup_{i \in Q} P^i$ if and only if there exist $u^i \in \mathbb{R}^{m_i}$, $u^i \geq 0$ such that

$$\alpha \geq u^i A^i, \quad \forall i \in Q, \tag{2.19a}$$

$$\beta \leq u^i b^i, \quad \forall i \in Q. \tag{2.19b}$$

To identify multipliers $u^i \geq 0$ such that the surrogates $(u^i A^i)x \geq u^i b^i$ dominate an inequality $\alpha x \geq \beta$, one can construct the following linear program:

$$\begin{aligned}
\max \quad & 0, && && (2.20a)\\
\text{s.t.} \quad & \alpha - u^i A^i \geq 0, && \forall i \in Q, && (2.20b)\\
& -\beta + u^i b^i \geq 0, && \forall i \in Q, && (2.20c)\\
& u^i \geq 0, && \forall i \in Q, && (2.20d)\\
& (\alpha, \beta) \in \mathbb{R}^{n+1}. && && (2.20e)
\end{aligned}$$

Inequalities (2.20b) and (2.20c) ensure that the inequality $\alpha x \geq \beta$ is valid, i.e. that it is dominated by a surrogate of every disjunct. The system (2.20) can be exclusively written in the variables u^i by eliminating the unrestricted variables α and β. Solving the resulting system, one obtains the multipliers u^i. A valid inequality $\alpha x \geq \beta$ is then generated by setting

$$\alpha_j = \max_{i \in Q} \left\{u^i A^i_j\right\}, \tag{2.21}$$

for $j = 1, \ldots, n$ and

$$\beta = \min_{i \in Q} \left\{u^i b^i\right\}, \tag{2.22}$$

where A^i_j is the j^{th} column of A^i.

2.3.2. Generating Deepest Disjunctive Cuts

Suppose a solution $x^* \notin X_{DP}$ is given and we seek to find a disjunctive inequality which is violated by this solution. A solution x^* violates an inequality $\alpha x \geq \beta$ if $\beta - \alpha x^* > 0$. The most violated (*deepest*) disjunctive cut is therefore found by adding an appropriate objective function to the linear program (2.20).

Proposition 2.3 ([19]). *Consider the polyhedra*

$$P^i = \left\{ x \in \mathbb{R}^n : A^i x \geq b^i, x \geq 0 \right\} \neq \emptyset, \quad \forall i \in Q, \tag{2.23}$$

and their union $X_{DP} = \bigcup_{i \in Q} P^i$. Moreover, suppose a solution $x^* \notin X_{DP}$ is given. Then the most-violated (deepest) disjunctive cut $\alpha x \geq \beta$ with respect to x^* is found by solving the so-called cut generating linear program (CGLP)

$$\begin{aligned} \text{(CGLP)} \quad \max \quad & \beta - \alpha x^*, \\ \text{s.t.} \quad \alpha - u^i A^i &\geq 0, \qquad \forall i \in Q, \\ -\beta + u^i b^i &\geq 0, \qquad \forall i \in Q, \\ u^i &\geq 0, \qquad \forall i \in Q, \\ (\alpha, \beta) &\in \mathbb{R}^{n+1}. \end{aligned} \tag{2.24}$$

Apparently, the system (2.24) is a polyhedral cone. Moreover, it is the cone that contains all valid inequalities for (2.12). This observation is important since it allows us to generate solutions for (2.24) (i.e. cutting planes) that are arbitrarily good (in the sense of the objective function). For example, suppose we have identified a feasible solution $(\bar{\alpha}, \bar{\beta}, \{\bar{u}^i\}_{i \in Q})$ to (2.24). This solution generates the valid inequality $\bar{\alpha} x \geq \bar{\beta}$ using the multipliers $\bar{u}^i$, $\forall i \in Q$. Let $\bar{\beta} - \bar{\alpha} x^* = \xi > 0$, i.e. the inequality is actually a cut for an arbitrary solution x^*. Now, consider a second solution $(\lambda\bar{\alpha}, \lambda\bar{\beta}, \{\lambda\bar{u}^i\}_{i \in Q})$ with $\lambda \in \mathbb{R}$, $\lambda > 0$. It is easy to check that this solution is also feasible for the linear program (2.24). Moreover, this solution produces the cut $\lambda\bar{\alpha} x \geq \lambda\bar{\beta}$ which is identical with that generated from the first solution. However, for $\lambda > 1$ this cut is λ times more violated than $\bar{\alpha} x \geq \bar{\beta}$ since $\lambda\bar{\beta} - \lambda\bar{\alpha} x^* = \lambda\xi > \xi > 0$. It is therefore possible to scale a solution (or cut) such that the objective function has the value infinity. This is one of the major drawbacks of maximizing the violation over a polyhedral cone. There are two approaches to overcoming issues related to scaling. Firstly, we could try to find a better objective function for the linear program (2.24). Secondly, we could truncate the polyhedral cone by a bounding constraint which prevents the solutions from scaling up indefinitely.

The ideal objective function for (2.24) would be to maximize the distance between the solution x^* and its orthogonal projection on the hyperplane $\alpha x = \beta$. This distance is given by $\frac{\beta - \alpha x^*}{\|\alpha\|}$, where $\|\cdot\|$ is the Euclidean norm. Obviously this objective function is not affected by scaling. On the other hand, it is non-linear ruling out standard LP methods to solve the resulting cut generating non-linear program. Accordingly, the objective function is kept unaltered and bounding constraints are considered.

Balas et al. [24, 25] discuss different bounding constraints for the system (2.24) which they call *normalizations.* These constraints are

$$|\beta| \le 1, \tag{2.25}$$

$$|\alpha_j| \le 1, \quad j = 1, \dots, n, \tag{2.26}$$

$$\sum_{j=1}^{n} |\alpha_j| \le 1, \tag{2.27}$$

and are imposed directly on the coefficient vector α and the right-hand side β of the disjunctive cut. Each of these normalizations has certain drawbacks. While introducing the constraints (2.25) or (2.26) into the linear program (2.24) is easy, normalization (2.25) does not guarantee that α is bounded in general and normalization (2.26) is computationally less attractive (with respect to the quality of the generated cuts). Similarly, while (2.27) performs well in practice, it has to be linearized. For a detailed study see [24, 25].

As an alternative, Balas [20] proposes the normalization

$$\sum_{i \in Q} \left(\sum_{l=1}^{m_i} u_l^i \right) \le k, \tag{2.28}$$

where $k > 0$. This normalization is considered by Ceria and Pataki [47] for the case $k = 1$. Instead of directly restricting the coefficients of the cut and its right-hand side, normalization (2.28) imposes a restriction on the non-negative multipliers

u^i, $\forall i \in Q$. As α and β are determined by a linear combination involving these multipliers (see Equations (2.21) and (2.22)), the normalization (2.28) bounds all variables in (2.24). It follows that the objective function is bounded as well.

A special case of the normalization (2.28) is given by

$$\sum_{i\in Q}\left(\sum_{l=1}^{m_i} u_l^i\right) = 1. \tag{2.29}$$

The latter normalization was studied by Ceria and Soares [48] and later by Balas and Perregaard [29, 30]. Normalization (2.29) is easily integrated into the linear program (2.24) and tends to produce cuts with nice properties (see [87]). It has thus become the most widely used normalization constraint.

2.3.3. Examples

Example 2.4. *Consider the integer program*

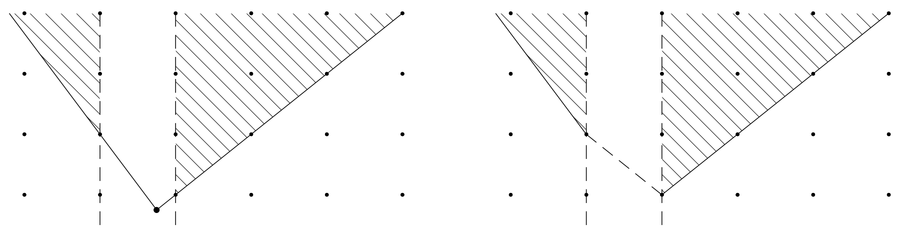
$$\begin{array}{lrcrcl} \min & x_1 & + & x_2, & & \\ \text{s.t.} & -x_1 & + & x_2 & \geq & -1, \\ & 5x_1 & + & 3x_2 & \geq & 11, \\ & x_1, x_2 & \in & \mathbb{Z}^1. & & \end{array} \tag{2.30}$$

Solving the LP relaxation yields the fractional solution $x^* = (1\frac{3}{4}, \frac{3}{4})$. With regard to this solution, we examine the valid disjunction

$$\begin{pmatrix} 5x_1 & + & 3x_2 & \geq & 11 \\ -x_1 & & & \geq & -1 \end{pmatrix} \bigvee \begin{pmatrix} -x_1 & + & x_2 & \geq & -1 \\ x_1 & & & \geq & 2 \end{pmatrix} \tag{2.31}$$

and try to find a valid inequality cutting off x^*. Figure 2.2(a) shows the feasible regions of each of the disjuncts (in the LP relaxation). We associate the multipliers $u, v \in \mathbb{R}^2_+$ with the first and second terms of the disjunction respectively

$$\begin{aligned} &\left((5u_1 - u_2)\, x_1 + 3u_1 x_2 \geq 11u_1 - u_2\right) \\ &\qquad\qquad \vee \left((-v_1 + v_2)\, x_1 + v_1 x_2 \geq -v_1 + 2v_2\right). \end{aligned} \tag{2.32}$$


(a) an integer program and a disjunctive relaxation

(b) a disjunctive inequality cutting off the fractional LP solution

Figure 2.2. An example of a disjunctive cut

Theorem 2.2 states that an inequality $\alpha x \geq \beta$ is valid for (2.30) *if and only if it is dominated by each of the surrogates in* (2.32)*. Setting $u = (1,2)$ and $v = (3,6)$, we obtain the surrogate $3x_1 + 3x_2 \geq 9$ for both disjuncts. Clearly, by Proposition 2.1 this inequality is then also valid for the original integer program* (2.30)*. Moreover, it cuts off the fractional solution x^* as shown in Figure 2.2(b).*

Example 2.5. *Consider the simple mixed-integer set*

$$X = \Big\{(x,y) \in \mathbb{R}^1_+ \times \mathbb{Z}^1 : x + y \geq b\Big\}. \tag{2.33}$$

A disjunctive relaxation of (2.33) *is given by the set $X_{DP} = P^1 \cup P^2$ (see Figure 2.3(a)) where*

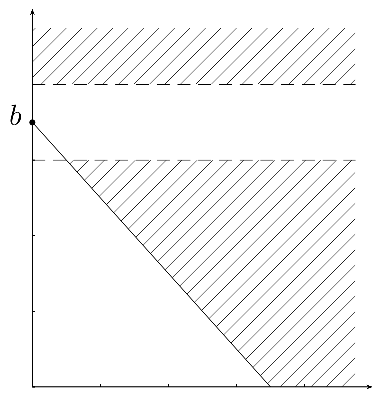
$$P^1 = \Big\{(x,y) \in \mathbb{R}^1_+ \times \mathbb{R}^1 : x + y \geq b, y \leq \lfloor b \rfloor\Big\} \tag{2.34}$$

and

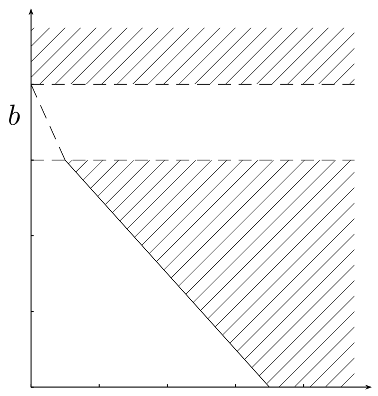
$$P^2 = \Big\{(x,y) \in \mathbb{R}^1_+ \times \mathbb{R}^1 : y \geq \lceil b \rceil\Big\}. \tag{2.35}$$

In what follows, we assume that the right-hand side b is fractional and let $f = b - \lfloor b \rfloor > 0$ denote the fractional part of b. The surrogates of the disjuncts P^1 and P^2 are given by

$$u \cdot \left(\begin{array}{ccccc} x & + & y & \geq & b \\ & & -y & \geq & -\lfloor b \rfloor \end{array} \right) \quad \bigvee \quad v \cdot (y \geq \lceil b \rceil) \tag{2.36}$$



(a) a simple mixed-integer set and a disjunctive relaxation



(b) a disjunctive inequality cutting off the fractional LP solution

Figure 2.3. The simple mixed-integer rounding inequality

where $u \in \mathbb{R}^2_+$ *and* $v \in \mathbb{R}^1_+$*. Let* $u = (\frac{1}{f}, \frac{1}{f} - 1)$ *and* $v = 1$*. We obtain the valid inequality*

$$\frac{x}{f} + y \geq \lceil b \rceil \tag{2.37}$$

for P^1 *and the valid inequality* $y \geq \lceil b \rceil$ *for* P^2*. In fact, by applying Proposition 2.1, we have that Inequality* (2.37) *is also valid for the simple mixed-integer set* (2.33) *(see Figure 2.3(b)). This disjunctive inequality is well known as the simple mixed-integer rounding inequality (see Wolsey [175]).*

Chapter 3.

Algorithms

In the preceding chapter, we introduced mixed-integer programming problems and two important relaxations. We pointed out that any MIP can be solved by finding a complete description of the convex hull of its feasible solutions. We showed, furthermore, how disjunctive relaxations can be used to derive valid inequalities.

In this chapter we deal with algorithms for solving MIPs. After formally defining the terms valid inequality and cutting plane, we discuss cutting plane algorithms which attempt to solve MIPs by applying cutting planes to the LP relaxation. We further consider the branch-and-bound algorithm which tackles MIPs by successively dividing them into smaller subproblems. We highlight the merits and demerits of each of these solution approaches and also discuss how state-of-the art MIP solvers combine the branch-and-bound algorithm with cutting planes.

This chapter is organized as follows. In Section 3.1 we deal with cutting planes and cutting plane algorithms. The branch-and-bound algorithm which employs a divide-conquer-strategy to solve MIPs is introduced in Section 3.2.

3.1. Cutting Planes

In practice, LP relaxations are often weak in the sense that the polyhedron P provides a bad approximation of the convex hull of feasible solutions P_I. Additional constraints (valid inequalities) are introduced to strengthen the LP relaxation by excluding fractional solutions from its feasible domain without removing integral solutions. An inequality $\alpha x \geq \beta$ is *valid* for P_I if $P_I \subseteq \{x \in \mathbb{R}^n : \alpha x \geq \beta\}$.

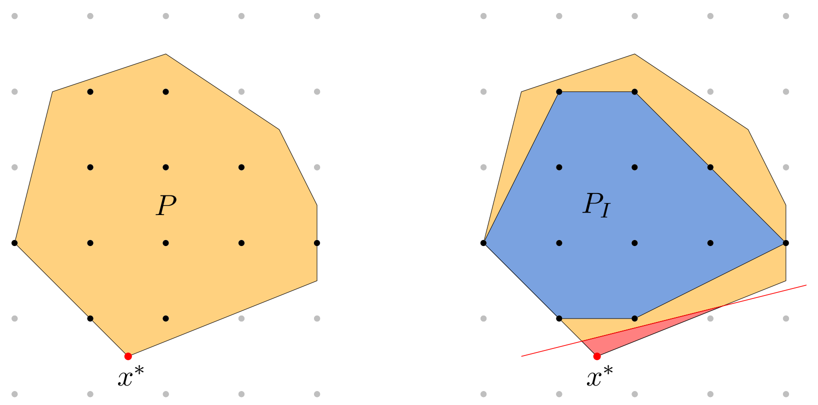


Figure 3.1. A cutting plane separating a fractional LP solution x^*

Moreover, an inequality $\alpha x \geq \beta$ is called a *cutting plane* (or *cut*) for $x^* \notin P_I$ if the solution x^* does not lie in the half-space defined by $\alpha x \geq \beta$, i.e.

$$x^* \notin \{x \in \mathbb{R}^n : \alpha x \geq \beta\}. \tag{3.1}$$

Consequently, the cut *separates* the solution x^* from the convex hull P_I (see Figure 3.1).

To find a violated cutting plane, one can solve the *separation problem.* This problem is defined as follows.

> Given a feasible solution x^* to the LP relaxation (2.4) which is not feasible to the MIP (2.1), find a valid inequality $\alpha x \geq \beta$ for the MIP (2.1) that is violated by (or, in other words, cuts off) x^*, i.e. $\alpha x^* < \beta$.

The separation problem for a class of cutting planes is often modeled as a (non-) linear optimization problem. An algorithm which is designed to solve the separation problem is referred to as a *separation algorithm.* There are two kinds of separation algorithms, *exact* and *heuristic.* An exact separation algorithm solves the separation problem and is therefore guaranteed to find a violated cutting plane, if one exists. Exact separation is not always cost-efficient, especially when the separation problem is large and difficult to solve. Separation heuristics perform a heuristic search for violated cutting planes, e.g. by examining a sufficiently small relaxation of the separation problem. While these heuristics are fast, they can fail to find a violated cut even if there is one.

Any MIP can be solved by finding a description of the convex hull of feasible solutions P_I. A *cutting plane algorithm* is that which approximates P_I by iteratively solving the LP relaxation and the separation problem (see Algorithm 3.1). If the

Algorithm 3.1. Cutting plane algorithm

```
        Input: MIP (2.1).
        Output: An optimal solution $\bar{x}$ to the MIP (2.1) with objective
                value $\bar{c}$, or model is infeasible indicated by $\bar{c} = \infty$.
(Step 1) Solve the LP relaxation
        Solve the LP relaxation (2.4) to optimality.
        (Check for infeasibility.) If $P = \emptyset$ (the LP relaxation is infeasible)
        set $\bar{c} := \infty$ and exit.
        Otherwise, let $x^*$ be an optimal solution to the LP relaxation with
        objective value $c^*$.
        (Check for optimality.) If $x^*$ is feasible for the MIP (2.1) set
        $\bar{x} := x^*$ and $\bar{c} := c^*$, and exit.
(Step 2) Solve the separation problem
        Solve the separation problem to obtain a cut $\alpha x \geq \beta$ for $x^*$.
        Add this cut to the formulation of the MIP.
        Goto Step 1.
```

solution to the LP relaxation is not feasible to the MIP, the algorithm separates this solution from P_I by solving the separation problem. This process is iterated until the optimal solution of the LP relaxation is feasible to the MIP or the LP relaxation is infeasible. Gomory [97, 99] proposed a cutting plane algorithm which converges finitely for integer programs in rational data. In practice, however, this algorithm usually needs an exponential number of cutting planes and iterations to achieve convergence. Thus the size of the problem formulation is increased dramatically, which complicates the solution of the LP relaxation. In addition, the cuts Gomory used in his algorithm are vulnerable to numerical inaccuracies which influence the numerical stability of the LP solver.[1] Gomory [98] also introduced a cutting plane algorithm for MIPs which does not have finite convergence

[1]However, Zanette et al. [177] have recently shown that Gomory's cutting plane algorithm [97, 99] can be effective when used in conjunction with the lexicographic dual simplex method.

properties in general. In the hope of achieving faster convergence, Algorithm 3.1 can be adapted to add more than one violated cut at the same time.

3.2. Branch-and-Bound

A method for solving general optimization problems is the *branch-and-bound algorithm*. The branch-and-bound algorithm is a *divide-and-conquer* solution technique which uses an *implicit enumeration* to explore the solution space. Given a problem instance, this means that the solution space is recursively divided into smaller subproblems. The hope is that these subproblems become easier to solve as the algorithm proceeds (and the subproblems become smaller and smaller). Land and Doig [127] were the first to present a branch-and-bound algorithm for integer programming.

The flow of the branch-and-bound algorithm is shown in Algorithm 3.2. The action of dividing a (sub-) problem into smaller subproblems is called *branching*. In so doing the algorithm creates a so-called *branching tree*. An example of such a tree is depicted in Figure 3.2. In Step 1 of Algorithm 3.2 the original MIP is added to the empty list $\mathcal{L}$ of unprocessed problems (nodes). This problem represents the root of the branching tree (see Figure 3.2). On the other hand, the leaves of the branching tree are either unprocessed (unsolved) subproblems or subproblems which were solved and taken out of consideration.

The process of removing parts of the branching tree which can be guaranteed only to contain inferior solutions is called *bounding* or *pruning*. Specifically, the bounding in Step 5 of Algorithm 3.2 prevents the algorithm from performing a complete enumeration of the solution space. A node can be pruned by *infeasibility*, *bound* or *optimality*. The key ingredients in the bounding step are strong dual (lower) bounds and primal (upper) bounds. Dual bounds are obtained by solving the LP relaxation S_R for each node S. Primal bounds are either found when a node is pruned by optimality or by primal heuristics. The effectiveness of bounding is also heavily dependent on the node selection and branching strategy. It is important to find a good primal bound and cut off parts of the search tree as early as possible.

Algorithm 3.2 is also called *LP-based* as it solves LP relaxations to obtain dual bounds. In general, an arbitrary relaxation can be used to this end. Such

Algorithm 3.2. Branch-and-bound algorithm

```
        Input: MIP (2.1) denoted by $R$.
        Output: An optimal solution $\bar{x}$ to the MIP (2.1) with objective
                value $\bar{c}$, or model is infeasible indicated by $\bar{c} = \infty$.
(Step 1) Initialize
        Add original problem to the list of unprocessed nodes $\mathcal{L} := \{R\}$,
        and set global upper bound $\hat{c} := \infty$.
(Step 2) Check termination criteria
        If $\mathcal{L} = \emptyset$, set $\bar{x} := \hat{x}$ and $\bar{c} := \hat{c}$, and exit.
(Step 3) Select node
        Choose node $S \in \mathcal{L}$, and update list of unprocessed nodes
        $\mathcal{L} := \mathcal{L} \setminus \{S\}$.
(Step 4) Calculate dual bound
        Solve the LP relaxation $S_R$ of $S$.
(Step 5) Bound
        (Prune by infeasibility.) If $S_R$ is infeasible, goto Step 2.
        Otherwise, let $x^*$ be an optimal solution to $S_R$ with $c^*$ being the
        objective value.
        (Prune by bound.) If $c^* \geq \hat{c}$, goto Step 2.
        (Prune by optimality.) If $x^*$ is feasible for $R$, set $\hat{x} := x^*$, $\hat{c} := c^*$,
        and goto Step 2.
(Step 6) Branch
        Create two subproblems $S = S_1 \cup S_2$, set $\mathcal{L} := \mathcal{L} \cup \{S_1, S_2\}$, and
        goto Step 2.
```

a relaxation is required to be relatively easy to solve and to yield strong dual bounds. Note that these requirements do not usually coincide. The optimal solution to the LP relaxation is also used to decide on which variable to branch next. Consider an integer variable x_j, $j \in N_I$, and let $x_j^* \notin \mathbb{Z}$ be the value of this variable in the optimal solution to the LP relaxation. We construct two subproblems $S_1 = S \cap \{x \in \mathbb{R}^n : x_j \leq \lfloor x_j^* \rfloor\}$ and $S_2 = S \cap \{x \in \mathbb{R}^n : x_j \geq \lceil x_j^* \rceil\}$, i.e. we branch on the disjunction

$$\left(x_j \leq \left\lfloor x_j^* \right\rfloor\right) \vee \left(x_j \geq \left\lceil x_j^* \right\rceil\right). \tag{3.2}$$

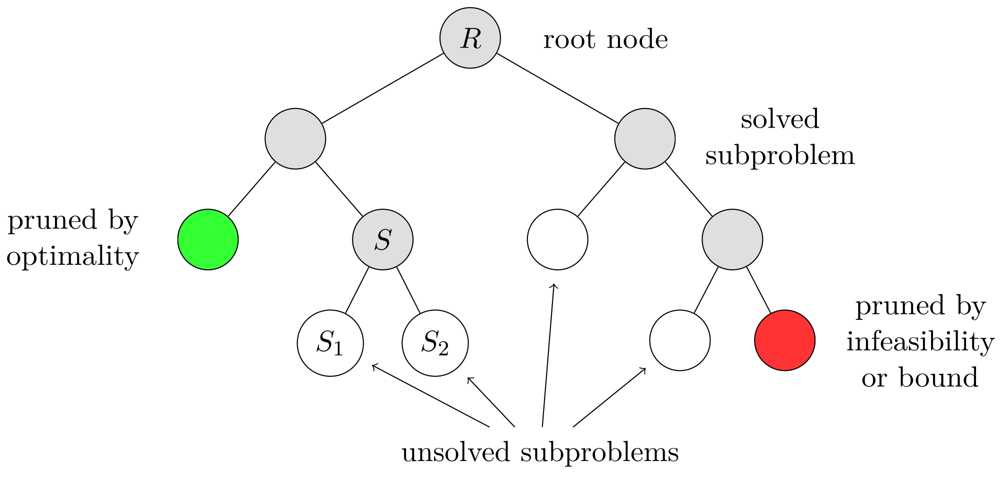


Figure 3.2. Branch-and-bound tree

Branching on such two-term disjunctions involving only single variables is called *single-variable branching*. Linderoth and Savelsbergh [132] and Achterberg et al. [5] give excellent overviews of single-variable branching rules and study their computational usefulness. A different approach is to branch on constraints or *general disjunctions* of the form

$$(\pi x \leq \pi_0) \vee (\pi x \geq \pi_0 + 1)\,. \tag{3.3}$$

Different strategies for branching on general disjunctions and computational results are discussed by Karamanov and Cornuéjols [117], Cornuéjols et al. [62] and Mahajan and Ralphs [134]. It is also possible to create more than two subproblems during branching, i.e. to branch on multiple-term disjunctions.

State-of-the-art MIP solvers combine the branch-and-bound method with cutting plane techniques to benefit from a stronger LP relaxation during bounding. The first approach, called *cut-and-branch*, strengthens the LP relaxation at the root node by a reasonable number of cutting planes, and then solves the problem by branch-and-bound. An alternative strategy is to strengthen the LP relaxation by cutting planes at further selected nodes of the branching tree. This approach is called *branch-and-cut*. Some complications arise since one needs to distinguish between locally and globally valid cutting planes. The latter can be applied throughout the entire search tree. By contrast, locally valid cutting planes use information about branching decisions and are thus only valid in the current subtree. Consequently these cuts must be removed from the LP relaxation once the algorithm leaves this part of the branching tree.

Part II.

State-of-the-Art

Chapter 4.

Single-Row Cutting Planes

In Part I of this thesis, we presented an introduction to mixed-integer programming and described solution approaches for mixed-integer programs. We focussed particularly on valid inequalities and cutting planes. We showed how valid inequalities can be derived from disjunctive arguments and discussed how the LP relaxation of MIPs can be strengthened by using valid inequalities as cutting planes.

In this chapter we are concerned with general-purpose cutting planes derived from single-row relaxations of MIPs. We review the state-of-the-art in cutting plane methods and give a unified presentation of the different techniques. In particular we concentrate on the Gomory mixed-integer cuts and discuss several approaches which seek to improve their performance.

This chapter is structured as follows. Section 4.1 explains why single-row relaxations are studied and defines the term "single-row inequality". In Section 4.2 we review the relevant literature. Section 4.3 treats Chvátal-Gomory cuts which are derived from single-row relaxations of pure integer programs. Section 4.4 is devoted to cutting planes for general MIPs and discusses different classes of split cuts.

4.1. Introduction

As the name suggests, single-row relaxations of MIPs only consist of single linear constraints. Such optimization problems are rarely encountered in practice. In fact, MIPs arising from practical applications tend to become more and more complex in the number of variables and constraints involved. On the other hand, single-row relaxations are very interesting from a theoretical point of view. In comparison with general MIPs, these relaxations are very simple, which makes them easier to analyze. Let us reconsider the MIP (2.1). If we aggregate the constraints of (2.1) using the weights $u \in \mathbb{R}^m_+$, we obtain the single-row MIP

$$\min \{cx : (uA)\, x \geq ub, x \geq 0, x_j \in \mathbb{Z}, \forall j \in N_I\} \tag{4.1}$$

which is a relaxation of (2.1). We refer to an inequality derived from a relaxation of the form (4.1) as a *single-row inequality* or *single-row cut* respectively. In other words, any inequality that is derived from a single linear constraint including lower (and upper) bounds and integrality requirements on the variables is a single-row inequality.

Any solution to (2.1) is also a solution to (4.1). Moreover, any inequality that is valid for (4.1) is also valid for (2.1). The latter fact is one of the main reasons why single-row relaxations are studied. More precisely, the hope is that strong inequalities for (4.1), which ideally would be facet-defining[1], will turn out to be strong inequalities for (2.1) and that these inequalities then enable more efficient solution of complex MIPs. The generation of valid inequalities (cutting planes) which are based on single-row relaxations is a central component of today's state-of-the-art MIP solvers. For example, Bixby and Rothberg [38] report on notable performance degradations obtained by switching off the generation of these cutting planes in Cplex [115].

Valid inequalities from single-row relaxations have been studied intensely during the last decades. In fact most valid inequalities in (mixed-) integer programming are based on a particular single-row relaxation. In the following section we give a brief review of the most important publications on this subject. For an excellent survey on single-row cutting planes we refer the reader to Cornuéjols [59].

4.2. Literature Review

In 1958 Gomory [97] published a seminal paper in which he proposed an algorithm to solve pure integer programs using a class of cutting planes which are derived

[1]A facet is a face of dimension one less than the dimension of the associated polyhedron. For a detailed discussion we refer the reader to Wolsey [175].

from the simplex tableau. This algorithm is today known as *Gomory's fractional cutting plane algorithm.* Gomory also stated (without proof) that his algorithm obtains the integer optimum in a finite number of steps. Later Gomory [99] analyzed his fractional cutting plane algorithm in more detail and proved finiteness (for integer programs in rational data). As no finite algorithm for solving integer programs was known to that day, Gomory's initial paper represents a revolution in the field of integer programming. Thus the year 1958 is considered to mark the birth of integer programming. In [98] Gomory extended his work to mixed-integer programs and proposed the *Gomory mixed-integer cut.* He also showed that a pure cutting plane algorithm which is based on these cuts (*Gomory's mixed-integer cutting plane algorithm*) converges finitely if the objective function is integer-valued.

Although the Gomory fractional and Gomory mixed-integer cuts have very nice theoretical properties, these were, for several reasons, considered useless in practice for more than thirty years. First of all, pure cutting plane algorithms turned out in practice to converge very slowly. This means that a large number of cutting planes (iterations) was necessary to achieve convergence. Moreover, working on the simplex tableau, Gomory cuts are highly prone to numerical inaccuracies.[2] In the 1990s, Balas et al. [26] reevaluated Gomory cuts in the course of their work on a special class of disjunctive cuts, namely lift-and-project cuts. Surprisingly, in their experiments, Gomory cuts proved to be effective. In contrast to previous approaches, Balas et al. embedded Gomory cuts into a branch-and-cut framework and added several cuts at a time (in rounds) before re-optimizing the LP relaxation. Moreover, enhancements in the numerical stability of LP solvers made Gomory cuts more reliable. Cornuéjols [58] gives an excellent overview of the history of Gomory cuts and the development that led to their rediscovery. Since then Gomory cuts have been studied intensively, resulting in several improvements and variations [8, 30, 46, 61, 129]. The most recent approaches are due to Fischetti and Salvagnin [88] who propose a relax-and-cut framework for Gomory mixed-integer cuts and Dash and Goycoolea [69] who discuss several heuristics for obtaining strong Gomory mixed-integer cuts from alternative bases of the LP relaxation.

[2]However, we note again that Zanette et al. [177] recently reported on an effective implementation of Gomory's algorithm [97].

Balas et al. [24] propose *lift-and-project cuts* which are a subclass of the disjunctive cuts. These cuts are derived from a simple disjunction of the form $x \leq 0$ or $x \geq 1$ on a 0-1 (binary) variable. The most-violated lift-and-project cut can be found by solving a cut generating linear program (cf. Proposition 2.3). Computational experience with lift-and-project cuts in a branch-and-cut framework is reported in [25] and additional enhancements of the method are documented in [20, 29]. Although lift-and-project cuts proved to be effective, setting up and solving the cut generating linear program involves a large amount of additional computational work which is not always cost-efficient. This changed when Balas and Perregaard [30] presented an elegant algorithm which mimics the optimization of the cut generating linear program by performing pivots on the original linear programming tableau. Different variants of this algorithm are discussed and evaluated computationally in [22, 23].

An additional class of cutting planes which are derived from a single-row relaxation are the *knapsack cover cuts* [18, 111, 172]. These cuts are based on the observation that certain variables (which form a so-called *cover*) are not allowed all simultaneously to have a non-zero value in a feasible integral solution. The process of lifting additional variables[3] into cover inequalities is treated in [144, 145, 173]. Crowder et al. [65] discuss an algorithm for solving 0-1 IPs which includes an a priori generation of cover cuts. Nowadays, all successful commercial software packages for solving MIPs contain separation routines for cover cuts. There are a large number of further publications related to cover inequalities. For instance, Weismantel [168] proposes the larger class of *weight inequalities* which includes cover inequalities. Other papers study extensions of cover inequalities which are able to handle additional side constraints [107, 174] or general integer knapsacks [45]. The complexity of the separation problem for cover cuts is examined in [108, 120]. On overview of knapsack cuts is given by Atamtürk [15].

Clique inequalities [144] are valid for set packing polytope and are derived from the so-called intersection graph. Each node in this graph represents a column

[3]In the context of cover inequalities, lifting means that coefficients for the variables which are not in the cover are computed. In general, the process of deriving a valid inequality for a set from an inequality which is valid for a lower-dimensional restriction of the set is called lifting. For instance, lifting is discussed thoroughly by Nemhauser and Wolsey [143].

in the set packing problem. Two nodes are connected by an edge whenever the corresponding columns appear together in a set packing constraint at least once. A clique in the intersection graph then translates into a set of variables from which only one can be selected in a feasible solution to the set packing problem. This gives rise to a family of valid inequalities, namely the above-mentioned clique inequalities. Hoffman and Padberg [113] use clique inequalities in a branch-and-cut framework to solve set partitioning problems arising from airline crew scheduling. Moreover, they also discuss the lifting of clique inequalities.

Another well-studied single-row mixed-integer set is the (binary) single-node flow set. The *flow cover inequalities* developed by Padberg et al. [147] are valid for this set. In [165] van Roy and Wolsey study a variant of the single-node flow set and define two families of valid inequalities. The separation algorithm for flow cover inequalities, its implementation and computational results are presented in [166]. Gu et al. [109] suggest an efficient approach to lift flow cover inequalities and report on extensive computational experiments.

Mixed-integer rounding (MIR) inequalities appeared in [142] and are valid for mixed-integer knapsack sets (see also [143, 174]). Whether or not MIR cuts are useful from a computational point of view was for quite a while an open question. Marchand and Wolsey [135] proposed a sophisticated separation heuristic for MIR cuts that features constraint aggregation and bound substitution. Their results showed that MIR cuts can significantly reduce the integrality gap on many instances from MIPLIB 3.0 [37]. Moreover, Marchand and Wolsey pointed out that other families of strong valid inequalities (weight, residual capacity, mixed cover and flow cover inequalities) are actually MIR inequalities. Given these positive results, separation routines for MIR cuts have been integrated into commercial MIP solvers where they are among the most effective cutting planes [38]. Recently, several extensions to MIR cuts have been proposed [16, 70, 71, 119]. Furthermore, the exact separation of MIR inequalities is investigated in [73].

Chvátal-Gomory (CG) inequalities [53] are valid inequalities for pure integer programs. These inequalities are derived by weighting and summing up a set of inequalities, followed by a rounding. If the weights (multipliers) are restricted to either 0 or $\frac{1}{2}$ we obtain the $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts of Caprara and Fischetti [42]. In particular, $\{0, \frac{1}{2}\}$-cuts subsume some classes of problem-specific valid inequalities for the matching polytope and the stable set polytope among others. The more general class of mod-k cuts is examined by Caprara et al. [43] who particularly concentrate on the complexity of the separation problem. Let us note that the mod-2 cuts are precisely the $\{0, \frac{1}{2}\}$-cuts. A separation algorithm for $\{0, \frac{1}{2}\}$-cuts and its implementation is discussed in [13, 126]. In addition, there are several other contributions concerning CG inequalities. For instance, Letchford and Lodi [129] propose a strengthened variant of the CG cuts and report on preliminary computational tests. Fischetti and Lodi [86] address the question of how useful CG cuts are in practice by solving the separation problem (an MIP) in a pure cutting plane framework. Their results show that CG cuts are very effective and succeed in closing large percentages of the integrality gaps of the pure IPs in MIPLIB 3.0 [37]. Another direction of research is to apply CG cuts to mixed-integer programs. Bonami et al. [40] extend the separation problem for CG cuts to handle continuous variables. The results with this approach are promising, especially when the continuous variables only play a minor role in the structure of the problem.

Typically, to compare different families of cutting planes their *elementary closures* are analyzed both theoretically and computationally. Given a family of cutting planes $\mathcal{F}$ and a polyhedron P, the elementary closure of P with respect to $\mathcal{F}$ is the set of all points in P satisfying the cutting planes in the family $\mathcal{F}$. In particular, Nemhauser and Wolsey [142] proved that Gomory mixed-integer cuts, split cuts and MIR cuts are equivalent, i.e. their elementary closures are identical. For an overview of the relation between various classes of cutting planes we refer the reader to Cornuéjols and Li [60]. Another important notion is the *rank* of an inequality which was introduced by Chvátal [53] for the special case of CG inequalities. Loosely speaking, the rank of an inequality from the original formulation of a polyhedron P is 0 whereas any inequality obtained from a combination of rank-0 inequalities for P has rank 1 and so on.[4] The concepts of the rank of inequalities and the closures of a polyhedron P are closely related. The elementary closure is precisely the intersection of all rank-1 cutting planes. As mentioned above, several authors investigate computationally the strength of the elementary closure of specific classes of cutting planes. Dash et al. [73]

[4]For a precise definition of the rank of an inequality we refer the reader to Chvátal [53].

study the MIR closure (see also [39]) while Balas and Saxena [31] analyze the strength of the split closure. The elementary closure of CG and projected CG cuts is studied by Fischetti and Lodi [86] and Bonami et al. [40] respectively.

4.3. Chvátal-Gomory Cuts

In this section we consider Chvátal-Gomory (CG) cuts for pure integer programs which were introduced by Chvátal [53]. This section comprises two subsections. Section 4.3.1 is devoted to a subclass of the CG cuts known as $\{0, \frac{1}{2}\}$-cuts [42]. In Section 4.3.2 we elaborate on how to obtain stronger CG cuts. In particular, we consider the strong CG cuts of Letchford and Lodi [129].

Pure integer programs (IPs) may be seen as special cases of MIPs in which all variables are integer-constrained. For convenience, we introduce IPs in a slightly different notation. Consider an IP of the form

$$\text{(IP)} \quad \min\left\{cx : x \in X_{LP}, x \in \mathbb{Z}^n\right\}, \tag{4.2}$$

where $c \in \mathbb{R}^n$ and the feasible set to the LP relaxation is given by

$$X_{LP} = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\} \tag{4.3}$$

with the integral matrix $A \in \mathbb{Z}^{m\times n}$ and the integral vector $b \in \mathbb{Z}^m$. Again, we use $P = X_{LP}$ to denote the polyhedron associated with the LP relaxation. The system defining P can be written as a system of equations $Ax + s = b$ by introducing a vector s of slack variables. As above, let P_I denote the convex hull of the feasible solutions of (4.2).

Proposition 4.1 ([53])**.** *The Chvátal-Gomory cut*

$$\lfloor uA \rfloor x \leq \lfloor ub \rfloor, \tag{4.4}$$

where $u \in \mathbb{R}^m_+$ is valid for the IP (4.2)*.*

It is easy to see that CG cuts match our definition of single-row cuts. The values $u_i \in \mathbb{R}_+$ for $i = 1, \ldots, m$ are called the *CG multipliers.* It can be shown that the vector of CG multipliers u of undominated CG cuts satisfies $u \in [0, 1[^m$.

CG cuts can be applied without knowing the structure of an IP (see Figure 4.1). Let $a = uA$ and $a_0 = ub$. For an arbitrary vector $u \in \mathbb{R}^m_+$ the inequality $ax \leq a_0$ is valid for the polyhedron P. It follows that the inequality $\lfloor a \rfloor x \leq a_0$ is also valid for P. Now suppose that a_0 is non-integral. Then the inequality $\lfloor a \rfloor x \leq \lfloor a_0 \rfloor$ is valid for P_I but not for P. On the other hand, if a_0 is integral the CG inequality will not cut off any part of the polyhedron P.

An important characteristic of CG cuts is that they are sufficient to describe the convex hull P_I of feasible solutions to the pure IP (4.2) [53, 97]. This means that any valid inequality for P_I can be obtained as a CG inequality by applying Proposition 4.1 a finite number of times.

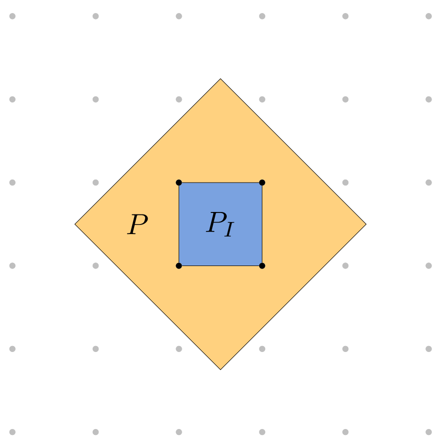
Given an arbitrary solution x^*, the CG separation problem is to find a vector u of CG multipliers such that the resulting CG inequality is violated by x^* (see Fischetti and Lodi [86]). Eisenbrand [82] proved that the separation problem for CG cuts is $\mathcal{NP}$-hard. The CG separation problem can be modeled as a mixed-integer program

$$\begin{aligned}
\max\ & \alpha x^* - \beta, \\
\text{s.t.} \quad & \alpha_j \leq uA_j, && \text{for } j = 1, \dots, n, \\
& \beta \geq ub - 1 + \epsilon, && \\
& u_i \geq 0, && \text{for } i = 1, \dots, m, \\
& (\alpha, \beta) \in \mathbb{Z}^{n+1},
\end{aligned} \tag{4.5}$$

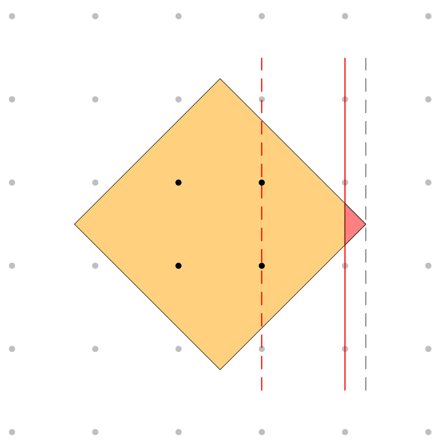
where A_j is the j^{th} column of A and ϵ is a small positive value. The objective function maximizes the violation of the CG cut $\alpha x \leq \beta$ with respect to the solution x^*. The constraint matrix of (4.5) has $n+1$ rows and $n+m+1$ columns. The CG separation problem is thus large and solving it is not always practicable.

We now consider a special case in which finding a separating CG cut is trivial. Let x^* be a non-integral basic solution to the LP relaxation of (4.2), i.e. a non-integral vertex of P. Suppose that x^*_i is non-integral and let $u = e_i A_B^{-1}$ be the i^{th} row of the basis inverse. Applying Proposition 4.1 we obtain the CG cut

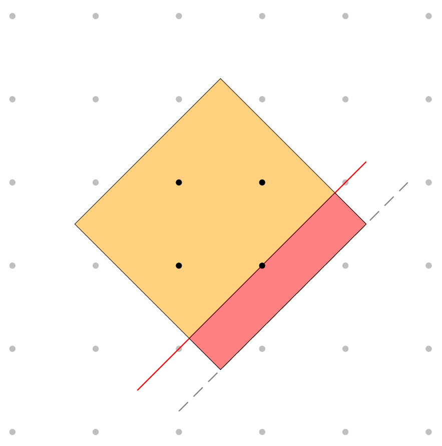
$$\left\lfloor \left(e_i A_B^{-1}\right) A \right\rfloor x \leq \left\lfloor \left(e_i A_B^{-1}\right) b \right\rfloor . \tag{4.6}$$



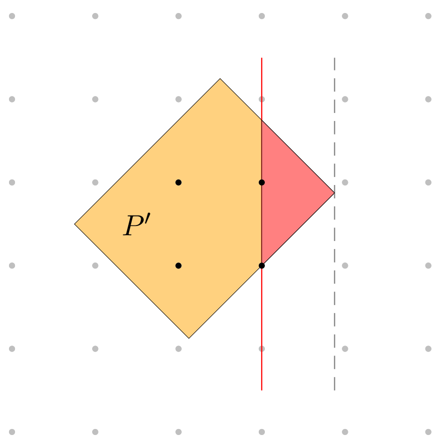
(a) A polyhedron P and its integer hull P_I in $\mathbb{R}^2$



(b) A valid inequality for P (gray dashed line) and the resulting CG cut (red solid line) which is valid for P_I. The cut depicted by the red dashed line is valid for P_I but cannot be derived from P as a CG cut.



(c) Another valid inequality for P (gray dashed line) and the resulting CG cut (red solid line) which is valid for P_I.



(d) The resulting polyhedron P' and a valid inequality for P' (gray dashed line). Now, the cut from Figure 4.1(b) can be derived as a CG cut from P' (red solid line).

Figure 4.1. Examples of Chvátal-Gomory cuts

Since we assumed that $x_i^* = (e_i A_B^{-1})b$ is non-integral the CG cut (4.6) is violated by the amount $x_i^* - \lfloor x_i^* \rfloor > 0$. The CG cut (4.6) is well known as the Gomory fractional cut [97, 99]. The Gomory fractional cut can thus be seen as a CG cut derived from a row of the simplex tableau.

4.3.1. $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts

The $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts (or $\{0, \frac{1}{2}\}$-cuts) of Caprara and Fischetti [42] are a special case of the general CG cuts. Here the components of the vector u of CG multipliers are constrained to be either 0 or $\frac{1}{2}$, i.e. we have $u \in \{0, \frac{1}{2}\}^m$. As mentioned above, one of the main reasons for studying $\{0, \frac{1}{2}\}$-cuts is that many problem-specific facet-defining inequalities for combinatorial optimization problems are actually $\{0, \frac{1}{2}\}$-cuts.

Proposition 4.2 ([42]). *The $\{0, \frac{1}{2}\}$-cut*

$$\lfloor uA \rfloor x \leq \lfloor ub \rfloor, \tag{4.7}$$

where $u \in \{0, \frac{1}{2}\}^m$ is valid for the IP (4.2).

Gentile et al. [94] showed that $\{0, \frac{1}{2}\}$-cuts are sufficient to describe the convex hull P_I of feasible solutions to the IP (4.2). Some CG inequalities may not be $\{0, \frac{1}{2}\}$-cuts with respect to the current polyhedron (or formulation) P, meaning that these CG inequalities can not be derived from linear combinations of the constraints defining P with weights $u \in \{0, \frac{1}{2}\}^m$. However, Gentile et al. showed that these inequalities can be derived as $\{0, \frac{1}{2}\}$-cuts from modified polyhedra P' in subsequent rounds of the CG procedure (see Figure 4.1).

Caprara and Fischetti [42] proved that the separation of $\{0, \frac{1}{2}\}$-cuts is $\mathcal{NP}$-hard. Given an arbitrary solution x^*, the most-violated $\{0, \frac{1}{2}\}$-cut is found by solving a separation problem similar to (4.5). In contrast to general CG cuts, however, every row i of (A, b) that is involved in the linear combination $(uA)x \leq ub$ is weighted with the same multiplier, namely $u_i = \frac{1}{2}$. Let $v \in \{0, 1\}^m$ be a binary vector such that $v = 2u$. The $\{0, \frac{1}{2}\}$-cut separation problem is then obtained by substituting $u = \frac{1}{2}v$ in (4.5). In addition, the $\{0, \frac{1}{2}\}$-cut (4.7) can be restated as

$$\left\lfloor \frac{1}{2}(vA) \right\rfloor x \leq \left\lfloor \frac{1}{2}(vb) \right\rfloor. \tag{4.8}$$

We now take a closer look at the objective function of the separation problem. Specifically, we investigate under what conditions a $\{0, \frac{1}{2}\}$-cut is violated. We can write

$$\begin{aligned} uA &= \lfloor uA \rfloor + \frac{1}{2}\left(vA \bmod 2\right), \\ ub &= \lfloor ub \rfloor + \frac{1}{2}\left(vb \bmod 2\right). \end{aligned} \tag{4.9}$$

Recall that the violation of a CG cut $\alpha x \leq \beta$ is given by $\alpha x^* - \beta$. In the special case of $\{0, \frac{1}{2}\}$-cuts we have

$$\begin{aligned} \alpha x^* - \beta &= \lfloor uA \rfloor x^* - \lfloor ub \rfloor, \\ &= \left(uA - \frac{1}{2}\left(vA \bmod 2\right)\right) x^* - \left(ub - \frac{1}{2}\left(vb \bmod 2\right)\right), \\ &= \frac{1}{2}\left(vb \bmod 2\right) - \frac{1}{2}\left(vA \bmod 2\right) x^* - u\left(b - Ax^*\right), \\ &= \frac{1}{2}\left(\left(vb \bmod 2\right) - \left(vA \bmod 2\right) x^* - vs^*\right). \end{aligned} \tag{4.10}$$

The violation is positive, if the inequality

$$\left(vb \bmod 2\right) > \left(vA \bmod 2\right) x^* + vs^* \tag{4.11}$$

holds. Note that both the left-hand side and the right-hand side of Inequality (4.11) take non-negative values.

Inequality (4.11) shows that the violation of a $\{0, \frac{1}{2}\}$-cut is dependent on the parity of the terms vb and vA. If vb is even, it follows that $\frac{1}{2}(vb)$ (or ub respectively) is integral. However, as mentioned above, non-integrality of ub is a necessary condition of a CG cut's being violated. Thus there is no violated $\{0, \frac{1}{2}\}$-cut with $vb \bmod 2 = 0$. Now, let $\bar{A} = A \bmod 2$, $\bar{b} = b \bmod 2$ where the modulo operation is applied component-wise. A $\{0, \frac{1}{2}\}$-cut is violated, then, if $v\bar{b}$ is odd and

$$\left(v\bar{A} \bmod 2\right) x^* + vs^* < 1. \tag{4.12}$$

Consider also the system $(\bar{A}, \bar{b})$. This system can be used to preprocess the $\{0, \frac{1}{2}\}$-cut separation problem. For instance, zero rows of $(\bar{A}, \bar{b})$ can be removed since they only affect the value of vs^* but not the parity of $v\bar{b}$ and $v\bar{A}$. Moreover, any row i of $(\bar{A}, \bar{b})$ having $s_i^* \geq 1$ can be removed since it will never be selected due to the condition (4.12). There are a number of additional preprocessing rules which we shall discuss in more detail in Part III of this thesis (see Section 8.8).

4.3.2. Strong Chvátal-Gomory Cuts

So far we discussed CG cuts and a subclass of the CG cuts known as $\{0, \frac{1}{2}\}$-cuts. In this section we address the strengthening of CG cuts. Firstly, we introduce some additional notation. Let $N = \{1, \dots, n\}$ and let $(uA)_j$ denote the j^{th} component of the vector uA. With this notation, the inequality $(uA)x \leq ub$ can be written as

$$\sum_{j \in N} (uA)_j \, x_j \leq ub \tag{4.13}$$

and the CG cut (4.4) reads

$$\sum_{j \in N} \left\lfloor (uA)_j \right\rfloor x_j \leq \lfloor ub \rfloor . \tag{4.14}$$

Using the integrality conditions on the variables one can apply a simple strengthening of the coefficients in the CG cut. Let $f_0 = ub - \lfloor ub \rfloor$ and $f_j = (uA)_j - \lfloor (uA)_j \rfloor$ for $j \in N$. Applying mixed-integer rounding to (4.13) we obtain the inequality

$$\sum_{j \in N} \left(\left\lfloor (uA)_j \right\rfloor + \frac{(f_j - f_0)^+}{1 - f_0} \right) x_j \leq \lfloor ub \rfloor , \tag{4.15}$$

where $(a)^+ = \max\{0, a\}$ (see Marchand and Wolsey [135]). This inequality is valid for (4.2) and clearly dominates the CG cut (4.14).

A different approach to strengthen CG cuts was proposed by Letchford and Lodi [129]:

Proposition 4.3 ([129]). *Suppose that $f_0 > 0$ and let $k \geq 1$ be the unique integer such that $\frac{1}{k+1} \leq f_0 < \frac{1}{k}$. Let $N_0 = \{j \in N : f_j \leq f_0\}$ and $N_p = \{j \in N : f_0 + \frac{(p-1)(1-f_0)}{k} < f_j \leq f_0 + \frac{p(1-f_0)}{k}\}$ for $p = 1, \dots, k$. The strong Chvátal-Gomory cut*

$$\sum_{p=0}^{k} \sum_{j \in N_p} \left(\left\lfloor (uA)_j \right\rfloor + \frac{p}{k+1} \right) x_j \leq \lfloor ub \rfloor \tag{4.16}$$

is valid for the IP (4.2) and dominates the CG cut (4.14).

While both strong CG cuts (4.16) and MIR cuts (4.15) dominate CG cuts (4.14) there is no dominance relationship between MIR cuts and strong CG cuts in general.

4.4. Cutting Planes for MIPs

This section deals with cutting planes for general MIPs. While all valid inequalities for pure IPs are CG inequalities, the CG rounding procedure can not be used to derive valid inequalities for general MIPs. To see this, let P again denote the polyhedron associated with LP relaxation of the MIP (2.1). Suppose, moreover, that the inequality $\alpha x \geq \beta$ is valid for P. As the terms $\alpha_j x_j$ with $j \in N \setminus N_I$ are not guaranteed to be integer-valued, the inequality $\lceil \alpha \rceil x \geq \lceil \beta \rceil$ is not valid for the MIP (2.1). A different approach for generating valid inequalities for MIPs is therefore needed.

Next we outline the basic concept that is used to generate cuts for MIPs. Suppose that x^* is a fractional basic solution to the LP relaxation of the MIP (2.1), i.e. a vertex of the polyhedron P. Now, consider a convex set $S \subseteq \mathbb{R}^n$ containing the fractional vertex x^* in its interior but no integral solutions. More formally, we require the convex set S to have the properties

$$x^* \in \operatorname{int}(S) \text{ and } x \notin \operatorname{int}(S), \forall x \in \mathbb{Z}^{N_I} \times \mathbb{R}^{N \setminus N_I}, \tag{4.17}$$

where $\operatorname{int}(S)$ denotes the interior of the set S. An inequality $\alpha x \geq \beta$ which only cuts off points $x \in \operatorname{int}(S)$ is valid for the MIP (2.1) as no points feasible to (2.1) lie in the interior of S. Thus cutting planes for MIPs can be obtained by first selecting a set S according to (4.17) and then computing a cut in relation to S.

Instead of working with convex sets S satisfying (4.17), we use disjunctions to express that a feasible solution is not allowed to lie in the interior of a set S. Thus all cutting planes derived using the above principle are in fact disjunctive cuts. In practice, the main questions are how to select the disjunction (or the set S respectively) and how to compute deep disjunctive cuts (see Section 2.3.2).

The results presented in the remainder of this chapter complement the basic theory of disjunctive inequalities discussed in Chapter 2. We outline several practical approaches for deriving disjunctive cuts. The most commonly used disjunctions for deriving cutting planes are split disjunctions which are defined by two hyperplanes. The resulting cuts are called split cuts. We formally introduce split disjunctions and split cuts in Section 4.4.1. In the succeeding sections we discuss several classes of split cuts. In Section 4.4.2 we deal with intersection cuts. We introduce Gomory mixed-integer cuts in Section 4.4.3. We also investigate k-cuts in Section 4.4.4, combined Gomory mixed-integer cuts in Section 4.4.5, reduce-and-split cuts in Section 4.4.6 and lift-and-project cuts in Section 4.4.7.

4.4.1. Split Cuts

Split cuts were introduced by Cook et al. [57]. These cuts are a special class of disjunctive cuts (see Section 2.3.1) generated from simple two-term disjunctions. Split cuts are very important since they subsume various other classes of cutting planes. In particular, all cutting planes presented in the remainder of this chapter are in fact split cuts.

We start with some basic definitions. A *split disjunction* is given by

$$(\pi x \leq \pi_0) \vee (\pi x \geq \pi_0 + 1)\,, \tag{4.18}$$

where $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$ and $\pi_j = 0$, $\forall j \in N \setminus N_I$. We shall write $D(\pi,\, \pi_0)$ to denote the disjunction (4.18). Clearly, any feasible solution to the MIP (2.1) satisfies one of the terms of $D(\pi,\, \pi_0)$. An inequality which is valid for a split disjunction is called a *split inequality* or *split cut*. With respect to the MIP (2.1), we call any valid inequality which is derived from a disjunction

$$\begin{pmatrix} Ax & \geq & b \\ -\pi x & \geq & -\pi_0 \end{pmatrix} \bigvee \begin{pmatrix} Ax & \geq & b \\ \pi x & \geq & \pi_0 + 1 \end{pmatrix} \tag{4.19}$$

a split inequality. Figure 4.2 shows how a split disjunction divides the feasible region of a set P into the two parts P_1 and P_2. In addition, a split inequality is shown. This inequality is valid for $\operatorname{conv}(P_1 \cup P_2)$.

As mentioned above, Nemhauser and Wolsey [142] showed that split cuts and MIR cuts are equivalent. Given a split cut derived from the disjunction (4.19) we can thus find a linear combination of the constraints of $Ax \geq b$ such that the MIR cut generated from this single-row relaxation is equivalent to the split cut.

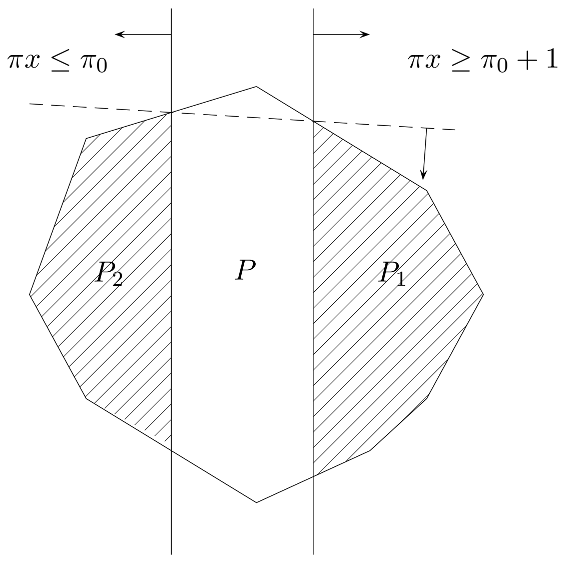


Figure 4.2. Split cut

Let us now consider the separation of split cuts. By associating multipliers $(u, u_0), (v, v_0) \in \mathbb{R}_+^{m+1}$ with the terms of the disjunction (4.19), we obtain the surrogates

$$((uA - u_0\pi)\, x \geq ub - u_0\pi_0) \vee ((vA + v_0\pi)\, x \geq vb + v_0\,(\pi_0 + 1))\,. \tag{4.20}$$

Given an LP solution x^*, the most-violated split cut $\alpha x \geq \beta$ is found by solving the mixed-integer non-linear program

$$\begin{array}{llllllll} \max & -\alpha x^* + \beta, & & & & & & \\ \text{s.t.} & \alpha & -uA & & +u_0\pi & & \geq & 0, \\ & \alpha & & -vA & & -v_0\pi & \geq & 0, \\ & -\beta & +ub & & -u_0\pi_0 & & \geq & 0, \\ & -\beta & & +vb & & +v_0\,(\pi_0+1) & \geq & 0, \\ & & ue & +ve & +u_0 & +v_0 & \leq & k, \end{array} \tag{4.21}$$

where $(u, u_0), (v, v_0) \in \mathbb{R}_+^{m+1}$, $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$, $\pi_j = 0$ for all $j \in N \setminus N_I$ and the vectors $(\alpha, \beta) \in \mathbb{R}^{n+1}$ are unrestricted in sign. The last inequality in (4.21) is the normalization constraint (2.28) where e is the vector of all ones of appropriate dimension. The non-linearity of (4.21) is caused by the multiplication of the multipliers (u, u_0) and (v, v_0) with the vectors (π, π_0) defining the split disjunction. Once the split disjunction is fixed, (4.21) reduces to an LP of the form (2.24). Caprara and Letchford proved that optimizing over the split closure is $\mathcal{NP}$-hard. Balas and Saxena [31] restate the problem (4.21) as a parametric MIP using the normalization constraint $u_0 + v_0 = 1$ and show that the elementary split closure gives a tight approximation of the convex hull of the feasible solutions of many practical MIPs.

Finally, we offer some comments on the connection between Chvátal-Gomory cuts and split cuts. In fact any CG cut is a split cut, implying that the split closure of a polyhedron is contained in its CG closure. Let $ax \leq a_0$ be a valid inequality for the polyhedron P associated with the LP relaxation of a pure integer program. The CG cut $\lfloor a \rfloor x \leq \lfloor a_0 \rfloor$ is then valid for the integer hull P_I. To see that this CG cut is a split cut, consider the split disjunction $(\lfloor a \rfloor x \leq \lfloor a_0 \rfloor) \vee (\lfloor a \rfloor x \geq \lfloor a_0 \rfloor + 1)$, i.e. $D(\lfloor a \rfloor, \lfloor a_0 \rfloor)$, and observe that $P \cap \{x : \lfloor a \rfloor x \geq \lfloor a_0 \rfloor + 1\} = \emptyset$.

4.4.2. Intersection Cuts

Intersection cuts were introduced by Balas [17]. These cuts are derived from a basic solution to the LP relaxation of an MIP and a violated disjunction. Furthermore, intersection cuts have a nice geometric interpretation. In this section we concentrate on intersection cuts derived from split disjunctions. Nevertheless, intersection cuts can also be derived using general disjunctions.

Let B be a basis of the LP relaxation of the MIP (2.3) and denote the associated basic solution by x^*. Consider also the rows of the simplex tableau

$$x_i = x_i^* - \sum_{j \in J} \bar{a}_{ij} x_j, \quad i \in B_I, \tag{4.22}$$

associated with the basic integer-constrained variables. Moreover, we have the trivial equation

$$x_j = 0 + x_j, \quad j \in J_I, \tag{4.23}$$

for all non-basic integer-constrained variables. Suppose that the split disjunction $D(\pi,\, \pi_0)$ is given. We assume that the disjunction is violated by x^*, i.e. we have that $\epsilon(\pi, \pi_0) = \pi x^* - \pi_0 > 0$.

In the following, we shall demonstrate a way to derive the intersection cut. Firstly, let us construct a linear combination of Equations (4.22) and (4.23). The weight we associate with each equation is the corresponding entry in the split disjunction, namely π_i for $i \in N_I$. The result of this linear combination is

$$\pi x = \pi x^* + \sum_{j \in J} \left(\pi r^j\right) x_j, \tag{4.24}$$

where r^j is defined as

$$r^j_k = \begin{cases} -\bar{a}_{kj} & \text{if } k \in B, \\ 1 & \text{if } k = j, \\ 0 & \text{otherwise,} \end{cases} \tag{4.25}$$

for $j \in J$. Substituting for πx in $D(\pi,\, \pi_0)$ and rewriting yields the disjunction

$$\left(-\sum_{j \in J} \left(\pi r^j\right) x_j \geq \epsilon\left(\pi, \pi_0\right)\right) \vee \left(\sum_{j \in J} \left(\pi r^j\right) x_j \geq 1 - \epsilon\left(\pi, \pi_0\right)\right). \tag{4.26}$$

Applying Proposition 2.1 we obtain the following result.

Proposition 4.4 ([17])**.** *Suppose that a basis B of the LP relaxation, the corresponding basic solution x^* and a split disjunction $D(\pi,\, \pi_0)$ are given. Let the split disjunction be violated by x^*, that is $\epsilon(\pi, \pi_0) = \pi x^* - \pi_0 > 0$. Then the intersection cut*

$$\sum_{j \in J} \left|\pi r^j\right| x_j \geq \min\left\{\epsilon\left(\pi, \pi_0\right), 1 - \epsilon\left(\pi, \pi_0\right)\right\} \tag{4.27}$$

associated with the basis B and the split disjunction $D(\pi,\, \pi_0)$ is valid for the MIP (2.3)*.*

Note that the intersection cut (4.27) is a split cut as it is derived from a split disjunction. However, it is of key importance that intersection cuts are generated from bases of the LP relaxation. Observe that the left-hand side of the intersection cut only contains non-basic variables which are at their bounds. Since we assumed all variables to have a lower bound of zero and an infinite upper bound, the whole left-hand side also has the value zero. Moreover, the right-hand side of the intersection cut is positive as we selected a violated split disjunction, i.e. we assumed that $\epsilon(\pi, \pi_0) > 0$. Therefore the intersection cut is violated by the current basic solution x^* to the LP relaxation.

Note that the coefficient of a continuous variable in the intersection cut derived from the split disjunction $D(\pi,\ \pi_0)$ and the basis B is not affected by components of π corresponding to non-basic variables. More formally, we have

$$\pi r^j = \begin{cases} \sum\limits_{i \in B} \pi_i r_i^j & \text{if } j \in J \setminus J_I, \\ \sum\limits_{i \in B} \pi_i r_i^j + \pi_j & \text{if } j \in J_I. \end{cases} \tag{4.28}$$

Depending on the way in which the disjunction (4.26) is written, different intersection cuts can be obtained. For instance, multiply the first term of the disjunction (4.26) with $1 - \epsilon(\pi, \pi_0) > 0$ and the second term with $\epsilon(\pi, \pi_0) > 0$. The intersection cut is then given by

$$\begin{aligned} \sum_{j \in J} \max \Big\{ & \left(-\pi r^j\right) \left(1 - \epsilon\left(\pi, \pi_0\right)\right), \\ & \left(\pi r^j\right) \epsilon\left(\pi, \pi_0\right) \Big\} x_j \geq \epsilon\left(\pi, \pi_0\right) \left(1 - \epsilon\left(\pi, \pi_0\right)\right). \end{aligned} \tag{4.29}$$

If the disjunction (4.26) is normalized[5], a third version of the intersection cut is obtained:

$$\sum_{j \in J} \max \left\{ \frac{-\pi r^j}{\epsilon\left(\pi, \pi_0\right)}, \frac{\pi r^j}{1 - \epsilon\left(\pi, \pi_0\right)} \right\} x_j \geq 1 \tag{4.30}$$

As mentioned before, intersection cuts have a very nice geometric interpretation (see Figure 4.3), which we shall discuss next. Let us again suppose that the basic solution x^* solves the LP relaxation and violates the split disjunction $D(\pi,\ \pi_0)$. Now consider the following relaxation of P

$$P\left(B\right) = \left\{x \in \mathbb{R}^n : Ax = b, x_j \geq 0, \forall j \in J\right\}, \tag{4.31}$$

[5]Given a disjunction $\bigvee_{i \in Q} (a^i x \geq b^i)$ with $a^i \in \mathbb{R}^n$, $b^i \in \mathbb{R}^1$ and provided that $b^i > 0, \forall i \in Q$, the normalized disjunction is $\bigvee_{i \in Q} ((\frac{a^i}{b^i})x \geq 1)$.

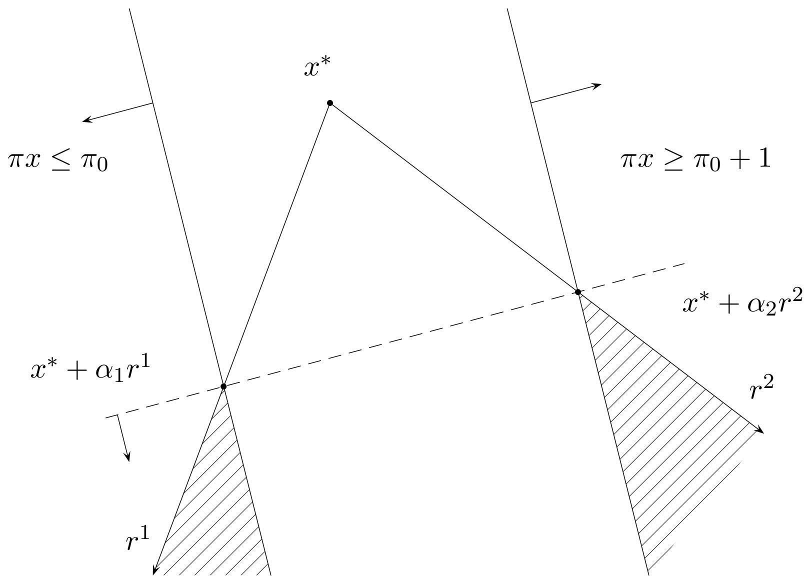


Figure 4.3. Intersection cut

where we drop the non-negativity conditions on the basic variables. In the case of non-degeneracy, these are the only non-binding constraints (with respect to the basic solution x^*). We can alternatively write $P(B) = x^* + C$ where C is the polyhedral cone $C = \{x \in \mathbb{R}^n : Ax = 0, x_j \geq 0, \forall j \in J\}$. The extreme rays of the cone C are given by the vectors r^j as defined in Equation (4.25) for $j \in J$. Now, for $j \in J$ define

$$\alpha_j = \begin{cases} -\frac{\epsilon(\pi,\pi_0)}{\pi r^j} & \text{if } \pi r^j < 0, \\ \frac{1-\epsilon(\pi,\pi_0)}{\pi r^j} & \text{if } \pi r^j > 0, \\ \infty & \text{otherwise.} \end{cases} \tag{4.32}$$

The scalars α_j can be interpreted in the following way. Consider the half-line starting from x^* in the direction r^j which is given by $x^* + \alpha r^j$ where $\alpha \in \mathbb{R}_+$. The scalar α_j is the smallest $\alpha \in \mathbb{R}_+$ such that $x^* + \alpha r^j$ satisfies the split disjunction $D(\pi, \pi_0)$. Therefore the point $x^* + \alpha_j r^j$ is the point at which the half-line $x^* + \alpha r^j$ and the hyperplanes $\pi x = \pi_0$ or $\pi x = \pi_0 + 1$ *intersect* (see Figure 4.3). This is why the resulting cut is called the intersection cut. Note that $\alpha_j = \infty$, if the inner product πr^j is zero and the extreme ray r^j is thus parallel to the hyperplanes $\pi x = \pi_0$ and $\pi x = \pi_0 + 1$. With the aid of the scalars (4.32), we can write the intersection cut from the basis B and the disjunction $\pi x \leq \pi_0$ or $\pi x \geq \pi_0 + 1$ in the form

$$\sum_{j \in J} \frac{x_j}{\alpha_j} \geq 1. \tag{4.33}$$

A natural question is whether the split disjunction $D(\pi,\, \pi_0)$ can be improved in order to obtain stronger split cuts. More precisely, we consider replacing $D(\pi,\, \pi_0)$ by

$$((\pi - h)\, x \leq \pi_0) \vee ((\pi - h)\, x \geq \pi_0 + 1)\,, \tag{4.34}$$

where, as before, $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$ and $\pi_j = 0,\ \forall j \in N \setminus N_I$. We denote the disjunction (4.34) by $D(\pi - h,\, \pi_0)$. Let $h \in \mathbb{Z}^n$ be a vector with the characteristic that $h_j = 0,\ \forall j \notin J_I$. In other words, we modify the split disjunction on the non-basic integer-constrained variables. Hence we have that $(\pi - h)x^* = \pi x^*$ and it follows that the split disjunctions $D(\pi,\, \pi_0)$ and $D(\pi - h,\, \pi_0)$ are violated by the same amount, i.e. $\epsilon(\pi - h, \pi_0) = \epsilon(\pi, \pi_0)$. Summing up Equations (4.22) and (4.23) with weights $\pi_i - h_i$ for $i \in N_I$ yields the linear combination

$$(\pi - h)\, x = (\pi - h)\, x^* + \sum_{j \in J} \Big((\pi - h)\, r^j \Big)\, x_j. \tag{4.35}$$

By inserting Equation (4.35) into the split disjunction $D(\pi - h,\, \pi_0)$, normalizing the disjunction and applying the disjunctive principle, we obtain the intersection cut

$$\sum_{j \in J} \max \left\{ \frac{hr^j - \pi r^j}{\epsilon\,(\pi, \pi_0)}, \frac{\pi r^j - hr^j}{1 - \epsilon\,(\pi, \pi_0)} \right\} x_j \geq 1. \tag{4.36}$$

Observe that $hr^j = h_j r^j_j = h_j$ for $j \in J$. Thus we have that $hr^j \geq 0$ if the non-basic variable x_j is integer-constrained and $hr^j = 0$ otherwise. Consequently, we can restate the intersection cut as

$$\sum_{j \in J_I} \max \left\{ \frac{h_j - \pi r^j}{\epsilon\,(\pi, \pi_0)}, \frac{\pi r^j - h_j}{1 - \epsilon\,(\pi, \pi_0)} \right\} x_j + \sum_{j \in J \setminus J_I} \max \left\{ \frac{-\pi r^j}{\epsilon\,(\pi, \pi_0)}, \frac{\pi r^j}{1 - \epsilon\,(\pi, \pi_0)} \right\} x_j \geq 1. \tag{4.37}$$

We are now concerned with the question of how to choose the vector h to obtain the strongest split cuts. We already noted that the violation of the cut (4.37) is the same as of (4.30). However, it turns out that the size of the coefficients on the non-basic variables greatly affects the strength of an intersection cut. For example, consider the intersection cut $\gamma x \geq 1$. The distance between the basic solution x^* and its orthogonal projection on the hyperplane $\gamma x = 1$ is given by

$$\frac{1}{\|\gamma\|}, \tag{4.38}$$

where $\|\cdot\|$ denotes the Euclidean norm. It is therefore desirable to obtain cut coefficients that are as small as possible to increase the distance cut off. It can be verified [7, 27] that the smallest coefficients are given by either setting $h_j = \lfloor \pi r^j \rfloor$ or $h_j = \lceil \pi r^j \rceil$ for $j \in J_I$. We get

$$\begin{aligned} &\min\left\{ \max\left\{ \frac{\lceil \pi r^j \rceil - \pi r^j}{\epsilon(\pi, \pi_0)}, \frac{\pi r^j - \lceil \pi r^j \rceil}{1 - \epsilon(\pi, \pi_0)} \right\}, \right. \\ &\qquad \left. \max\left\{ \frac{\lfloor \pi r^j \rfloor - \pi r^j}{\epsilon(\pi, \pi_0)}, \frac{\pi r^j - \lfloor \pi r^j \rfloor}{1 - \epsilon(\pi, \pi_0)} \right\} \right\}, \\ &= \min\left\{ \frac{\lceil \pi r^j \rceil - \pi r^j}{\epsilon(\pi, \pi_0)}, \frac{\pi r^j - \lfloor \pi r^j \rfloor}{1 - \epsilon(\pi, \pi_0)} \right\}, \end{aligned} \tag{4.39}$$

for $j \in J_I$.

Proposition 4.5 ([27]). *Suppose that a basis B of the LP relaxation, the corresponding basic solution x^* and a split disjunction $D(\pi, \pi_0)$ are given. Let the split disjunction be violated by x^*, that is $\epsilon(\pi, \pi_0) = \pi x^* - \pi_0 > 0$. Then the strengthened intersection cut (or strengthened split cut)*

$$\begin{aligned} &\sum_{j \in J_I} \min\left\{ \frac{\lceil \pi r^j \rceil - \pi r^j}{\epsilon(\pi, \pi_0)}, \frac{\pi r^j - \lfloor \pi r^j \rfloor}{1 - \epsilon(\pi, \pi_0)} \right\} x_j \\ &\qquad + \sum_{j \in J \setminus J_I} \max\left\{ \frac{-\pi r^j}{\epsilon(\pi, \pi_0)}, \frac{\pi r^j}{1 - \epsilon(\pi, \pi_0)} \right\} x_j \geq 1 \end{aligned} \tag{4.40}$$

associated with the basis B and the split disjunction $D(\pi, \pi_0)$ is valid for the MIP (2.3).

Concerning the split disjunction $D(\pi - h,\, \pi_0)$, it follows that we have to choose $h_j = \lfloor \pi r^j \rfloor$ if the inequality

$$\begin{aligned} \frac{\lceil \pi r^j \rceil - \pi r^j}{\epsilon\left(\pi, \pi_0\right)} &> \frac{\pi r^j - \lfloor \pi r^j \rfloor}{1 - \epsilon\left(\pi, \pi_0\right)}, \\ \Longleftrightarrow \quad \left\lceil \pi r^j \right\rceil - \pi r^j &> \epsilon\left(\pi, \pi_0\right), \end{aligned} \tag{4.41}$$

holds and $h_j = \lceil \pi r^j \rceil$ otherwise.

Proposition 4.6 ([8])**.** *Suppose that a basis B of the LP relaxation, the corresponding basic solution x^* and a split disjunction $D(\pi,\, \pi_0)$ are given. Let the split disjunction be violated by x^*, that is $\epsilon(\pi, \pi_0) = \pi x^* - \pi_0 > 0$. The strengthened split disjunction is then given by $D(\pi - h,\, \pi_0)$ where*

$$h_j = \begin{cases} \lfloor \pi r^j \rfloor & \text{if } \lceil \pi r^j \rceil - \pi r^j > \epsilon\left(\pi, \pi_0\right), \\ \lceil \pi r^j \rceil & \text{if } \lceil \pi r^j \rceil - \pi r^j \le \epsilon\left(\pi, \pi_0\right), \end{cases} \tag{4.42}$$

for $j \in J_I$.

Any split inequality is equal to or dominated by a split cut derived from a basis of the LP relaxation and a split disjunction. The split closure of an MIP (i.e. the intersection of all split cuts) can thus be obtained using only intersection cuts by considering also non-optimal and infeasible bases. This correspondence was established by Balas and Perregaard [30] for mixed 0-1 programs. Andersen et al. [9] generalized it to mixed-integer programs.

In the previous discussion we concentrated on generating intersection cuts from split disjunctions. In general, intersection cuts can also be derived from more complex convex sets (or, in other words, from more complex disjunctions). However, we must still require that these sets contain the fractional basic solution to the LP relaxation and no feasible solution to the (mixed) integer program.

Balas and Margot [28] recently generalized the notion of intersection cuts by replacing the polyhedral cone used to derive these cuts by a more general polyhedron.

4.4.3. Gomory Mixed-Integer Cuts

Gomory mixed-integer (GMI) cuts were introduced by Gomory [98] in the early 1960s as one of the first classes of cutting planes for MIPs. One of the merits of GMI cuts is their separation: GMI cuts can be easily read from rows of the simplex tableau associated with fractional integer variables.

We assume that the MIP (2.3) is given. Consider a row of the simplex tableau

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} x_j, \quad i \in B_I, \tag{4.43}$$

associated with the basic integer-constrained variable x_i which has a fractional value in the solution to the LP relaxation of the MIP (2.3). More formally, we assume that $f_{i0} = \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor > 0$. In addition, let $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$ for $j \in J$.

Suppose we would like to generate an intersection cut (see Section 4.4.2) that cuts off x^*. Remember that intersection cuts are generated from a basis and a violated disjunction. We use the current optimal basis B of the LP relaxation and the simple split disjunction

$$(x_i \leq \lfloor \bar{a}_{i0} \rfloor) \vee (x_i \geq \lfloor \bar{a}_{i0} \rfloor + 1) . \tag{4.44}$$

Clearly, this disjunction is violated since $\epsilon(\pi, \pi_0) = \pi x^* - \pi_0 = \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor = f_{i0} > 0$. Furthermore, it is equivalent to $D(\pi, \lfloor \pi x^* \rfloor)$ where $\pi = e_i$ and e_i is the i^{th} unit vector. As disjunctions of the form (4.44) are imposed only on a single variable, they are referred to as *elementary disjunctions.* The intersection cut from the convex set $\{x \in \mathbb{R}^n : \lfloor \bar{a}_{i0} \rfloor \leq x_i \leq \lceil \bar{a}_{i0} \rceil\}$ is equal to the simple disjunctive inequality derived from the above split disjunction. Indeed, using the split disjunction (4.44) (cf. Proposition 4.4) we obtain the intersection cut

$$\sum_{j \in J} \max \left\{ \bar{a}_{ij} \left(1 - f_{i0}\right), -\bar{a}_{ij} f_{i0} \right\} x_j \geq f_{i0} \left(1 - f_{i0}\right) . \tag{4.45}$$

This cut can be strengthened using Proposition 4.5. We get the strengthened intersection cut

$$\sum_{j\in J_I} \min\left\{f_{ij}\left(1-f_{i0}\right),\left(1-f_{ij}\right)f_{i0}\right\}x_j \\ + \sum_{j\in J\setminus J_I} \max\left\{\bar{a}_{ij}\left(1-f_{i0}\right),-\bar{a}_{ij}f_{i0}\right\}x_j \geq f_{i0}\left(1-f_{i0}\right). \quad (4.46)$$

By rearranging the terms on the left-hand side and dividing by $1 - f_{i0}$ we obtain the GMI cut. So we have just proved the following result.

Proposition 4.7 ([98])**.** *The Gomory mixed-integer cut generated from a row of the simplex tableau* (4.43)

$$\sum_{j\in J_I: f_{ij}\leq f_{i0}} f_{ij}x_j + \sum_{j\in J_I: f_{ij}>f_{i0}} \frac{f_{i0}\left(1-f_{ij}\right)}{1-f_{i0}}x_j \\ + \sum_{j\in J\setminus J_I: \bar{a}_{ij}\geq 0} \bar{a}_{ij}x_j + \sum_{j\in J\setminus J_I: \bar{a}_{ij}<0} \frac{f_{i0}\left(-\bar{a}_{ij}\right)}{1-f_{i0}}x_j \geq f_{i0} \quad (4.47)$$

is valid for the MIP (2.3)*.*

The GMI cut is equivalent to the strengthened split (or intersection) cut from the basis B and the disjunction (4.44). Thus GMI cuts are also split cuts. Moreover, it can be shown that split cuts and GMI cuts are equivalent [142] with respect to their elementary closures. The GMI cut can alternatively be derived by strengthening the split disjunction (4.44) (cf. Proposition 4.6) before applying the intersection cut. The strengthened version of the split disjunction (4.44) is given by $D(\pi,\, \pi_0)$ with

$$\pi_j = \begin{cases} \lfloor \bar{a}_{ij} \rfloor & \text{if } j \in J_I \text{ and } f_{ij} \leq f_{i0}, \\ \lceil \bar{a}_{ij} \rceil & \text{if } j \in J_I \text{ and } f_{ij} > f_{i0}, \\ 1 & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases} \quad (4.48)$$

for $j \in N$ and $\pi_0 = \lfloor \bar{a}_{i0} \rfloor$. The intersection cut generated from the basis B and this disjunction is equivalent to the GMI cut obtained from (4.43). It is straightforward to see that

$$\epsilon(\pi, \pi_0) = \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor = x_i^* - \lfloor x_i^* \rfloor = f_{i0} \tag{4.49}$$

and

$$\begin{aligned} \pi r^j &= \pi_i r_i^j + \pi_j r_j^j = r_i^j + \pi_j = -\bar{a}_{ij} + \pi_j, \\ &= \begin{cases} -f_{ij} & \text{if } j \in J_I \text{ and } f_{ij} \leq f_{i0}, \\ 1 - f_{ij} & \text{if } j \in J_I \text{ and } f_{ij} > f_{i0}, \\ -\bar{a}_{ij} & \text{if } j \in J \setminus J_I, \end{cases} \end{aligned} \tag{4.50}$$

using (4.48) and the definition of the extreme rays (4.25) of the polyhedral cone defined by the basis B. Inserting these values into (4.30) gives the GMI cut. As MIR cuts and GMI cuts are equivalent, the GMI cut can also be obtained by applying mixed-integer rounding [135] to the tableau row (4.43). Thus a GMI cut is in fact a MIR cut derived from a row of the simplex tableau.

GMI cuts can not only be derived from single rows of the simplex tableau but also from linear combinations of these rows.

Proposition 4.8 ([46]). *Suppose the rows of the simplex tableau* (4.43) *associated with basic integer-constrained variables are given. Moreover, let $\pi \in \mathbb{Z}^n$ be a vector with $\pi_i = 0$ for $i \notin B_I$. Define $\bar{a}_0 = \sum_{i \in B_I} \pi_i \bar{a}_{i0}$, $f_0 = \bar{a}_0 - \lfloor \bar{a}_0 \rfloor$ and $\bar{a}_j = \sum_{i \in B_I} \pi_i \bar{a}_{ij}$, $f_j = \bar{a}_j - \lfloor \bar{a}_j \rfloor$ for $j \in J$. Then the Gomory mixed-integer cut*

$$\begin{aligned} \sum_{j \in J_I : f_j \leq f_0} f_j x_j + \sum_{j \in J_I : f_j > f_0} \frac{f_0 (1 - f_j)}{1 - f_0} x_j & \\ + \sum_{j \in J \setminus J_I : \bar{a}_j \geq 0} \bar{a}_j x_j + \sum_{j \in J \setminus J_I : \bar{a}_j < 0} \frac{f_0 (-\bar{a}_j)}{1 - f_0} x_j &\geq f_0 \end{aligned} \tag{4.51}$$

is valid for the MIP (2.3)*.*

The GMI cut (4.51) can also be obtained as a strengthened intersection cut from the basis B and the split disjunction $D(\pi, \lfloor \pi x^* \rfloor)$ where π is chosen as in Proposition 4.8.

The Quality of a Gomory Mixed-Integer Cut

The question of how to strengthen GMI cuts is related to the problem of measuring cut quality. Let $\alpha x \geq \beta$ be an arbitrary GMI cut. The violation of the GMI cut is given by

$$\beta - \alpha x^* = \beta \tag{4.52}$$

as $\alpha x^* = 0$. Thus the violation is equal to the fractional part of the right-hand side of the corresponding simplex tableau row (or the violation of the split disjunction respectively). We may also consider the distance between the solution x^* and its orthogonal projection on the cut hyperplane $\alpha x = \beta$ which is defined as

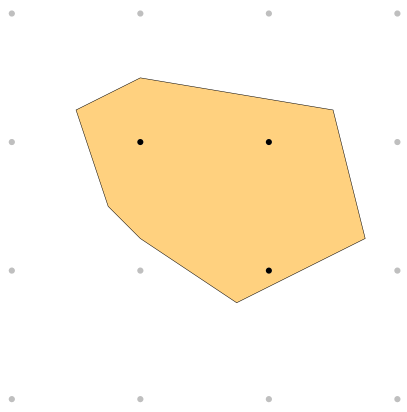
$$\frac{\beta - \alpha x^*}{\|\alpha\|} = \frac{\beta}{\|\alpha\|}. \tag{4.53}$$

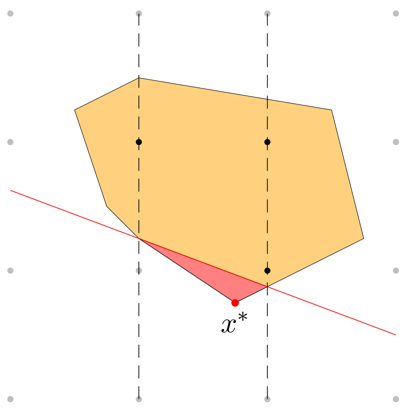
The value of (4.53) is also referred to as the distance cut off. Observe that to enhance the distance cut off, we can either try to increase the fractional part of the right-hand side of a tableau row (numerator) or decrease the size of the coefficients in the GMI cut (denominator). The coefficients of the integer-constrained variables in a GMI cut are in the interval $[0, 1]$ while coefficients on continuous variables are not bounded and depend on the size of the entries in the corresponding simplex tableau row.

In the following sections we present four approaches which improve the performance of the GMI cuts by increasing the violation or the distance cut off. These approaches either manipulate the disjunction (see Figure 4.4) or the basis (see Figure 4.5) on which a GMI cut (or strengthened intersection cut) is based.

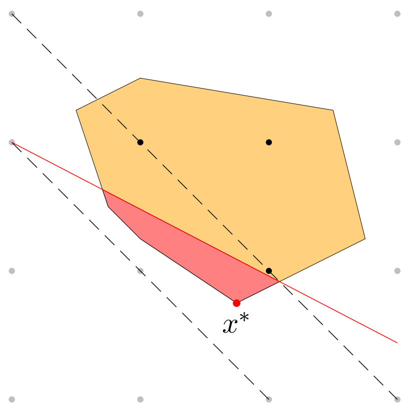
4.4.4. K-Cuts

In this section we discuss an approach to improve the performance of the Gomory mixed-integer cuts developed by Cornuéjols et al. [61]. Reconsider a row of the simplex tableau (4.43) associated with a basic integer-constrained variable x_i which is fractional. Recall, moreover, that the GMI cut generated from this row is equivalent to the strengthened intersection cut from the basis B and the elementary split disjunction (4.44). The approach outlined in this section modifies this disjunction.


(a) A polyhedron

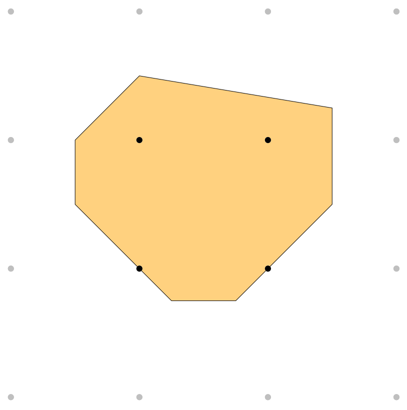


(b) A basic solution x^*, a split disjunction and the corresponding intersection cut

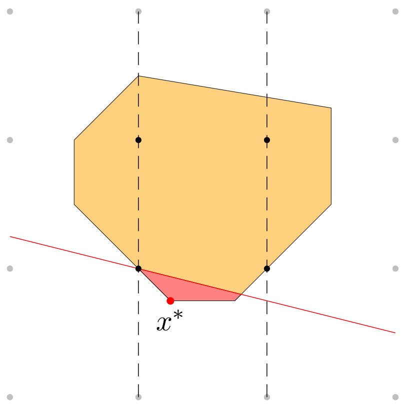


(c) A modified split disjunction which produces a deeper intersection cut

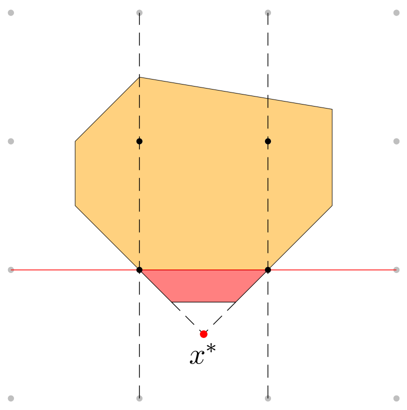
Figure 4.4. Modifying the disjunction to obtain deeper intersection cuts



(a) A polyhedron



(b) A basic solution x^*, a split disjunction and the corresponding intersection cut



(c) An infeasible basic solution x^* which produces a deeper intersection cut

Figure 4.5. Modifying the basis to obtain deeper intersection cuts

Specifically, Cornuéjols et al. [61] consider the disjunction

$$(kx_i \leq \lfloor k\bar{a}_{i0} \rfloor) \vee (kx_i \geq \lfloor k\bar{a}_{i0} \rfloor + 1)\,, \tag{4.54}$$

where k is an integer with $k \neq 0$. We denote this disjunction by $D(\bar{\pi},\, \lfloor \bar{\pi}x^* \rfloor)$ where $\bar{\pi} = k\pi = ke_i$, and let $\bar{\pi}_0 = \lfloor k\bar{a}_{i0} \rfloor$. Clearly, the multiplication with $k \neq 0$ affects the violation of the split disjunction. More precisely, assuming that $k\bar{a}_{i0}$ is non-integral, we have $\epsilon(\bar{\pi}, \bar{\pi}_0) = k\bar{a}_{i0} - \lfloor k\bar{a}_{i0} \rfloor > 0$. Now consider the intersection cut generated from the basis B and the modified split disjunction (4.54)

$$\sum_{j\in J} \max \left\{ \frac{k\left(-\pi r^j\right)}{\epsilon\left(\bar{\pi}, \bar{\pi}_0\right)}, \frac{k\left(\pi r^j\right)}{1 - \epsilon\left(\bar{\pi}, \bar{\pi}_0\right)} \right\} x_j \geq 1. \tag{4.55}$$

For each $k \neq 0$ a variation of the plain intersection cut ($k = 1$) is obtained. Note that the violation of the split disjunction $\epsilon(\bar{\pi}, \bar{\pi}_0)$ is equal to the violation of the GMI cut (cf. Proposition 4.7). Therefore some values of k may increase the amount by which the associated GMI cut is violated. Moreover, it becomes apparent from (4.55) that the size of the non-zero integer k has a direct influence on the size of the coefficients in the intersection cut. Specifically, large values for k lead to large coefficients in the intersection cut. Note that this is undesirable since it decreases $1/\|\gamma\|$, i.e. the distance cut off. The strengthening of Proposition 4.5 resolves this issue for the integer-constrained variables. On the other hand, the size of the coefficients on the continuous variables in the intersection cut and the GMI cut remains proportional to the size of k. This situation is also observed by Cornuéjols et al. [61] who report on a deterioration of the coefficients on continuous variables in preliminary experiments as k increases.

Cornuéjols et al. call the strengthened intersection cut from the modified disjunction (4.54) a *k-cut*. Alternatively the k-cut can be derived as the GMI cut from the scaled tableau row

$$kx_i = k\bar{a}_{i0} - \sum_{j\in J} \left(k\bar{a}_{ij}\right) x_j. \tag{4.56}$$

The following proposition states this result more formally.

Proposition 4.9 ([61]). *Suppose a row of the simplex tableau* (4.43) *and a non-zero integer k are given. Moreover, let $f_{i0} = k\bar{a}_{i0} - \lfloor k\bar{a}_{i0} \rfloor$ and $f_{ij} = k\bar{a}_{ij} - \lfloor k\bar{a}_{ij} \rfloor$ for $j \in J$. Then the k-cut*

$$\sum_{j \in J_I : f_{ij} \le f_{i0}} f_{ij} x_j + \sum_{j \in J_I : f_{ij} > f_{i0}} \frac{f_{i0}\,(1 - f_{ij})}{1 - f_{i0}} x_j + \sum_{j \in J \setminus J_I : k\bar{a}_{ij} \ge 0} k\bar{a}_{ij} x_j + \sum_{j \in J \setminus J_I : k\bar{a}_{ij} < 0} \frac{f_{i0}\,(-k\bar{a}_{ij})}{1 - f_{i0}} x_j \ge f_{i0} \tag{4.57}$$

is valid for the MIP (2.3).

Cornuéjols et al. [61] prove that in the pure integer case k-cuts perform variable-wise better than GMI cuts with exactly fifty percent probability.

4.4.5. Combined Gomory Mixed-Integer Cuts

Ceria et al. [46] present a method which obtains strengthened GMI cuts by modifying the underlying split disjunctions. They argue that the quality of a split (or intersection) cut is, among other factors, influenced by the violation of the split disjunction and the size of the coefficients of the integer variables. As a result, they propose a procedure controlling the latter two factors by constructing split disjunctions on several basic integer variables.

Ceria et al. [46] assume that the rows of the simplex tableau (4.43) are given in rational data. We have

$$x_i = \frac{e_{i0}}{D} - \sum_{j \in J} \frac{e_{ij}}{D} x_j, \quad i \in B_I, \tag{4.58}$$

where $e_{i0}, D \in \mathbb{Z}$ and $e_{ij} \in \mathbb{Z}$ for $i \in B_I$ and $j \in J$. The integer D is the common denominator in which the coefficients of the simplex tableau can be expressed. This representation of the simplex tableau allows for a detailed analysis of the connection between the properties of the split disjunction and the strengthened intersection cut generated from it. In particular we shall see that it enables us to select the split disjunction such that a strengthened intersection cut with a maximal right-hand side and minimal coefficients on certain variables is obtained. To simplify the notation, let $B_I = \{1, \dots, r\}$.

Maximizing the Violation of a Split Disjunction

Firstly, we deal with the violation of the split disjunction which is given by

$$\epsilon(\pi, \pi_0) = \sum_{i=1}^{r} \pi_i \left(\frac{e_{i0}}{D}\right) - \left\lfloor \sum_{i=1}^{r} \pi_i \left(\frac{e_{i0}}{D}\right) \right\rfloor . \tag{4.59}$$

Let $< e_{10}, e_{20}, \ldots, e_{r0}, D >$ be the greatest common divisor of $e_{10}, e_{20}, \ldots, e_{r0}$ and D. Suppose we would like to obtain the violation $\epsilon(\pi, \pi_0) = \frac{e}{D}$. Suppose furthermore that $< e_{10}, e_{20}, \ldots, e_{r0}, D >$ divides e. Ceria et al. [46] show that this specific violation is obtained by choosing the multipliers π_i (or the disjunction π) appropriately.

Proposition 4.10 ([46]). *The violation $\epsilon(\pi, \pi_0) = \frac{e}{D}$ can be obtained by setting*

$$\pi_i = D - lp_i, \quad i = 1, \ldots, r, \tag{4.60}$$

where $l = \frac{D-e}{<e_{10}, e_{20}, \ldots, e_{r0}, D>}$ and $p_1, p_2, \ldots, p_r$ and q are integers which solve the diophantine equation

$$< e_{10}, e_{20}, \ldots, e_{r0}, D >= e_{10}p_1 + e_{20}p_2 + \ldots + e_{r0}p_r + qD. \tag{4.61}$$

The requirement that $< e_{10}, e_{20}, \ldots, e_{r0}, D >$ divides e must be made to ensure that l is integral. It follows that π_i is integral for $i = 1, \ldots, r$. With the proposed choice of the multipliers π_i we have

$$\begin{aligned} \sum_{i=1}^{r} \pi_i \left(\frac{e_{i0}}{D}\right) &= \sum_{i=1}^{r} (D - lp_i) \left(\frac{e_{i0}}{D}\right), \\ &= \sum_{i=1}^{r} e_{i0} - \frac{l}{D} \sum_{i=1}^{r} p_i e_{i0}, \\ &= \left(\sum_{i=1}^{r} e_{i0} + lq - 1\right) + \frac{e}{D}. \end{aligned} \tag{4.62}$$

As the term in the brackets is integral, we obtain the violation $\epsilon(\pi, \pi_0) = \frac{e}{D}$. Note that the maximal value of $\epsilon(\pi, \pi_0)$ in Equation (4.59) is $\frac{D-<e_{10}, e_{20}, \ldots, e_{r0}, D>}{D}$. Setting $e = D - < e_{10}, e_{20}, \ldots, e_{r0}, D >$ in Equation (4.60), we obtain that choosing $\pi_i = D - p_i$ for $i = 1, \ldots, r$ maximizes $\epsilon(\pi, \pi_0)$.

Minimizing the Coefficients of Integer Variables

In the following we concentrate on minimizing a coefficient of an integer-constrained variable. Firstly, we offer some remarks on solving diophantine equations of the form (4.61). Following Ceria et al. [46] we use the algorithm of Rosser [151] to solve such type of equations. The algorithm provides us with a family of solutions

$$(p_1, p_2, p_3, \dots, p_r, q) = P_1 + P_2 y_2 + P_3 y_3 + \dots + P_{r+1} y_{r+1}, \tag{4.63}$$

where y_k for $k = 2, \dots, r+1$ are arbitrary integers and $P_k \in \mathbb{Z}^{r+1}$ are vectors of the form

$$P_k = \left(p_1^k, p_2^k, p_3^k, \dots, p_r^k, p_{r+1}^k\right) \tag{4.64}$$

for $k = 1, \dots, r+1$. Suppose that the family (4.63) is a general solution to (4.61). Then for an arbitrary choice of the integers $y_2, \dots, y_{r+1}$ this solution yields multipliers maximizing $\epsilon(\pi, \pi_0)$. On the other hand, different coefficients are obtained on the left-hand side of the cut.

Accordingly, Ceria et al. propose to select a set of multipliers from the family (4.63) which gives the best coefficient on an integer variable while keeping the right-hand side maximal. Specifically, given an index $j \in J_I$ we wish to minimize

$$\sum_{i=1}^{r} \pi_i f_{ij} - \left\lfloor \sum_{i=1}^{r} \pi_i f_{ij} \right\rfloor, \tag{4.65}$$

where, as above, the values $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$ are the fractional parts of the coefficients of the tableau rows (see Equation (4.43)). The multipliers maximizing $\epsilon(\pi, \pi_0)$ are given by

$$\pi_i = D - p_i^1 - \sum_{k=2}^{r+1} p_i^k y_k, \quad i = 1, \dots, r. \tag{4.66}$$

To find a particular set of multipliers from this family that minimizes (4.65) for some $j \in J_I$ we insert the general solution (4.66) into (4.65). We have that

$$\sum_{i=1}^{r} \pi_i f_{ij} = \sum_{i=1}^{r} \left(D - p_i^1\right) f_{ij} + \sum_{k=2}^{r+1} \left(\sum_{i=1}^{r} -p_i^k f_{ij} \right) y_k \tag{4.67}$$

and can rewrite (4.65) as

$$g_0 + \sum_{k=2}^{r+1} g_k y_k - \left\lfloor g_0 + \sum_{k=2}^{r+1} g_k y_k \right\rfloor \tag{4.68}$$

with $g_0 = \sum_{i=1}^{r}(D - p_i^1)f_{ij} - \lfloor \sum_{i=1}^{r}(D - p_i^1)f_{ij} \rfloor$ and $g_k = \sum_{i=1}^{r}(-p_i^k f_{ij}) - \lfloor \sum_{i=1}^{r}(-p_i^k f_{ij}) \rfloor$. Recall that the integers p_i^k represent a specific family of solutions maximizing $\epsilon(\pi, \pi_0)$. Thus the only variables at this point are the integers y_k, $k = 2, \dots, r+1$, which need to be chosen such that the selected coefficient (4.68) is minimized. As we assumed that the rows of the simplex tableau are given in rational numbers, we can restate (4.68) as

$$\frac{e_0}{D} + \sum_{k=2}^{r+1}\left(\frac{e_k}{D}\right) y_k - \left\lfloor \frac{e_0}{D} + \sum_{k=2}^{r+1}\left(\frac{e_k}{D}\right) y_k \right\rfloor \tag{4.69}$$

with $g_0 = \frac{e_0}{D}$, $g_k = \frac{e_k}{D}$, $e_0 \in \mathbb{Z}$ and $e_k \in \mathbb{Z}$ for $k = 2, \dots, r+1$. Let $< e_2, e_3, \dots, e_{r+1}, D >$ denote the greatest common divisor of $e_2, e_3, \dots, e_{r+1}$ and D. Then there exist integers $p_2, p_3, \dots, p_{r+1}$ and q such that

$$< e_2, e_3, \dots, e_{r+1}, D >= e_2 p_2 + e_3 p_3 + \dots + e_{r+1} p_{r+1} + qD. \tag{4.70}$$

Rosser's algorithm again yields a general solution of the form (4.63) to the diophantine Equation (4.70).

Proposition 4.11 ([46]). *Suppose that $u = \frac{e_0}{<e_2, e_3, \dots, e_{r+1}, D>}$ is integral. Then, setting $y_k = -up_k$ yields a zero coefficient of the selected non-basic integer variable x_j in the strengthened intersection cut.*

Indeed with the proposed choice of y_k for $k = 2, \dots, r+1$ we obtain

$$\begin{aligned} \frac{e_0}{D} + \sum_{k=2}^{r+1}\left(\frac{e_k}{D}\right) y_k &= \frac{e_0}{D} - u \sum_{k=2}^{r+1} \frac{e_k p_k}{D}, \\ &= \frac{e_0}{D} - \left(\frac{e_0\,(< e_2, e_3, \dots, e_{r+1}, D > -qD)}{< e_2, e_3, \dots, e_{r+1}, D > D}\right), \\ &= uq, \end{aligned} \tag{4.71}$$

which is integral if u is integral. As a result, using these values for y_k in Equation (4.66) yields a family of multipliers which maximizes the right-hand side and minimizes the coefficient of the selected variable in the strengthened intersection cut.

In our presentation we interpreted the procedure of Ceria et al. [46] as a strengthening of the split disjunction $D(\pi, \pi_0)$. In terms of the rows of the simplex tableau associated with basic integer variables, the procedure selects a vector π of multipliers such that the GMI cut from the linear combination of these rows (cf. Proposition 4.8) is strong. Here strong means that it has a maximal right-hand side and minimal coefficients on some integer variables. We call the resulting cuts *combined Gomory mixed-integer (cGMI) cuts.*

The procedure outlined above does not, however, take into account the size of the coefficients of the continuous variables in the strengthened intersection cut or GMI cut. The size of these coefficients is directly dependent on the size of the entries in the vector π. Therefore large multipliers (or equivalently large coefficients in the split disjunction) are likely to produce cuts with weak (large) coefficients on the continuous variables. Again this has a negative influence on the distance cut off.

4.4.6. Reduce-and-Split Cuts

In the previous section we discussed a method that maximizes the right-hand side of the GMI cut and minimizes the coefficients on some integer variables. Even though the size of these coefficients is important, the strengthening of Proposition 4.5 in fact guarantees that any integer variable has a coefficient that is in the interval $[0, 1]$ in a strengthened intersection cut or GMI cut. However, as mentioned above, the coefficients of the continuous variables in a strengthened intersection cut as well as in a GMI cut are not bounded. The size of these coefficients is influenced by the underlying split disjunction and has a direct influence on the distance cut off. Andersen et al. [8] accordingly propose a method which reduces the size of these coefficients. This method is closely related to the basis reduction algorithm of Lenstra et al. [128].

We start our discussion by analyzing in more detail the influence of the split disjunction on the size of the coefficients of the continuous variables in an intersection cut. Suppose a basis B and the corresponding basic solution x^* are given. Let $D(\pi,\, \pi_0)$ be an arbitrary split disjunction which is violated. Let $\gamma x \geq 1$ be the strengthened intersection cut (4.40) generated from B and $D(\pi,\, \pi_0)$. The distance cut off by this cut is $1/\|\gamma\|$. Now observe that the size of a coefficient on a continuous variable x_j in this cut is dependent on the size of $|\pi r^j|$. Therefore the distance cut off can be increased by reducing the size of $|\pi r^j|$.

Andersen et al. [8] propose an algorithm which reduces the size of $|\pi r^j|$ by modifying the vector π (or the split disjunction $D(\pi,\, \pi_0)$ respectively). This algorithm works as follows. Consider an additional split disjunction $D(\pi',\, \pi'_0)$ distinct from $D(\pi,\, \pi_0)$. The algorithm replaces the disjunction $D(\pi,\, \pi_0)$ by the disjunction $D(\pi(\delta),\, \pi_0(\delta))$ where $\pi(\delta) = \pi + \delta\pi'$ and $\pi_0(\delta) = \lfloor \pi(\delta)x^* \rfloor$ with $\delta \in \mathbb{Z}$. The integer δ is chosen such that it minimizes the function

$$\begin{aligned} f(\delta) &= \sum_{j \in J \setminus J_I} \left(\pi\left(\delta\right) r^j\right)^2, \\ &= \sum_{j \in J \setminus J_I} \left(\left(\pi r^j\right)^2 + 2\delta\left(\pi r^j\right)\left(\pi' r^j\right) + \delta^2\left(\pi' r^j\right)^2\right). \end{aligned} \tag{4.72}$$

The function $f(\delta)$ measures nothing more than the squared Euclidean norm of the vector with entries $\pi(\delta)r^j$ for $j \in J \setminus J_I$. An alternative interpretation of $\pi(\delta)$ is the following. Combine the rows of the simplex tableau associated with basic integer variables with weights $\pi(\delta)$. Then $f(\delta)$ measures the squared Euclidean norm of this combined row on the continuous variables. But why do we measure the quality of the combined disjunction $D(\pi(\delta),\, \pi_0(\delta))$ by the squared Euclidean norm? Let $h(\pi, \pi') = \sum_{j \in J \setminus J_I}(\pi r^j)(\pi' r^j)$ and $g(\pi) = \sum_{j \in J\setminus J_I}(\pi r^j)^2$. Then the first derivative of $f(\delta)$ is given by $f'(\delta) = 2h(\pi, \pi') + 2\delta g(\pi')$ and the second derivative reads $f''(\delta) = 2g(\pi') > 0$. As $f(\delta)$ is a quadratic convex function in δ, its minimum can be found by rounding. More precisely, the optimal solution is either

$$\delta^* = -\left\lfloor \frac{h\left(\pi, \pi'\right)}{g\left(\pi'\right)} \right\rfloor \quad \text{or} \quad \delta^* = -\left\lceil \frac{h\left(\pi, \pi'\right)}{g\left(\pi'\right)} \right\rceil. \tag{4.73}$$

If $f(\delta^*) < f(0)$, then the disjunction $D(\pi,\, \pi_0)$ is replaced by the disjunction $D(\pi(\delta^*),\, \pi_0(\delta^*))$ and the process is iterated. Andersen et al. [8] call the resulting cuts *reduce-and-split (R&S) cuts.* The above procedure is started from the elementary split disjunctions (4.44).

This procedure can also be carried on the rows of the simplex tableau (4.43). In each iteration a pair of rows associated with two basic integer-constrained variables, say x_i and x_k, is selected and the linear combination

$$x_i + \delta x_k = \bar{a}_{i0} + \delta \bar{a}_{k0} - \sum_{j \in J} \left(\bar{a}_{ij} + \delta \bar{a}_{kj}\right) x_j \tag{4.74}$$

with $\delta \in \mathbb{Z}$ is considered. Note that we intend to replace the original rows of the simplex tableau (4.43) by combined rows of the form (4.74). Thus in subsequent iterations of the algorithm a row of the simplex tableau originally associated with the basic variable x_i may contain several other basic integer variables. Nevertheless, the variable $x_i' = x_i + \delta x_k$ is also integer-constrained. The effect that using the multiplier δ has on the size of the coefficients on the continuous variables in the combined row is measured by (4.72). Specifically, we can also write $f(\delta)$ as

$$f(\delta) = \sum_{j \in J \setminus J_I} \left(\bar{a}_{ij} + \delta \bar{a}_{kj}\right)^2 . \tag{4.75}$$

The optimal value of δ minimizing $f(\delta)$ is then given by

$$\delta^* = - \left\lfloor \frac{\sum\limits_{j \in J \setminus J_I} \bar{a}_{ij} \bar{a}_{kj}}{\sum\limits_{j \in J \setminus J_I} \bar{a}_{kj}^2} \right\rfloor \quad \text{or} \quad \delta^* = - \left\lceil \frac{\sum\limits_{j \in J \setminus J_I} \bar{a}_{ij} \bar{a}_{kj}}{\sum\limits_{j \in J \setminus J_I} \bar{a}_{kj}^2} \right\rceil . \tag{4.76}$$

We have identified a reduction, if the inequality $f(\delta^*) < f(0)$ holds. Then the linear combination (4.74) is calculated with the optimal multiplier δ^*. The combined row then replaces the original row and the process is iterated.

In a recent paper Cornuéjols and Nannicini [64] propose to select a certain subset of the continuous non-basic variables and to use the reduce-and-split approach to reduce the coefficients of these variables.

4.4.7. Lift-and-Project Cuts

So far we have discussed three approaches to improving the performance of the strengthened intersection (or GMI) cut. What all of these approaches have in common is that they modify the elementary split disjunctions (4.44) on basic integer variables. The strengthened intersection cut from the basis B and this improved disjunction is then generated. However, each approach aims at optimizing different characteristics of the strengthened intersection cut, be it the size of the coefficients in the cut or its violation. In this section we consider a procedure which finds the best basis B (in some sense to be discussed later) from which the intersection cut is generated while leaving the underlying elementary disjunction unchanged.

Firstly, we introduce some additional notation. Reconsider the MIP (2.1) and suppose that all integer-constrained variables are 0-1 variables, i.e. we have a (mixed) 0-1 program. Let N be the set of variables and $N_I = \{1, \dots, p\} \subseteq N$ be the set of 0-1 variables. For technical reasons we assume that the constraint system of our mixed 0-1 program explicitly contains the simple upper bounds on the 0-1 variables. We assume in addition that the lower bound constraints on all structural variables are as well explicitly present as constraints. More formally, we consider the mixed 0-1 program

$$\text{(MBP)} \quad \min \left\{ cx : x \in X_{LP}, x_j \in \{0,1\}, j = 1, \dots, p \right\}, \tag{4.77}$$

where the set X_{LP} is defined as

$$X_{LP} = \left\{ x \in \mathbb{R}^n : \begin{array}{rcll} Ax & \geq & b & \\ -x_j & \geq & -1, & j = 1, \dots, p \\ x & \geq & 0 & \end{array} \right\}. \tag{4.78}$$

The LP relaxation of (4.77) is given by

$$\text{(LP)} \quad \min \left\{ cx : x \in X_{LP} \right\}. \tag{4.79}$$

Typically, MIP solvers store the bounds of the variables separate from the constraint matrix. We shall discuss this and other practical issues in Part III of this thesis (see Section 8.5). As before, we denote by $P = X_{LP}$ the polyhedron associated with (LP). We shall denote the whole system defining the polyhedron P by $\tilde{A}x \geq \tilde{b}$.

The system $\tilde{A}x \geq \tilde{b}$ consists of $m+p+n$ rows and can be written as $\tilde{A}x - s = \tilde{b}$ by introducing surplus variables. Note that the vector $s \in \mathbb{R}^{m+p+n}$ consists of m

surplus variables from the constraints in A, p surplus variables from the upper bound constraints, and n surplus variables from the lower bound constraints. As all bounds on the 0-1 variables are contained in the constraint system $\tilde{A}x \geq \tilde{b}$, all of these variables are unrestricted and can without loss of generality be assumed to be basic. Since $s_{m+p+j} = x_j$ for $j = 1, \ldots, n$, it is possible to write the rows of the simplex tableau completely using only surplus variables

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j, \quad i \in B_I. \tag{4.80}$$

With respect to notation, also note the following. We use (x^*, s^*) to denote an optimal basic solution to (LP). On the other hand, concerning the simplex tableau associated with the current basic solution (x, s) to (LP), we denote by $\bar{a}_{ij}$ and $\bar{a}_{i0}$ the coefficient of variable j in row i and the right-hand side of row i respectively.

Lift-and-project (L&P) cuts [24, 25] are disjunctive cuts which are derived from a disjunction of the form

$$\begin{pmatrix} \tilde{A}x \geq \tilde{b} \\ -x_i \geq 0 \end{pmatrix} \vee \begin{pmatrix} \tilde{A}x \geq \tilde{b} \\ x_i \geq 1 \end{pmatrix} \tag{4.81}$$

on a fractional 0-1 variable. Further developments of the method were documented in [20, 29, 149]. The most-violated (deepest) L&P cut $\alpha x \geq \beta$ from the disjunction (4.81) is obtained by solving the cut generating linear program

$$\begin{array}{llllllll}
(\mathrm{CGLP}_i) & \min & \alpha x^* - \beta, & & & & & \\
 & \text{s. t.} & \alpha & & - u\tilde{A} & + u_0 e_i & & = 0, \\
 & & \alpha & & & - v\tilde{A} & - v_0 e_i & = 0, \\
 & & & - \beta & + u\tilde{b} & & & = 0, \\
 & & & - \beta & & + v\tilde{b} & + v_0 & = 0,
\end{array} \tag{4.82}$$

where $u, v, u_0, v_0 \geq 0$. Recall that the set of feasible solutions to (CGLP$_i$) is a cone which needs to be truncated by a normalization constraint

$$\sum_{i=1}^{m+p+n} (u_i + v_i) + u_0 + v_0 = 1 \tag{4.83}$$

in order to obtain a bounded set. The linear program (CGLP$_i$) is a special case of the general CGLP (2.24) for disjunctive programs.

The cut generating linear program (CGLP$_i$) is large. Specifically, it consists of $2n + 3$ rows and $2(m + p) + 3(n + 1)$ columns. Moreover, it can be shown to be highly-degenerate. Thus solving (CGLP$_i$) may be too expensive from a computational point of view.

Note that Proposition 2.3 tells us that the first two constraints in (CGLP$_i$) defining α should be $\geq$-inequalities. However, the identity matrix (lower bound constraints) which we added to A serves as a vector of surplus variables for these constraints. Similarly, the last two constraints in (CGLP$_i$) could be relaxed to $\geq$-inequalities. As the constraints $x_j \geq 0$ and $-x_j \geq -1$ are contained in the system $\tilde{A}x \geq \tilde{b}$, the trivial inequality $0x \geq -1$ is also implicitly present in this system. We can therefore require that the inequalities defining β hold at equality.

While the most-violated L&P cut is found by solving (CGLP$_i$) to optimality, any solution to (CGLP$_i$) yields an L&P cut. Let $\tilde{A}_j$ be the j^{th} column of $\tilde{A}$. Furthermore, let $\alpha_j^1 = u\tilde{A}_j - u_{m+p+j}$ and $\alpha_j^2 = v\tilde{A}_j - v_{m+p+j}$ for $j = 1, \ldots, n$. Then the L&P cut $\alpha x \geq \beta$ associated with the basic solution $(\alpha, \beta, u, v, u_0, v_0)$ of (CGLP$_i$) is given by

$$\alpha_j = \begin{cases} \max\{\alpha_j^1 - u_0, \alpha_j^2 + v_0\} & \text{if } j = i, \\ \max\{\alpha_j^1, \alpha_j^2\} & \text{if } j \neq i, \end{cases} \tag{4.84}$$

and $\beta = u\tilde{b} = v\tilde{b} + v_0$.

L&P cuts can be strengthened by considering the integrality of some of the variables. The *strengthened L&P cut* $\bar{\alpha}x \geq \beta$ has the coefficients

$$\bar{\alpha}_j = \begin{cases} \min\{\alpha_j^1 + u_0 \lceil m_j \rceil, \alpha_j^2 - v_0 \lfloor m_j \rfloor\} & \text{if } j \in \{1, \ldots, p\} \setminus \{i\}, \\ \alpha_j & \text{otherwise,} \end{cases} \tag{4.85}$$

where

$$m_j = \frac{\alpha_j^2 - \alpha_j^1}{u_0 + v_0}. \tag{4.86}$$

The strengthened L&P cut can also be obtained by replacing the split disjunction $-x_i \geq 0$ or $x_i \geq 1$ by the more general split disjunction

$$\left(-\sum_{j=1}^{p} \pi_j x_j \geq 0\right) \vee \left(\sum_{j=1}^{p} \pi_j x_j \geq 1\right). \tag{4.87}$$

The vector $\pi \in \mathbb{Z}^p$ producing the best (smallest) coefficients in the L&P cut can be shown [25] to be given by $\pi_j = \lfloor m_j \rfloor$ or $\pi_j = \lceil m_j \rceil$ for $j = 1, \dots, p$.

The foundations for a more efficient separation of L&P cuts were laid by Balas and Perregaard [30]. Given a basis B of (LP), Balas and Perregaard show that the intersection cut generated from the tableau row (4.80) associated with the basic fractional integer variable x_i is equivalent to an L&P cut from a particular basis of (CGLP$_i$). Moreover, they show that the correspondence between bases of (LP) and (CGLP$_i$) is well defined. This means that a basis of (LP) can be constructed such that it gives an intersection cut which is equivalent to a specific L&P cut. The following section details this correspondence.

A Precise Correspondence

Consider a feasible basic solution $(\alpha, \beta, u, v, u_0, v_0)$ of (CGLP$_i$) which yields the L&P cut $\alpha x \geq \beta$ and the strengthened L&P cut $\bar{\alpha} x \geq \beta$. We suppose that $u_0 > 0$ and $v_0 > 0$ since otherwise the L&P cut is just a non-negative linear combination of the constraints of $\tilde{A}x \geq \tilde{b}$. Let the sets M_1 and M_2 contain the indices of the basic components of u and v respectively. Moreover, let $J = M_1 \cup M_2$ and $\tilde{A}_J$ be the matrix which consists of the rows of $\tilde{A}$ indexed by J. It can be verified that $M_1 \cap M_2 = \emptyset$ and $|M_1 \cup M_2| = n$. Therefore $\tilde{A}_J$ is a $n \times n$ square matrix which is invertible [30]. With this notation we can write

$$\begin{aligned} \tilde{A}_J x &= \tilde{b}_J + s_J, \\ x &= \tilde{A}_J^{-1} \tilde{b}_J + \tilde{A}_J^{-1} s_J. \end{aligned} \tag{4.88}$$

Then the row associated with the basic variable x_i can be written as

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j, \tag{4.89}$$

where $\bar{a}_{i0} = (\tilde{A}_J^{-1} \tilde{b}_J)_i$ and $\bar{a}_{ij} = -(\tilde{A}_J^{-1})_{ij}$. This row of the simplex tableau is the same as (4.80). It can furthermore be shown that $0 < \bar{a}_{i0} < 1$ due to the assumption that $u_0 > 0$ and $v_0 > 0$. Thus there is a correspondence between the basic components of u and v in (CGLP$_i$) and the non-basic components of the surplus variables s in (LP). Conversely, the non-basic components of u and v are connected to basic components of s or x respectively.

Theorem 4.12 ([30])**.** *The strengthened L&P cut $\bar{\alpha} x \geq \beta$ is equivalent to the GMI cut generated from* (4.89)*.*

On the other hand, this correspondence can be used to construct a basic feasible solution of (CGLP$_i$) such that the strengthened L&P cut is equivalent to the GMI cut from (4.89). Suppose that a row of the simplex tableau (4.89) associated with a basis B of (LP) and a basic variable x_i is given such that $0 < \bar{a}_{i0} < 1$. Note that the basis B does not have to be optimal or feasible.

Theorem 4.13 ([30])**.** *Let (M_1, M_2) be a partition of J such that $j \in M_1$, if $\bar{a}_{ij} < 0$ and $j \in M_2$, if $\bar{a}_{ij} > 0$. Then the strengthened L&P cut $\bar{\alpha} x \geq \beta$ which is defined by the solution to* (CGLP$_i$) *associated with the basis*

$$(\alpha, \beta, u_0, v_0, \{u_k : k \in M_1\}, \{v_k : k \in M_2\}) \tag{4.90}$$

is equivalent to the GMI cut derived from (4.89)*.*

Theorem 4.13 is based on the partition (M_1, M_2) of the non-basic variables J. But if there are any non-basic variables with $\bar{a}_{ij} = 0$, this partition is not unique. In this case we are free to assign the variable either to M_1 or M_2. Therefore Theorem 4.13 relates each basis B of (LP) to a number of different bases of (CGLP$_i$). However, these bases are degenerate and correspond to the same basic solution of (CGLP$_i$). So there is a one-to-one relation between basic solutions of (LP) and (CGLP$_i$). The correspondence stated in Theorem 4.12 and Theorem 4.13 can also be established between the unstrengthened L&P cut and the intersection cut.

Solving the CGLP on the LP Tableau

Based on these insights, Balas and Perregaard [30] developed a very elegant method which mimics the optimization of (CGLP$_i$) by performing a sequence of pivots on the original (LP) tableau. In each iteration of this procedure a pivot in a row of the simplex tableau associated with the basic variable x_k with $k \neq i$ is performed. This pivot produces a linear combination

$$x_i + \gamma x_k = \bar{a}_{i0} + \gamma \bar{a}_{k0} - \sum_{j \in J} \left(\bar{a}_{ij} + \gamma \bar{a}_{kj} \right) s_j \tag{4.91}$$

of our reference row (4.89) associated with x_i and the selected row. The procedure aims at selecting the pivot such that the GMI cut from the combined row (4.91) is more violated than that obtained from the original row. Let $\gamma = -\frac{\bar{a}_{ip}}{\bar{a}_{kp}}$ for some $p \in J$. The pivot in Equation (4.91) then makes the basic variable x_k leave the basis and the non-basic variable x_p enter the basis. The variable x_k is also called the pivot row and the variable x_p is called the pivot column. To guide the search for an improving pivot, the correspondence between bases of (LP) and (CGLP$_i$) is used.

In order to be able to perform a pivot, a variable which leaves the basis needs to be selected in a first step. This selection is guided by the fact that each basic variable x_k of the (LP) simplex tableau corresponds to a pair u_k, v_k of non-basic variables of (CGLP$_i$). The reduced cost of the non-basic variables u_k and v_k can be calculated from the entries in the (LP) tableau rows associated with x_i and x_k and the solution vector x^* for each row $k \notin J \cup \{i\}$

$$r_{u_k} = -\sigma + \bar{a}_{k0} \left(1 - x_i^* \right) - \tau_k, \tag{4.92a}$$

$$r_{v_k} = -\sigma - \bar{a}_{k0} \left(1 - x_i^* \right) + s_k^* + \tau_k, \tag{4.92b}$$

where

$$\sigma = \frac{\sum\limits_{j \in M_2} \bar{a}_{ij} s_j^* - \bar{a}_{i0} \left(1 - x_i^* \right)}{1 + \sum\limits_{j \in J} |\bar{a}_{ij}|}, \tag{4.93}$$

and

$$\tau_k = \sum_{j \in M_1} \sigma \bar{a}_{kj} + \sum_{j \in M_2} \left(s_j^* - \sigma \right) \bar{a}_{kj}. \tag{4.94}$$

If one of the reduced cost r_{u_k} or r_{v_k} is negative, the L&P cut can be improved by pivoting the non-basic variable u_k or v_k respectively into the basis of (CGLP$_i$). In terms of the (LP) tableau, the equivalent effect can be achieved by pivoting the basic variable x_k out of the basis. If none of the variables u_k or v_k with $k \notin J \cup \{i\}$ has negative reduced cost, there is no improving pivot and the current basis of (LP) corresponds to an optimal basis of (CGLP$_i$). In this case, the GMI cut from the row (4.89) of the simplex tableau associated with the current basis is equivalent to the optimal L&P cut.

In a second step a non-basic variable x_p from the row associated with the basic variable x_k is selected to enter the basis. The two functions

$$f^+(\gamma) = \frac{\sum\limits_{j \in J} \max\{\bar{a}_{ij}, -\gamma \bar{a}_{kj}\} s_j^* - \bar{a}_{i0} + (\bar{a}_{i0} + \gamma \bar{a}_{k0}) x_i^*}{1 + \gamma + \sum\limits_{j \in J} |\bar{a}_{ij} + \gamma \bar{a}_{kj}|} \tag{4.95a}$$

and

$$f^-(\gamma) = \frac{\sum\limits_{j \in J} \max\{0, \bar{a}_{ij} + \gamma \bar{a}_{kj}\} s_j^* - (\bar{a}_{i0} + \gamma \bar{a}_{k0})(1 - x_i^*)}{1 - \gamma + \sum\limits_{j \in J} |\bar{a}_{ij} + \gamma \bar{a}_{kj}|} \tag{4.95b}$$

are used to measure the effect that pivoting x_p into the basis of (LP) has on the value of the objective function of (CGLP$_i$). The functions $f^+(\gamma)$ and $f^-(\gamma)$ are minimized to identify the entering variable x_p which brings about the largest improvement of the violation of the L&P cut. If a leaving and an entering variable are selected, the corresponding pivot is performed and the process is iterated. For recent computational studies of different variants of this algorithm see [22, 23].

Connection to Split Cuts

In this section we discuss the connection of L&P cuts and split cuts. Multiplying the left-hand side of the disjunction (4.81) with $u, u_0 \geq 0$ and the right-hand side with $v, v_0 \geq 0$ we obtain

$$\left(u\tilde{A}x - u_0 x_i \geq u\tilde{b}\right) \vee \left(v\tilde{A}x + v_0 x_i \geq v\tilde{b} + v_0\right). \tag{4.96}$$

Solving (CGLP_i) optimizes the multipliers u, v, u_0, v_0 and generates the most-violated intersection (or simple disjunctive) cut with respect to the elementary split disjunction $x_i \leq 0$ or $x_i \geq 1$. Using the correspondence discussed above, an equivalent result can be obtained by performing pivots on the (LP) tableau. The intersection cut can then be strengthened by using the integrality conditions on the non-basic integer variables (cf. Proposition 4.5). This second operation can be seen as a strengthening of the underlying disjunction (cf. Proposition 4.6). Therefore the most-violated strengthened intersection cut (or GMI cut) is the result of a two-stage procedure. One would like ideally to optimize the basis and the disjunction at the same time. This is equivalent to finding an optimal split cut which can be separated by solving a mixed-integer non-linear program (see Section 4.4.1).

The Role of the Normalization

Fischetti et al. [87] examine the strengths and weaknesses of the standard normalization constraint (4.83). This normalization has several positive characteristics. To see this, consider the right-hand side of the normalization constraint to be a resource that must be shared among the multipliers u, v, u_0, and v_0. As a consequence large multipliers are generally undesirable as they consume large amounts of this resource. Therefore the standard normalization (4.83) will produce cuts with relatively small coefficients due to the usage of relatively small multipliers. This in turn implies that multipliers associated with cuts need to be comparably large for the cuts to become relevant. Thus cuts with relatively low rank are separated. Since original inequalities from the problem formulation are normally sparse and the normalization produces relatively sparse multiplier vectors, the generated cuts also tend to be sparse.

A weakness of the standard normalization is that it is dependent on the scaling of the constraint system. Consider an inequality $ax \geq b$ and its scaled version $a'x \geq b'$ where $a' = \mu a$ and $b' = \mu b$ with $\mu > 1$. The multipliers of the second inequality are $u' = \frac{u}{\mu}$ and $v' = \frac{v}{\mu}$. Selecting the second inequality is therefore more favorable as it consumes fewer resources with respect to the right-hand side of the normalization constraint. By an appropriate scaling previously generated cuts can also become relevant. Thus the nice properties of the normalization, i.e. the generation of sparse low-rank cuts, are lost. Constraints that become redundant in (CGLP$_i$) due to the disjunction used pose an additional problem.

To overcome the discussed drawbacks, Fischetti et al. [87] propose the Euclidean normalization

$$\sum_{i=1}^{m+p+n} \|\tilde{a}_i\| (u_i + v_i) + u_0 + v_0 = 1, \tag{4.97}$$

where $\tilde{a}_i$ is the i^{th} row of $\tilde{A}$ and $\|\cdot\|$ denotes the Euclidean norm. This approach is equivalent to scaling the system $\tilde{A}x \geq \tilde{b}$ such that every row of $\tilde{A}$ has Euclidean norm equal to 1. Clearly the Euclidean normalization is not affected by scaling.

Balas and Bonami [23] take up the ideas of Fischetti et al. and study the normalization constraint

$$\sum_{i=1}^{m+p+n} \lambda_i (u_i + v_i) + u_0 + v_0 = \lambda_0, \tag{4.98}$$

where $\lambda_i \geq 0$ for $i = 1, \ldots, m+p+n$ and λ_0 is a positive integer. Introducing the new normalization (4.98) into (CGLP$_i$) is easy. Moreover, the correspondence between bases of (LP) and (CGLP$_i$) which is stated in Theorem 4.12 and 4.13 is not affected by modifying the normalization. However, the algorithm optimizing (CGLP$_i$) by pivoting on the original (LP) tableau needs to be adapted. Specifically, the calculation of the reduced cost (4.92) and the evaluation functions (4.95) is based on the assumption that the standard normalization $ve+ue+v_0+u_0 = 1$ is used. When the normalization (4.98) is used the expressions of the reduced cost are given by

$$r_{u_k} = -\sigma\lambda_k + \bar{a}_{k0}(1 - x_i^*) - \tau_k, \tag{4.99a}$$

$$r_{v_k} = -\sigma\lambda_k - \bar{a}_{k0}(1 - x_i^*) + s_k^* + \tau_k, \tag{4.99b}$$

where

$$\sigma = \frac{\sum_{j\in M_2} \bar{a}_{ij} s_j^* - \bar{a}_{i0}(1 - x_i^*)}{1 + \sum_{j\in J} |\bar{a}_{ij}| \lambda_j}, \tag{4.100}$$

and

$$\tau_k = \sum_{j\in M_1} \sigma \bar{a}_{kj}\lambda_j + \sum_{j\in M_2} \left(s_j^* - \sigma\lambda_j\right) \bar{a}_{kj}. \tag{4.101}$$

In addition, the two evaluation functions (4.95) have to be slightly modified:

$$f^{+}(\gamma) = \frac{\left(\sum_{j\in J} \max\left\{\bar{a}_{ij}, -\gamma\bar{a}_{kj}\right\} s_j^* - \bar{a}_{i0} + \left(\bar{a}_{i0} + \gamma\bar{a}_{k0}\right) x_i^*\right)\lambda_0}{1 + \gamma\lambda_k + \sum_{j\in J} \left|\bar{a}_{ij} + \gamma\bar{a}_{kj}\right| \lambda_j}, \tag{4.102a}$$

$$f^{-}(\gamma) = \frac{\left(\sum_{j\in J} \max\left\{0, \bar{a}_{ij} + \gamma\bar{a}_{kj}\right\} s_j^* - \left(\bar{a}_{i0} + \gamma\bar{a}_{k0}\right)\left(1 - x_i^*\right)\right)\lambda_0}{1 - \gamma\lambda_k + \sum_{j\in J} \left|\bar{a}_{ij} + \gamma\bar{a}_{kj}\right| \lambda_j}. \tag{4.102b}$$

Chapter 5.

Multi-Row Cutting Planes

In this chapter we consider cutting planes which are derived using multiple rows of the simplex tableau simultaneously. We give a brief review of the relevant literature on this subject. We also discuss group relaxations in more detail and elaborate on the connection between valid inequalities and lattice-free convex sets.

This chapter is organized as follows. Section 5.1 introduces multi-row relaxations and Section 5.2 presents a literature review. In Section 5.3 we treat the (master) group relaxation. The derivation of valid inequalities is discussed in Section 5.4. In particular we deal with the generation of valid inequalities from two rows of the simplex tableau.

5.1. Introduction

In the previous chapter we were concerned with the generation of cutting planes from single-row relaxations. These relaxations are obtained by aggregating the constraints of an MIP. The resulting single constraint is then used as the input data for a cutting plane separation algorithm. In particular, any row of the simplex tableau can be seen as a single-row relaxation. In this chapter, however, we concentrate on generating cutting planes using more than one row of the simplex tableau at the same time. This approach is not new. The fundamental results on which it is based were discovered more than 40 years ago. However, recently cutting planes from multiple rows have been revisited and new interesting theoretical results have been proposed. The new impetus to the field of multi-row cuts is mainly due to the constant need for strong cutting planes and the limitations of the single-row cuts.

Once again, let B be a basis of the LP relaxation of the MIP (2.3). As before, let the set J index the non-basic variables and let x^* be the basic solution associated with B. Using the extreme rays defined in Equation (4.25), we can write

$$\begin{aligned} x &= x^* + \sum_{j\in J} r^j x_j, \\ x &\geq 0, \\ x_j &\in \mathbb{Z}, \quad j \in N_I. \end{aligned} \tag{5.1}$$

Let x_B and x_J denote the basic and non-basic components of x respectively. We consider the simplex tableau

$$\begin{aligned} x_B &= x_B^* + \sum_{j\in J} r^j x_j, \\ x &\geq 0, \\ x_j &\in \mathbb{Z}, \quad j \in N_I. \end{aligned} \tag{5.2}$$

We assume that the basic solution $(x_B, x_J) = (x_B^*, 0)$ is integer infeasible. The simplex tableau is nothing more than a reformulation of the MIP (2.3) in the sense that every row of the tableau is a linear combination of the original rows of the system (A, b). Note that we slightly abuse notation by denoting the columns of the simplex tableau by r^j. Here and in what follows the vectors r^j only contain the components of the extreme rays (4.25) that are associated with the selected vector of basic variables.

A Gomory mixed-integer cut is generated from a single row of the simplex tableau (see Section 4.4.3). This single row provides only very limited information about the structure of the underlying MIP. An interesting question is thus whether the additional information provided by the remaining rows of the simplex tableau can be used to construct stronger (or at least different) valid inequalities.

The following section presents a brief literature review. Recent surveys on multi-row cutting planes are given by Conforti et al. [55] and Dey and Tramontani [77]. An excellent review of the group theoretic approach in integer programming is given by Richard and Dey [150].

5.2. Literature Review

In 1969 Gomory [101] introduced the corner relaxation (or group relaxation) of an integer program which is obtained by dropping the non-negativity restrictions on the basic variables. The convex hull of feasible solutions to this relaxation is called the corner polyhedron. Moreover, Gomory studied the master corner relaxation (or master group relaxation) and the associated master polyhedron. These polyhedra can be seen as a data-independent generalization of corner polyhedra.

Different algorithms for solving the corner relaxation have been proposed. Gomory [100] discusses a dynamic programming algorithm for optimizing a linear function over the corner relaxation. Other algorithms solving the corner relaxation are presented in [49, 95, 114, 153, 154]. In [101] Gomory also proved the asymptotic theorem. This theorem gives necessary conditions under which the optimal solution to the corner relaxation is equal to the optimal solution to the original integer program. In a computational study Gorry et al. [105] showed that for most real-life integer programs the (asymptotic) corner relaxation does not solve the original integer program.

Valid inequalities for the (master) corner relaxation are also valid for the original integer program. It therefore seems reasonable to derive valid inequalities for the group relaxation (group inequalities) and apply them to integer programs. Gomory and Johnson [102, 103] studied group inequalities and particularly concentrated on the facets of master polyhedra. Gomory [101] showed that the Gomory fractional cut [97] can also be derived as a facet of the master polyhedron. Similarly, the Gomory mixed-integer cut [99] is a facet of the mixed-integer extension of the master polyhedron (see [102]). Both of these inequalities are one-dimensional group inequalities, i.e. they are based on a group relaxation that only consists of a single row. Since the GMI cut is very effective from a computational point of view [26], several attempts to find other effective single-row group inequalities have been made. For instance, Dash et al. [70] proposed two-step MIR cuts and Kianfar and Fathi [119] introduced the more general class of n-step MIR cuts. However, none of these variants has proved to outperform the GMI cuts in solving practical (mixed-) integer programs (see also [72]). Dey and Wolsey [79] suppose that this is because the GMI cut has the strongest coefficients on the continuous variables among all single-row group inequalities.

Johnson [116] studied inequalities which are based on multi-row group relaxations. These inequalities are known to be important to describe the convex hull of (mixed-) integer programs. For example, Cook et al. [57] present a simple mixed-integer set whose convex hull cannot be obtained using split cuts while a single multi-row cut yields the convex hull. Gomory and Johnson [104] point out that multi-row inequalities are able to reflect the structure of the columns associated with continuous variables more accurately. For these reasons there has been a renewed interest in multi-row cuts. Borozan and Cornuéjols [41] consider a semi-infinite relaxation and establish a connection between minimal valid inequalities for this relaxation and maximal lattice-free convex sets. Andersen et al. [11] and Cornuéjols and Margot [63] study cutting planes from two rows. Dey and Wolsey [78, 79] address the lifting of non-basic integer variables in two-row cuts, i.e. they show that two-row cuts can be strengthened using the integrality of some of the non-basic variables (see also Conforti et al. [54] and Basu et al. [34]). Andersen et al. [10] demonstrate that stronger two-row cuts can be obtained by considering the bounds on the non-basic variables. Dey and Wolsey [80] study S-free cuts which are generated from multi-row relaxations containing additional constraints (e.g. bounds) on the basic integer variables (see also Basu et al. [35] and Fukasawa and Günlük [92]). Basu et al. [33] compare the strength of the elementary closures of different families of two-row cuts (see also Andersen et al. [12]). He et al. [112] provide a probabilistic comparison of two-row cuts and split cuts. Dey and Louveaux [76] and Basu et al. [36] study the split rank of multi-row cuts. Espinoza [83, 84], Basu et al. [32] and Dey et al. [75] report on computational experience with multi-row cuts.

5.3. Group Relaxations

In this section we investigate multi-row and group relaxations in more detail. Our presentation here is partly based on [150].

We start by considering a relaxation of (5.2) which was first studied by Gomory [101]. We only consider the rows of the simplex tableau associated with basic integer variables. For simplicity we assume that all basic variables are integer-

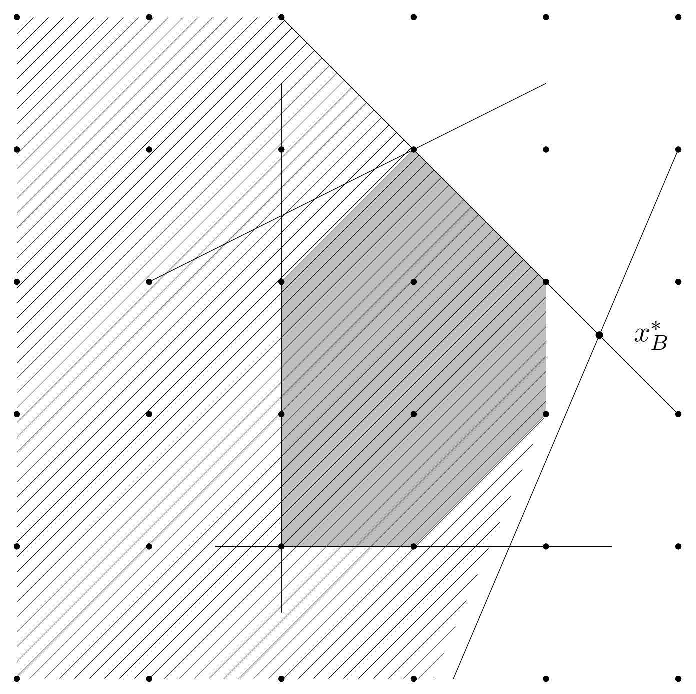


Figure 5.1. Corner polyhedron

constrained, i.e. $B = B_I = \{1, \ldots, m\}$. Moreover, we drop the non-negativity conditions on all basic variables. We get

$$\begin{aligned} x_B &= x_B^* + \sum_{j \in J} r^j x_j, \\ x_j &\geq 0, \quad j \in J, \\ x_j &\in \mathbb{Z}, \quad j \in N_I. \end{aligned} \tag{5.3}$$

The system (5.3) is known as the *corner relaxation.* The convex hull of the feasible solutions to the corner relaxation is known as Gomory's *corner polyhedron* (see [101]). Note that when the polyhedron P (or the simplex tableau respectively) is non-degenerate then the non-negativity restrictions on the basic variables are the only non-binding constraints. Valid inequalities for the corner relaxation are also valid for (5.2) and thus for the MIP (2.3). Figure 5.1 shows an integer program and its convex hull of feasible solutions (gray area). The hatched area which continues off the figure is the corner polyhedron associated with the basic solution x^*_B.

We now examine the rows of the simplex tableau defining the corner relaxation (5.3) from a different viewpoint. Let f be a vector with $f_i = (x^*_B)_i - \lfloor (x^*_B)_i \rfloor$ for $i = 1, \ldots, m$. To obtain a feasible integral vector x_B to the corner relaxation (which is not necessarily non-negative) the fractional part of the sum $\sum_{j \in J} r^j x_j$ has to be $1 - f$. In other words, the sum $\sum_{j \in J} r^j x_j$ has to add up to $-f$ modulo 1. This can be stated by the following congruence.

$$\begin{aligned} \sum_{j \in J_I} r^j x_j + \sum_{j \in J \setminus J_I} r^j x_j \;&\equiv\; -f \,(\bmod 1)\,, \\ x_j \;&\geq\; 0, \quad j \in J, \\ x_j \;&\in\; \mathbb{Z}, \quad j \in N_I. \end{aligned} \tag{5.4}$$

Corner polyhedra are closely related to groups. Specifically, the problem (5.4) can be interpreted as finding a finite sum of group elements such that they add up to a group element that gives an integral vector x_B. The corner relaxation is thus often called the *group relaxation.* For the non-basic integer variables the fractional part of the product $r^j_i x_j$ is just an integer multiple of the fractional part of r^j_i for $i = 1, \ldots, m$. We can therefore replace the column r^j of the tableau by the vector of its fractional components for all $j \in J_I$. For the sake of simplicity, we also denote this vector by r^j.

The structure of the corner relaxation depends heavily on the underlying mixed-integer program, i.e. the columns r^j. Therefore an analysis of the corner relaxation yields very problem specific results. A different idea is to abstract from the specific system (5.3) by introducing additional variables. Let G and W be two sets which contain the columns associated with the integer-constrained and continuous variables respectively, i.e. $\{r^j : j \in J_I\} \subseteq G$ and $\{r^j : j \in J \setminus J_I\} \subseteq W$. We obtain the so-called *master corner relaxation* or *master group relaxation*

$$\begin{aligned} x_B &= f + \sum_{g \in G} g \cdot t\,(g) + \sum_{w \in W} w \cdot s\,(w)\,, \\ x_B &\in \mathbb{Z}^m, \\ t(g) &\in \mathbb{Z}, \quad g \in G, \\ t(g), s(w) &\geq 0, \quad g \in G, \quad w \in W, \\ & t, s \text{ have finite support.} \end{aligned} \tag{5.5}$$

The system (5.5) contains extra variables for the vectors $g \in G$ or $w \in W$ which are not columns of the original simplex tableau. By fixing the variables associated with these new columns to zero we again obtain the corner relaxation (5.3). Thus every feasible solution to the corner relaxation (5.3) can be translated into a solution of (5.5). Note that we require the vectors t and s to have finite support in a feasible solution to (5.5), meaning that these vectors only have a finite number of non-zero components. We thereby ensure that the sums in the first equation of (5.5) are well-defined.

Gomory and Johnson [103] and Johnson [116] considered the case in which the set G is a group. Let I^m be the group of m-dimensional vectors $[0, 1[^m$ where addition is taken modulo 1 component-wise. Further let S^m be the set of m-dimensional real vectors $w = (w_1, w_2, \ldots, w_m)$ satisfying $\max\{|w_i| : i = 1, \ldots, m\} = 1$. Suppose that G is a subgroup of I^m and that W is a subset of S^m, i.e. $W \subseteq S^m$. As above, the group G and the set W contain the columns of the integer and continuous variables respectively, i.e. $\{r^j : j \in J_I\} \subseteq G$ and $\{r^j : j \in J \setminus J_I\} \subseteq W$. Given a vector $r \in \mathbb{R}^m$, let $\mathcal{F}(r)$ denote the vector in I^m whose i^{th} component is $r_i (\operatorname{mod} 1)$. We obtain

$$\begin{aligned} \sum_{g \in G} g \cdot t\,(g) + \mathcal{F}\left(\sum_{w \in W} w \cdot s\,(w)\right) &\equiv \mathcal{F}\,(-f)\,(\operatorname{mod} 1)\,, \\ t(g) &\in \mathbb{Z}, \quad g \in G, \\ t(g), s(w) &\geq 0, \quad g \in G, \quad w \in W, \\ & t, s \text{ have finite support.} \end{aligned} \tag{5.6}$$

A valid inequality for (5.5) is given by two functions $\phi : G \longrightarrow \mathbb{R}_+$ and $\pi : W \longrightarrow \mathbb{R}_+$ such that

$$\sum_{g \in G} \phi(g) \cdot t(g) + \sum_{w \in W} \pi(w) \cdot s(w) \geq 1 \tag{5.7}$$

is valid for all solutions (x_B, t, s) to (5.5). The functions ϕ and π are referred to as *valid functions* if Inequality (5.7) is valid for (5.5). Therefore the terms "valid inequality" and "valid function" are often used interchangeably. A valid function (ϕ, π) is *minimal* if there exists no other valid function (ϕ', π') such that $(\phi, \pi) \neq (\phi', \pi')$ and $\phi'(g) \leq \phi(g)$ for all $g \in G$ and $\pi'(w) \leq \pi(w)$ for all $w \in W$. A valid function (ϕ, π) is *extreme* if there do not exist two distinct valid functions (ϕ^1, π^1) and (ϕ^2, π^2) such that (ϕ, π) can be written as a convex combination of (ϕ^1, π^1) and (ϕ^2, π^2). The term extreme can be seen as a generalization of the term facet-defining when the group G is not finite.

5.4. Valid Inequalities

In this section we investigate two additional relaxations of the corner relaxation (5.3) which have played an important role in the recent development of valid inequalities from multi-row relaxations. In what follows we assume that the polyhedron P associated with the LP relaxation of the MIP (2.3) is rational, i.e. $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. By dropping the integrality restrictions on the non-basic integer variables in the corner relaxation we obtain a system of the form

$$\begin{aligned} x &= f + \sum_{j=1}^{k} r^j s_j, \\ x &\in \mathbb{Z}^q, \\ s &\in \mathbb{R}^k_+, \end{aligned} \tag{5.8}$$

where all basic integer and non-basic continuous variables are denoted by x and s respectively, and $J = \{1, \ldots, k\}$. Let $R_f(r^1, \ldots, r^k)$ denote the convex hull of all vectors $s \in \mathbb{R}^k_+$ such that the sum $f + \sum_{j=1}^{k} r^j s_j$ is integral. We have that $f, r^1, \ldots, r^k \in \mathbb{Q}^q$. Moreover, remember that $f \notin \mathbb{Z}^q$, i.e. the solution f is integer infeasible.

Borozan and Cornuéjols [41] propose to relax the system (5.8) by introducing a variable s_r for every $r \in \mathbb{Q}^q$. Thus they turn the k-dimensional space of the non-basic variables into an infinite dimensional space, leading to the *semi-infinite relaxation*

$$\begin{aligned} x &= f + \sum_{r \in \mathbb{Q}^q} r s_r, \\ x &\in \mathbb{Z}^q, \\ s &\geq 0, \\ s &\text{ has finite support.} \end{aligned} \tag{5.9}$$

Let R_f denote the convex hull of all vectors $s \geq 0$ having finite support such that the sum $f + \sum_{r \in \mathbb{Q}^q} r s_r$ is integral. Borozan and Cornuéjols [41] show that any valid inequality for R_f that cuts off the infeasible basic solution $x = f$, $s = 0$ is of the form

$$\sum_{r \in \mathbb{Q}^q} \psi(r)\, s_r \geq 1 \tag{5.10}$$

with $\psi : \mathbb{Q}^q \longrightarrow \mathbb{R}_+ \cup \{+\infty\}$. Moreover, they establish a beautiful correspondence between minimal valid inequalities of the form (5.10) and maximal lattice-free convex sets. A *lattice-free convex set* $S \subseteq \mathbb{R}^q$ is a convex set which contains no integer points in its interior, i.e. $\operatorname{int}(S) \cap \mathbb{Z}^q = \emptyset$. A lattice-free convex set S is *maximal* if there is no other lattice-free convex set S' distinct from S such that S is contained in S'. Now, consider the set

$$S_\psi = \{x \in \mathbb{Q}^q : \psi(x - f) \leq 1\} \tag{5.11}$$

and let $\operatorname{cl}(S_\psi)$ denote the topological closure of S_ψ in $\mathbb{R}^q$.

Theorem 5.1 ([41])**.** *Let $f \in \mathbb{Q}^q \setminus \mathbb{Z}^q$. A minimal valid inequality ψ for R_f is non-negative, piecewise linear, positively homogeneous and convex. Furthermore, the set $\operatorname{cl}(S_\psi)$ is a full-dimensional maximal lattice-free convex set containing f. Conversely, for any full-dimensional maximal lattice-free convex set $S \subset \mathbb{R}^q$ containing f there exists a minimal valid function ψ for R_f such that $\operatorname{cl}(S_\psi) = S$, and when f is in the interior of S, this function is unique.*

Theorem 5.1 states that any minimal valid inequality for R_f arises from a particular maximal lattice-free convex set S. Moreover, these inequalities are unique provided that f lies in the interior of S. So there is a one-to-one correspondence between maximal lattice-free convex sets and minimal valid functions ψ. This case is called *non-degenerate* and ψ is referred to as a *non-degenerate function.* On the other hand, there is the *degenerate* case in which the maximal-lattice free convex set S contains f on its boundary. In this case, the set S corresponds to several minimal valid *degenerate functions* ψ yielding different minimal inequalities. However, Zambelli [176] showed that for any minimal valid inequality for the finite dimensional set $R_f(r^1, \dots, r^k)$ there exists a non-degenerate function generating it.

5.4.1. Two-Row Cuts

In the previous discussion we presented some results concerning the relaxation $R_f(r^1, \dots, r^k)$. While these results are valid for an arbitrary dimension q of the system (5.8), several authors have studied the two-row special case ($q = 2$) in detail.

Andersen et al. [11] were the first to study the problem $R_f(r^1, \dots, r^k)$ with two rows. They showed that all facets of $R_f(r^1, \dots, r^k)$ are intersection cuts which are generated from two-dimensional lattice-free convex sets such as splits, triangles and quadrilaterals. Cornuéjols and Margot [63] present sufficient conditions for a lattice-free convex set to define a facet of $R_f(r^1, \dots, r^k)$. They show that degenerate lattice-free convex sets are not needed to define the facets of $R_f(r^1, \dots, r^k)$. Moreover, Cornuéjols and Margot characterize the extreme inequalities for R_f in the two-row case.

Dey and Wolsey [78, 79] address the strengthening of multi-row inequalities by using the integrality of some of the non-basic variables (see also Conforti et al. [54] and Basu et al. [34]). Moreover, they provide a classification of maximal lattice-free triangles. Triangles of type 1 have integral vertices and contain an integer point in the relative interior of each edge. Triangles of type 2 have more than one integral point in the relative interior of one edge. Triangles of type 3 have non-integral vertices and contain one integral point in the relative interior of each edge. Now, consider the two-row relaxation

$$\begin{aligned} x_1 &= f_1 + \sum_{j \in J} r_1^j x_j, \\ x_2 &= f_2 + \sum_{j \in J} r_2^j x_j, \\ x_j &\geq 0, \quad j \in J, \\ x_j &\in \mathbb{Z}, \quad j \in N_I, \end{aligned} \tag{5.12}$$

where $\{1, 2\} \subseteq B_I$, i.e. x_1 and x_2 are basic integer variables. Let $S \subseteq \mathbb{R}^2$ be a maximal lattice-free convex set containing $f = (f_1, f_2)$ in its interior. Then define the function $\pi : \mathbb{R}^2 \longrightarrow \mathbb{R}_+$ with

$$\pi(w) = \begin{cases} 0 & \text{if } w \in \text{ recession cone of } S, \\ \lambda & \text{if } f + \frac{1}{\lambda} w \in \text{boundary}\,(S)\,, \end{cases} \tag{5.13}$$

where boundary(S) denotes the boundary of the set S. The inequality

$$\sum_{j \in J} \pi\left(r^j\right) x_j \geq 1 \tag{5.14}$$

is valid for (5.12). However, as some of the non-basic variables are integer-constrained this inequality is not minimal in general. The inequality

$$\sum_{j \in J_I} \phi^0\left(r^j\right) x_j + \sum_{j \in J \setminus J_I} \pi\left(r^j\right) x_j \geq 1 \tag{5.15}$$

is also valid for (5.12) where $\phi^0 : [0,1[^2 \longrightarrow [0,1]$ is called the *trivial fill-in function*

$$\phi^0\left(w\right) = \min_{u \in \mathbb{Z}^2} \left\{\pi\left(w + u\right)\right\}. \tag{5.16}$$

Since $\pi(r^j) \geq \phi^0(r^j)$ for $j \in J_I$, Inequality (5.15) dominates (5.14). Again note that we assume that the columns associated with non-basic integer variables have been replaced by their fractional parts such that $r^j \in [0,1[^2$. Dey and Wolsey showed that Inequality (5.15) is minimal when the maximal lattice-free convex set S is a triangle of type 1 or type 2.

Let $S_f(r^1, \dots, r^k)$, $T_f(r^1, \dots, r^k)$ and $Q_f(r^1, \dots, r^k)$ denote the closures of the split, triangle and quadrilateral inequalities respectively. The closure of the split inequalities is the intersection of all valid inequalities derived from lattice-free split bodies. The triangle and quadrilateral closure are defined analogously. Basu et al. [33] examine the relative strength of these closures from a theoretical point of view. Given the results discussed above, we have $R_f(r^1, \dots, r^k) = S_f(r^1, \dots, r^k) \cap T_f(r^1, \dots, r^k) \cap Q_f(r^1, \dots, r^k)$. Now, an interesting question is whether one of these closures plays a more important role in approximating $R_f(r^1, \dots, r^k)$ than the others. Basu et al. proved that $T_f(r^1, \dots, r^k) \subseteq S_f(r^1, \dots, r^k)$ and $Q_f(r^1, \dots, r^k) \subseteq S_f(r^1, \dots, r^k)$, i.e. they showed that the triangle and quadrilateral closure are at least as strong as the split closure. They showed, moreover, that the triangle closure and the quadrilateral closure close at least half of the integrality gap while the amount of integrality gap closed by the split closure can be arbitrarily small. Andersen et al. [12] generalized this result to relaxations containing an arbitrary number of rows of the simplex tableau. They showed that intersection cuts derived from lattice-free convex sets with so-called full split-dimension are crucial to obtaining a good approximation of the integer hull. These results suggest that cutting planes from two rows are stronger than most of the cutting planes used in state-of-the-art MIP solvers.

On the other hand, Dey and Louveaux [76] showed that intersection cuts generated from maximal lattice-free triangles (except for type 1 triangles, see Cook et al. [57]) and quadrilaterals have finite split rank. Almost all triangle and quadrilateral cuts can therefore be obtained through a sequence of split cuts. A more general characterization of cuts with infinite split rank is provided by Basu et al. [36].

He et al. [112] present a probabilistic comparison of type 1 triangle cuts and split cuts for two-row mixed-integer programs. They show that, if the vectors f and r^j are subject to specific probability distributions, then there is a high likelihood that split cuts dominate type 1 triangle cuts with respect to the size of the cut coefficients and the volume cut off from the LP relaxation.

Espinoza [83, 84] reports on successful computational experience with unstrengthened intersection cuts derived from several families of maximal lattice-free convex sets. Basu et al. [36] study the computational effectiveness of a family of strengthened type 2 triangle cuts generated from degenerate simplex tableaus. Their results show that the selected family of two-row cuts only provides a slight improvement over the Gomory mixed-integer cuts. Dey et al. [75] consider a different family of strengthened type 2 triangle cuts. They show that these cuts are effective in increasing the amount of integrality gap closed on randomly generated multidimensional knapsack instances.

5.4.2. Intersection Cuts

In this section we deal with intersection cuts from general lattice-free convex sets. We discussed the special case where the lattice-free convex set is a split set in Section 4.4.2. Consider the multi-row relaxation

$$\begin{aligned} x &= f + \sum_{j \in J} r^j x_j, \\ x_j &\geq 0, \quad j \in J, \\ x_j &\in \mathbb{Z}, \quad j \in N_I, \end{aligned} \tag{5.17}$$

which is essentially the same as (5.1) except that the non-negativity conditions on the basic variables have been removed. We assume that the solution f is integer infeasible. Now suppose that the set

$$S = \left\{ x \in \mathbb{R}^n : \pi^i x \leq \pi_0^i, i = 1, \dots, l \right\} \tag{5.18}$$

is a lattice-free convex set containing f in its interior. We require that $\pi_j^i = 0$ for all $j \in N \setminus N_I$ and $i = 1, \dots, l$ since these components of x are associated with continuous variables. We can rewrite the set S in the form $\{x \in \mathbb{R}^n : \wedge_{i=1}^l (\pi^i x \leq \pi_0^i)\}$. We can therefore represent the fact that the solution f is not allowed to lie in the interior of S by the l-term disjunction

$$\bigvee_{i=1}^{l} \left(\pi^i x \geq \pi_0^i \right). \tag{5.19}$$

By using the definition of x from (5.17) we obtain the valid inequalities

$$\pi^i \left(f + \sum_{j \in J} r^j x_j \right) \geq \pi_0^i, \quad i = 1, \ldots, l. \tag{5.20}$$

The solution f lies in the interior of the set S, implying that $\pi_0^i - \pi^i f > 0$ for $i = 1, \ldots, l$. We can therefore rewrite this set of inequalities as

$$\sum_{j \in J} \left(\frac{\pi^i r^j}{\pi_0^i - \pi^i f} \right) x_j \geq 1, \quad i = 1, \ldots, l. \tag{5.21}$$

To obtain a valid inequality for the disjunction (5.19), we apply the disjunctive principle (cf. Proposition 2.1). We get the intersection cut

$$\sum_{j \in J} \max_{i=1,\ldots,l} \left\{ \frac{\pi^i r^j}{\pi_0^i - \pi^i f} \right\} x_j \geq 1. \tag{5.22}$$

For each non-basic variable $j \in J$ define a vector α^j with components

$$\alpha_i^j = \begin{cases} \frac{\pi_0^i - \pi^i f}{\pi^i r^j} & \text{if } \pi^i r^j > 0, \\ +\infty & \text{otherwise,} \end{cases} \tag{5.23}$$

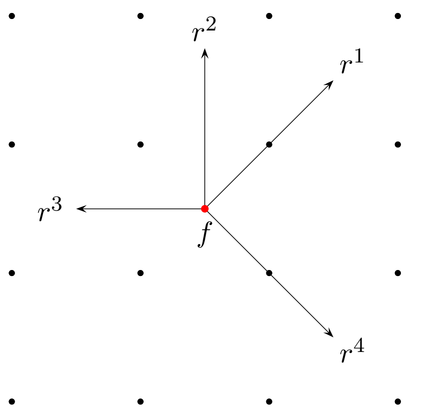
for $i = 1, \ldots, l$. We can then restate the intersection cut as

$$\sum_{j \in J} \frac{x_j}{\min_{i=1,\ldots,l} \left\{ \alpha_i^j \right\}} \geq 1. \tag{5.24}$$

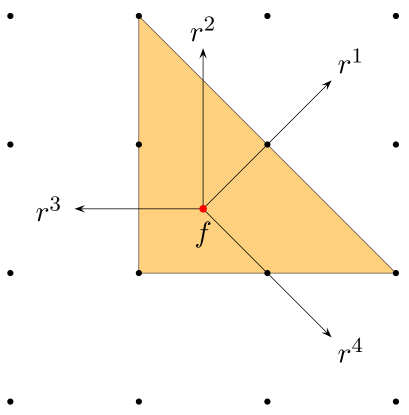
Consider the half-line $f + \alpha r^j$ with $\alpha > 0$ starting in f in the direction r^j. The point $f + \alpha_i^j r^j$ is the point at which the extreme ray r^j and the hyperplane $\pi^i x = \pi_0^i$ intersect (see Figure 5.2).

Next we discuss the strengthening of the intersection cut. We again consider modifying each term of the disjunction (5.19) on the non-basic integer variables. We get

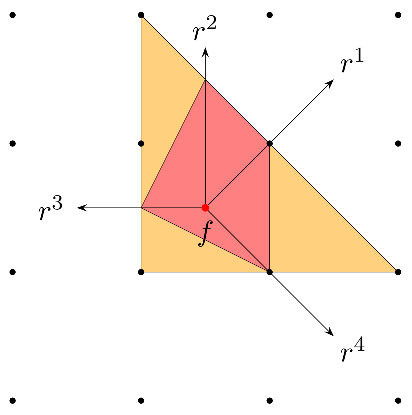
$$\bigvee_{i=1}^{l} \left(\left(\pi^i - h^i \right) x \geq \pi_0^i \right), \tag{5.25}$$



(a) A fractional solution f and four extreme rays



(b) A maximal lattice-free triangle



(c) A two-row cut

Figure 5.2. Example of the derivation of a two-row cut

where $h^i \in \mathbb{Z}^n$ with $h^i_j = 0$ for $i = 1, \dots, l$ and for all $j \notin J_I$. If the original disjunction (5.19) is a split disjunction (4.18) any modified disjunction of the form (4.34) is again a split disjunction and thus satisfied by all feasible integral solutions (see Section 4.4.2). The optimal strengthening is then given by a closed-form formula (see Proposition 4.5). Concerning general multiple-term disjunctions like (5.19) the modification shown in Equation (5.25) is not, however, valid for arbitrary choices of the vectors h^i. We therefore consider a special case in which for $i = 1, \dots, l$ the vectors h^i are given by

$$h^i_j = \begin{cases} \pi^i u^j & \text{if } j \in J_I, \\ 0 & \text{otherwise,} \end{cases} \tag{5.26}$$

for $j = 1, \dots, n$ and where $u^j \in \mathbb{Z}^n$ for $j \in J_I$. We obtain the modified disjunction

$$\bigvee_{i=1}^{l} \left(\pi^i x - \sum_{j \in J_I} \left(\pi^i u^j \right) x_j \geq \pi^i_0 \right), \tag{5.27}$$

which we can rewrite as

$$\bigvee_{i=1}^{l} \left(\pi^i \left(x - \sum_{j \in J_I} u^j x_j \right) \geq \pi^i_0 \right). \tag{5.28}$$

Since we assumed that the original disjunction (5.19) is valid and the term $x - \sum_{j \in J_I} u^j x_j$ is integral the modified disjunction is indeed valid. We obtain the intersection cut

$$\sum_{j \in J_I} \max_{i=1,\dots,l} \left\{ \frac{\pi^i \left(r^j - u^j \right)}{\pi^i_0 - \pi^i f} \right\} x_j + \sum_{j \in J \setminus J_I} \max_{i=1,\dots,l} \left\{ \frac{\pi^i r^j}{\pi^i_0 - \pi^i f} \right\} x_j \geq 1. \tag{5.29}$$

Clearly, we are interested in making the coefficients on the integer variables as small as possible. We get

$$\sum_{j \in J_I} \min_{u^j \in \mathbb{Z}^n} \left\{ \max_{i=1,\dots,l} \left\{ \frac{\pi^i \left(r^j - u^j \right)}{\pi^i_0 - \pi^i f} \right\} \right\} x_j + \sum_{j \in J \setminus J_I} \max_{i=1,\dots,l} \left\{ \frac{\pi^i r^j}{\pi^i_0 - \pi^i f} \right\} x_j \geq 1. \tag{5.30}$$

The trivial fill-in function can thus be viewed as a strengthening of the disjunction (5.19).

Chapter 6.

Required Work

The purpose of this chapter is to describe in detail the goals of this thesis. We first summarize the review of the state-of-the-art in cutting plane approaches we presented in Chapters 4 and 5 and identify research gaps. Following this discussion we then introduce the objectives of this thesis.

Our review of the literature on general-purpose cutting planes points to a number of conclusions. The large number of publications shows that general-purpose cutting planes are an active area of research. Balas and Saxena [31] demonstrate that the elementary split closure gives a tight approximation of the integer hull of many mixed-integer programs. Optimizing over the split closure, however, is $\mathcal{NP}$-hard as shown by Caprara and Letchford [44]. In line with these findings, research is devoted to families of split cuts such as the Gomory mixed-integer cuts [99] which can be generated efficiently. Several authors propose algorithms for obtaining improved Gomory mixed-integer cuts; see Cornuéjols et al. [61], Ceria et al. [46], Andersen et al. [8] and Balas et al. [24]. As pointed out by Cornuéjols and Nannicini [64], the existing approaches for the efficient generation of split cuts are, however, far from exploiting the full strength of the split closure. Moreover, a computational comparison of these approaches has not been conducted.

Andersen et al. [11] initiated new interest in cutting planes generated from multi-row relaxations, showing that all facet-defining inequalities of the convex hull of a mixed-integer set defined by a system of two equations with two free integer variables and non-negative continuous variables are intersection cuts derived from maximal lattice-free splits, triangles and quadrilaterals. Borozan and Cornuéjols [41] consider a semi-infinite relaxation consisting of q equations with q free integer variables and an infinite number of non-negative continuous variables. They show that minimal valid inequalities for this relaxation correspond to maximal lattice-free convex sets. Dey and Wolsey [78, 79] demonstrate how cutting planes derived from two rows of a simplex tableau can be strengthened using the integrality of some of the non-basic variables. Considering the advances in the theoretical understanding of inequalities derived from multi-row relaxations, there is only limited computational experience with their use as cutting planes in a cut-and-branch (or branch-and-cut) framework. Espinoza [83, 84] was the first to show that generating cutting planes from multi-rows of a simplex tableau can positively influence the performance of an MIP solver by implementing separators for subclasses of these inequalities as a cut callback in Cplex. Espinoza does not, however, consider the strengthening proposed by Dey and Wolsey. Basu et al. [32] study a family of two-row cuts generated from degenerate simplex tableaus. In their experiments Basu et al. observe the selected family of two-row cuts not to be competitive with the Gomory mixed-integer cuts. Dey et al. [75] report on preliminary computational experience with lifted two-row cuts (trivial fill-in function) derived from a family of maximal lattice-free triangles of type 2 on randomly generated multidimensional knapsack instances. They show that this family of cuts is effectively reducing the integrality gap in comparison with the Gomory mixed-integer cuts. Dey et al. also point out the need for further computational experimentation with multi-row cuts. In general, there are only few publications discussing implementation details of cut separators. It is, on the other hand, well known that technical details can greatly affect the performance of a cut separator and thus the overall performance of an MIP solver. In summary, research is needed into the separation of single-row and multi-row cuts, efficient implementations of cut separators and meaningful computational experiments.

Based on the review of the state-of-the-art summarized above, this thesis pursues three main research objectives which are of computational nature.

1. We propose to develop a new heuristic approach for improving the performance of the Gomory mixed-integer cuts which is based on pivoting. The main idea behind our approach is to increase the distance cut off by a Gomory mixed-integer cut by reducing the size of the coefficients of the continuous variables in the row of the simplex tableau from which it is

derived. Andersen et al. [8] successfully used a similar reduction algorithm to generate reduce-and-split cuts.

We want to implement various cut separators which generate split cuts for integer and mixed-integer programs, including two Chvátal-Gomory cut separators, the Gomory mixed-integer cut separator and five variations of the latter cut separator. Furthermore, we want to give a detailed description of our implementation of these cut separators. We particularly aim to highlight important computational techniques making these cut separators efficient in practice.

In addition, we want to conduct meaningful computational experiments with the discussed cut separators. The questions we intend to answer are: which approach to strengthening Gomory's mixed-integer cuts is most effective in solving practical MIP instances? Concerning the performance of the lift-and-project cuts, is it beneficial to apply disjunctive modularization or the Euclidean normalization? How do the Chvátal-Gomory cut separators perform in comparison with the Gomory mixed-integer cut separator and its variants?

2. We aim to implement cut separators which generate cuts from multiple rows of a simplex tableau, i.e. intersection cuts which are derived from maximal lattice-free convex sets other than split sets. We also intend to discuss in detail our implementation of these cut separators. In particular, we want to address the construction of a multi-row relaxation and the properties of selected families of maximal lattice-free convex sets. Espinoza [84] used some of these sets to generate intersection cuts and obtained promising results. We also want to integrate the trivial fill-in function into our implementation in order to strengthen the multi-row cuts.

   Besides the description of the multi-row cut separators, we want to provide computational results which enable evaluation of their practical value in solving mixed-integer programs. Questions we want to address are: How do multi-row cuts perform in comparison with split cuts? Are intersection cuts derived from certain families of maximal lattice-free convex sets more effective than others? What is the benefit of deriving cuts from relaxations which consist of more than two rows? Or, in other words, do intersection cuts derived from higher-dimensional maximal lattice-free convex sets yield a larger performance improvement than those derived from triangles or quadrilaterals? How is the performance affected if the multi-row cuts are strengthened using the trivial fill-in function?

3. The first two objectives of this thesis involve developing various cut separators. Typically, these cut separators produce large numbers of cutting planes violated by the current optimal solution of the LP relaxation of the associated MIP. Adding all generated cutting planes to the problem formulation is problematic since they slow down the solution of the LP relaxation. We want therefore to develop a cut selection algorithm which only selects a subset of the best cuts with respect to some quality measure. We furthermore intend to study various quality measures and to compare their performance by computational experiments.

The remainder of this thesis is organized as follows. Chapter 7 introduces the MIP solver Mops and discusses some basic aspects of our implementation such as data structures and numerical considerations. In Chapter 8 we describe several cut separators for subclasses of split cuts and present a new pivoting algorithm for improving the performance of the Gomory mixed-integer cuts. We also report on our computational experience with these cut separators. Chapter 9 describes our implementation of several variants of multi-row cut separators and discusses computational results. In Chapter 10 we develop a cut selection algorithm and analyze its effect on the performance of Mops. Chapter 11 summarizes the results of this thesis, offers some conclusions and points to opportunities for further research.

Part III.

Implementation and Numerical Results

Chapter 7.

Framework

Part II of this thesis was devoted to the state-of-the-art in cutting plane technology. We discussed general-purpose cutting planes for both pure integer and mixed-integer programs. In Chapter 4 we considered split cuts such as Chvátal-Gomory cuts and Gomory mixed-integer cuts and also discussed several approaches for improving the performance of the Gomory mixed-integer cuts. In Chapter 5 we showed how to derive cutting planes from relaxations which consist of multiple rows of the simplex tableau.

In this chapter we describe the framework in which we implement separation algorithms for some of the cutting planes discussed in Chapters 4 and 5. We discuss the system architecture and some of the main data structures.

This chapter is organized as follows. In Section 7.1 we describe the MIP solver MOPS and detail its architecture, history and main algorithms. Some important aspects of our implementation are discussed in Section 7.2.

7.1. MOPS - An MIP Solver

The software package MOPS[1] is a high-performance solver for linear and mixed-integer programming problems. Starting as a pure LP solver based on a primal simplex algorithm in 1987, the system today features powerful IP preprocessing and an effective branch-and-cut algorithm. In particular, MOPS ranks among the top systems in the world for solving large-scale real-world linear and mixed-integer programming problems. MOPS is, for the most part, written in Fortran77 and is available for various platforms such as standard PCs, servers and mainframes.

[1]**M**athematical **OP**timization **S**ystem

topic	reference(s)
- system architecture	[158]
- LU factorization	[160]
- LU update	[156]
- LP preprocessing	[137]
- primal simplex algorithm	[157]
- dual simplex algorithm	[123–125]
- IP preprocessing	[162, 163]
- cutting planes	[51, 169–171]
- branch-and-bound	[90]
- branch-and-cut	[167]

Table 7.1. Documentation of algorithms and computational techniques

The system is a commercial product which has been used in many practical applications [121, 155, 159, 161]. Since its initial version, Mops was significantly improved in terms of algorithms, software design and implementation. These improvements have been documented in many scientific publications. Some of these publications are shown in Table 7.1.

7.1.1. Evolution

Like other MIP solvers, Mops has undergone a rapid evolution in the last two decades. Some of the main steps in the development of Mops are shown in Table 7.2 (cf. Koberstein [123]).

Only very limited conclusions concerning the overall performance of an MIP or LP solver can be drawn from the performance on particular problem instances. Nevertheless, this approach provides an indication of how the performance of an MIP or LP solver has improved. For several versions of Mops, Tables 7.3 and 7.4 show the times which the different LP engines and the MIP engine need to solve the benchmark instance `oil` to optimality (cf. Koberstein [123]). This instance consists of 5563 constraints, 6181 variables in total, including 74 binary variables, and 39597 non-zero elements in the coefficient matrix.

year	version	description
1987	1.0	primal simplex, LU factorization, and PFI update
1988	1.1	LU update of the basis factorization
1989	1.2	LP preprocessing, update
1991	1.3	new pivot row selection minimizing the sum of infeasibilities
1992	1.4	new scaling, ftran, devex
1994	2.0	mixed 0-1 programming with supernode processing (IP preprocessing)
1995	2.5	mixed-integer programming with general node selection
1997	3.0	first version of dual simplex for branch-and-bound phase
1998	3.5	improved supernode processing
1999	4.0	additional interior point algorithm to solve initial LP
2001	5.0	new memory management, improved numerical kernels
2003	6.0	lifted cover cuts
2003	7.0	fixed charge and general bound reduction by solving LPs
2004	7.5	new dual simplex algorithm for initial LP and branch-and-bound
2004	7.6	Gomory mixed-integer cuts
2005	7.7	improved dual simplex algorithm
2006	7.8	improved primal simplex algorithm
2007	8.0	MOPS studio with AMPL interface
2008	9.0	mixed-integer rounding cuts, new branch-and-bound algorithm
2009	10.0	new LP preprocessing, lifted clique cuts

Table 7.2. Development of MOPS

year	version	hardware and software platform	solution time (seconds)
1991	1.4	I486 (25 MHz)	612.4
1995	2.5	P133 Win 3.11	20.7
1999	4.0	PIII (400 MHz), Win 98	5.1
2001	5.0	PIII (500 MHz), Win 98	3.9
2002	6.0	PIV (2.2 GHz), Win 2000	0.9
2005	7.6	PIV (3.0 GHz), Win 2000, primal	1.1
2005	7.8	PIV (3.0 GHz), Win 2000, dual	1.6
2005	8.0	PIV (3.0 GHz), Win 2000, IPM	0.6

Table 7.3. Improvement of the MOPS LP engines on model `oil`

year	version	hardware and software platform	solution time (seconds)
1994	2.0	PII (500 MHz), LIFO MIP	1794.3
1995	2.5	PII (500 MHz), general node selection	450.1
1999	4.0	PIV (2.2 GHz), IPM for initial LP	75.2
2003	6.3	PIV (2.2 GHz), various improvements	39.6
2005	7.8	PIV (3.0 GHz), Gomory mixed-integer cuts, dual simplex in B&B	11.4
2008	9.0	PIV (3.0 GHz), new B&B algorithm	6.9
2009	10.0	Core2Duo, new LP preprocessing	3.6

Table 7.4. Improvement of the MOPS MIP engine on model `oil`

7.1.2. External System Architecture

This section details the external system architecture of MOPS which is also depicted in Figure 7.1 (cf. Koberstein [123]). One way of accessing the MOPS code is via a dynamic link library (mops.dll) and a static link library (mops.lib). Both of these libraries can be integrated into user applications. The static link library provides direct access to the data structures and solutions routines by means of the Fortran, C and IMR interfaces. The dynamic link library provides interface functions which allow for access to a selected subset of the MOPS core routines.

The MOPS executable (mops.exe) is available for several system platforms. It is controlled via a text file, the so-called MOPS profile, which enables the user to change the parameters of the LP and MIP optimization. The problem data are passed to MOPS using the MPS file format (see, for instance, Murtagh [140]) or a MOPS-specific triplet format. Additional files are used to store statistics, solution data, LP bases and branching trees.

Finally, we point to two user-friendly graphical tools. ClipMOPS is an MS Excel Add-In which is based on the mops.dll and allows for formulating and solving LP

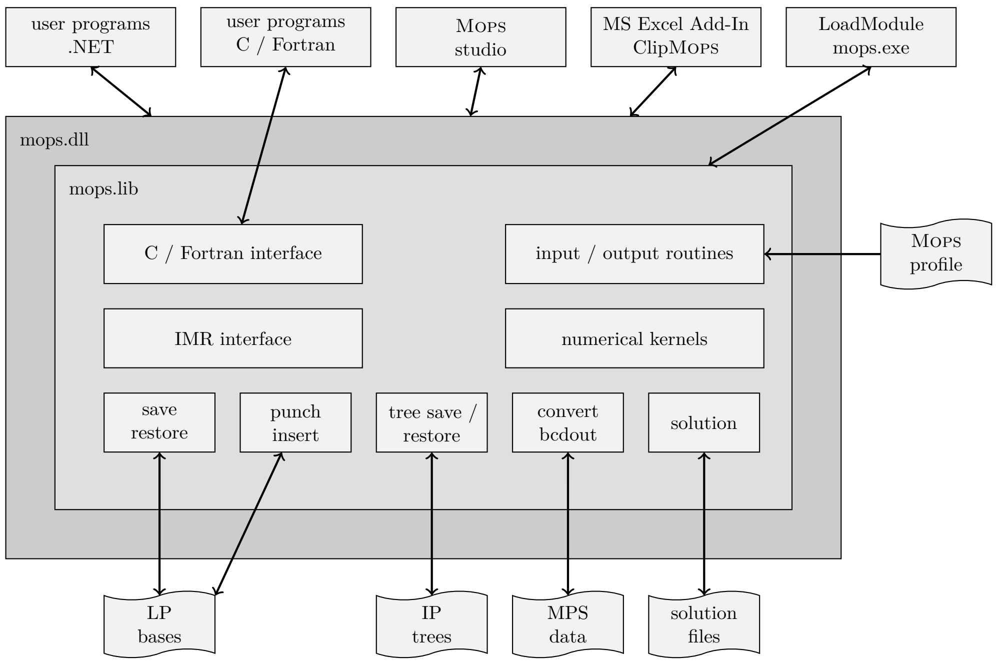


Figure 7.1. External architecture of Mops

and MIP models with up to 250 columns and 400 constraints. Mops studio is a powerful frontend for Mops in which models can be formulated in a modeling language such as AMPL [89] and solved interactively.

7.1.3. External and Internal Model Representation

In the previous chapters we worked with MIPs in the standard forms (2.1) and (2.3). While any MIP can be transformed into these standard forms, MIP solvers like Mops must be able to handle MIPs which are of a more general form. Consider the MIP

$$\begin{aligned} \text{(EMR)} \quad \min \quad & cx, \\ \text{s.t.} \quad & L \le \tilde{A}x \le D, \\ & \tilde{l} \le x \le \tilde{d}, \\ & x_j \in \mathbb{Z}, \quad \forall j \in \tilde{N}_I, \end{aligned} \tag{7.1}$$

where c, $x \in \mathbb{R}^n$, $\tilde{A} \in \mathbb{R}^{m\times n}$, $L, D \in (\mathbb{R} \cup \{-\infty, +\infty\})^m$, $\tilde{l}, \tilde{d} \in (\mathbb{R} \cup \{-\infty, +\infty\})^n$ and $\tilde{N}_I \subseteq \tilde{N} = \{1, ..., n\}$. In contrast to the standard form, the MIP (7.1) contains ranges (L, D) on the constraints and bounds ($\tilde{l}$, $\tilde{d}$) on the variables. We refer to (7.1) as the *external model representation (EMR).*

Internally the MIP (7.1) is transformed into a form which is quite similar to that of the standard equality form (2.3). This is accomplished by a standardized procedure which adds a complete identity matrix to the constraint matrix $\tilde{A}$. We get

$$\begin{array}{rlrcl} \text{(IMR)} & \min & & cx_S, & \\ & \text{s.t.} & Ax & = & 0, \\ & l \leq & x & \leq & d, \\ & & x_j & \in & \mathbb{Z}, \quad \forall j \in N_I, \end{array} \tag{7.2}$$

where $A = (\tilde{A}, I) \in \mathbb{R}^{m\times(n+m)}$, $x = (x_S, x_L) \in \mathbb{R}^{n+m}$, $l = (\tilde{l}, -D) \in \mathbb{R}^{n+m}$, $d = (\tilde{d}, -L) \in \mathbb{R}^{n+m}$ and $N_I \subseteq N = \{1, ..., n+m\}$. The system (7.2) is called the *internal model representation (IMR).* Note that the matrix A has full row rank. The ranges on the constraints are transformed into bounds on the logical variables. The variables x_S which are associated with the matrix $\tilde{A}$ are called *structural variables* (or *structurals*) and the remaining variables which were introduced to obtain equality constraints are called *logical variables* (or *logicals*).

We keep to the notation we introduced for the MIPs in standard form (2.1) and (2.3). Note that a basis of the LP relaxation of the IMR may also contain columns which are associated with logical variables. In the remainder of this thesis we shall consider MIPs mainly in their IMR. For simplicity we assume that the IMR does not contain free variables. All of the algorithms discussed in the following chapters can nevertheless be easily adapted to handle free variables.

7.1.4. MIP Solution Process

The process of solving an LP or MIP problem consists of several steps which are shown in Figure 7.2 (cf. Koberstein [123]). In the *data management phase* memory is allocated and the data structures are initialized. Typically, the standard data structures used in MOPS are arrays. These arrays are arranged in a contiguous

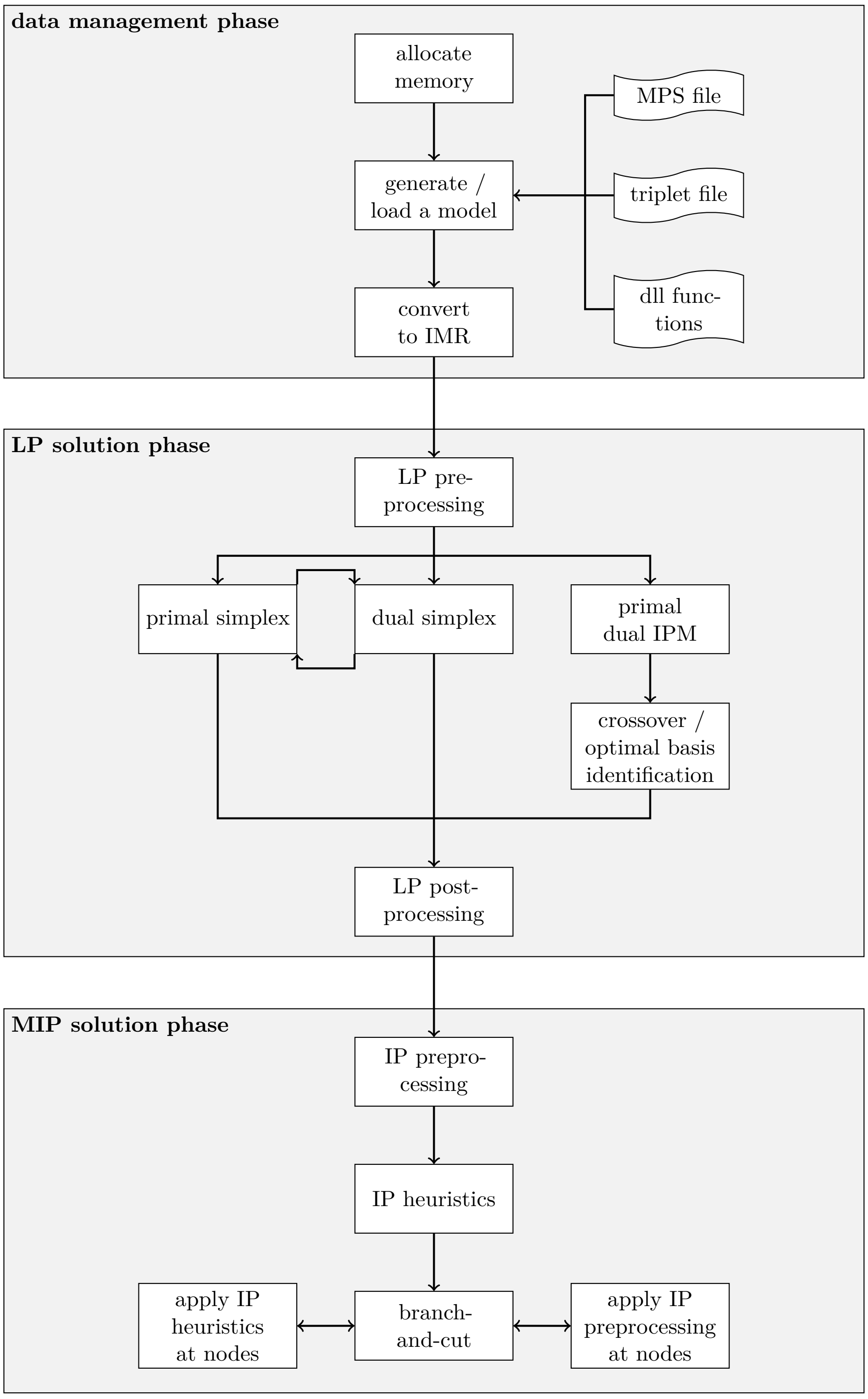


Figure 7.2. MIP solution process

memory block in order to benefit from data caching. The problem data are then read from a file or passed by a dll function and converted into the IMR format.

The first step in the *LP solution phase* is the LP preprocessing. During LP preprocessing a number of techniques are applied with the objective of reducing the size of the problem. For instance, redundant constraints and variables are removed. In addition, an elimination procedure is used further to reduce the number of variables and constraints. The level of preprocessing that is applied is dependent on whether the model is an MIP or LP problem. In particular, the most aggressive LP preprocessing is only applied to LP problems. For MIP problems, a slightly restricted LP preprocessing is performed, since aggressive LP preprocessing can lead to a less effective IP preprocessing. The preprocessed problem is then solved by the primal simplex algorithm, the dual simplex algorithm or by an interior point method (IPM). The main difference between the two simplex engines and the IPM engine is that the optimal solution computed by the IPM engine is usually not basic. To obtain a basic optimal solution a crossover algorithm is performed. This algorithm is in fact a specialized simplex algorithm. If one of the simplex engines is used, the problem is perturbed (see [123]) in order to cope with degeneracy. With respect to these perturbations, the primal and dual simplex algorithm form an entangled pair: when solving a problem with the dual simplex algorithm, the primal simplex algorithm is used to remove the dual perturbation, and vice versa. Thus efficient implementations of both simplex engines are needed. The last step of the LP solution phase is the LP postprocessing which deduces an optimal solution to the original problem from an optimal solution to the preprocessed problem. If an MIP problem is solved, the third solution phase is entered. Otherwise the solution process terminates.

The third phase is the *MIP solution phase*. The first step in this phase is to perform the IP preprocessing (or supernode processing [162]) in order to strengthen the LP relaxation of the MIP problem. Besides coefficient or bound reduction techniques, cutting planes play a major role in this regard. Mops generates cutting planes in rounds. Usually a large number of cutting planes can be derived. When added to the problem formulation, cutting planes increase the problem size and slow down the LP engine. Therefore it is important to select only a careful subset of the generated cuts. Primal heuristics are applied in order to find feasible solutions to the MIP problem. The objective value of any such feasible solution provides a primal bound on the optimal objective value. The branch-and-cut algorithm is then started. By default the LP relaxations of the subproblems encountered during the branch-and-bound search are solved with the dual simplex algorithm. Heuristics and specific preprocessing techniques (such as cutting planes) can also be applied at nodes of the branching tree. The default setting of Mops is to only derive cutting planes at the root node, i.e. Mops in fact uses a cut-and-branch algorithm.

7.2. Implementation

The purpose of this section is to present some technical details of our implementation. We describe the main data structures and discuss some numerical considerations.

7.2.1. Basic Data Structures

The Mops MIP solver stores the problem data and all intermediate and final results of algorithms in array-based data structures. Typically arrays are arranged in a contiguous memory block. This approach has the advantage that all arrays can be addressed very efficiently by calculating the offsets in this block. In the following we discuss the data structures for storing matrices and vectors.

MIP solvers like Mops usually store matrices in a compact form meaning that only non-zero entries are considered. Figure 7.3 shows the standard data structure

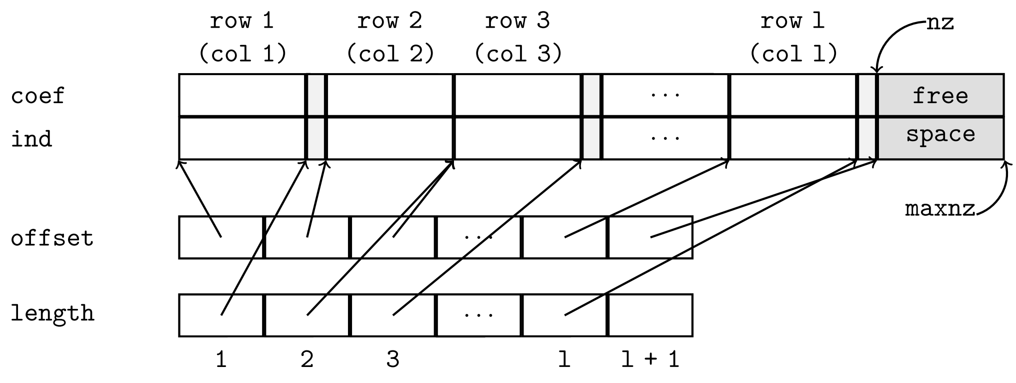


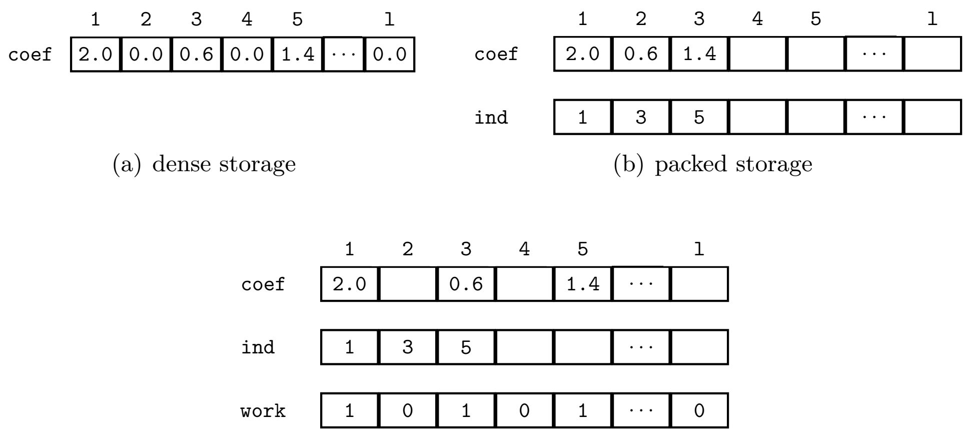
Figure 7.3. Compact storage of a matrix in row-wise or column-wise format

for storing matrices. For instance, the matrix $\tilde{A}$ which is the structural part of the coefficient matrix in the IMR (7.2) is stored using this data structure. In general a matrix can be stored in row-wise or column-wise format. Let us suppose we chose the row-wise format. For each row of the matrix the arrays `coef` and `ind` store the coefficients and the column indices respectively. The components of the array `offset` point to the starting positions of the rows in `coef` and `ind`. The array `length` is used to store the lengths of the rows, i.e. the number of non-zero elements in each row. In Figure 7.3 each element of `length` points to the last element of a row in `coef` and `ind`. In fact this last element is given by `offset[i] + length[i] - 1`. The light gray areas represent free space between rows. The gray area at the end of `coef` and `ind` is the free space for additional rows. As hinted in Figure 7.3 this data structure can similarly store a matrix in the column-wise format.

The advantages of storing a matrix in this compact row-wise (column-wise) format are that the rows (columns) can be added and accessed very efficiently. The data structure also easily allows for adding non-zeros to existent rows or columns. Moreover, rows or columns can also be replaced efficiently. On the other hand, given a row-wise representation of a matrix it is not possible to find all rows that contain a certain column index without iterating over the array `ind` of column indices. Therefore using a row-wise representation is inefficient if columns need to be accessed. Conversely, a column-wise representation is not the right choice if rows need to be extracted frequently. To avoid performance degradation a matrix is often stored in both row-wise and column-wise format. Note that changes in the row-wise structure need to be synchronized with the column-wise structure and vice versa. In MIP solvers this is accomplished by performing a sparse transpose which extracts one of the two representations from the respective other one.

We now turn to a second important data structure. In our code we often work with vectors, e.g. cut coefficient vectors. Mathematical vectors can be stored in different ways.

- *Dense storage:* A single array contains all of the vector's entries including zero elements.



(a) dense storage

(b) packed storage

(c) indexed storage

Figure 7.4. Vector storage

- *Packed storage:* A vector is represented by two compact arrays. One of these arrays contains only the non-zero elements of the vector while the other one stores the corresponding indices.
- *Indexed Storage:* The elements of the vector are stored in a dense array and the indices are stored in a compact array. The non-zero positions of the vector are held in an additional array.

Figure 7.4 presents these data structures. Dense storage consumes less memory than the other two methods. However, iterating over the non-zero positions of a sparse vector in dense storage is very expensive. Moreover, the array `coef` needs to be zeroed out after each usage. Concerning packed storage the situation is different. As only the non-zero elements are stored in a compact form, looping over them is cheap. But packed storage also consumes additional memory for the stack `ind` which stores the vector indices. Moreover, it is not possible directly to access the element of a vector at a specific index without iterating over the stack of indices. Indexed storage combines the advantages of packed and dense storage. Looping over a vector can be done efficiently and all elements can be accessed directly. In addition we use the array `work` to mark the non-zero positions in the array of coefficients `coef`. With this it is possible to decide whether a vector contains an element at a specific index without checking for a non-zero coefficient (numerical issues). Nor do we zero out the array `coef` in indexed storage but zero out the marker array `work` using the stack `ind` after each usage.

7.2.2. Numerical Considerations

Floating point calculations are subject to rounding errors and are consequently by nature inexact (see [96]). On the other hand, algorithms using floating point arithmetic are usually considerably faster than those based on exact (rational) arithmetic. Like most MIP solvers MOPS is based on floating point arithmetic. MOPS deals with numerical issues by introducing tolerances for feasibility, optimality and integrality among others. In most cases the inherent inaccuracy of floating point arithmetic can be neglected, i.e. a solution can be declared feasible or optimal if it satisfies the accuracy requirements (in terms of the respective tolerance parameters). On the other hand, exact solutions are required in some situations. Concerning the solution of linear programs some authors [14, 93] describe exact implementations of the simplex method using rational arithmetic. Fukasawa and Goycoolea [91] study the exact separation of mixed-integer knapsack cuts. With respect to running times they report that exact computations are on average 100 times slower than floating point computations.

All implementations discussed in this thesis are based on floating point arithmetic. As mentioned above, tolerances are used when checking two floating point numbers for equality or strict inequality in order to cope with rounding errors. Similarly, an integrality tolerance is used to decide whether a variable takes an integral value. A robust implementation of a cutting plane separation algorithm particularly has to take into account that its input data is subject to an accumulated rounding error. Concerning cutting planes derived from the simplex tableau, the computation of the rows of the tableau involves a large number of arithmetic operations. Numerical inaccuracies might be introduced during the elimination procedure in LP preprocessing and in the computation of the LU factorization of the basis matrix. Thus the numerical robustness of algorithms which generate cutting planes from rows of the simplex tableau is to some extent directly dependent on the numerical robustness of the underlying LP solver.

In our code we use two techniques to increase the numerical stability of the generated cuts. We remove variables having very small (quasi-zero) coefficients from the generated cuts by substituting their lower or upper bounds for them respectively, thus relaxing the cut. If this is not possible, the cut is rejected. Moreover, we compute the *dynamism* of each cut, i.e. the ratio between the absolute values of the largest and smallest non-zero coefficient in the cut (see Margot [136]). If the dynamism is larger than a threshold, the cut is rejected as well.

Chapter 8.

Single-Row Cutting Plane Separators

In this chapter we describe separation algorithms for various general-purpose single-row cutting planes. We discuss our implementation of these algorithms and highlight important details making them efficient in practice. We also propose a novel algorithm for improving the performance of the Gomory mixed-integer cuts. To assess the effectiveness of the discussed implementations we perform benchmarks on a number of publicly available test sets. Thus this chapter is an implementation-oriented complement to Chapter 4.

This chapter is organized as follows. In Section 8.1 we detail our implementation of the Gomory mixed-integer cuts. We discuss k-cuts in Section 8.2 and combined Gomory mixed-integer cuts in Section 8.3. We treat reduce-and-split cuts in Section 8.4 and present technical details of our implementation of the lift-and-project method in Section 8.5. Section 8.6 introduces a new pivoting procedure for strengthening the Gomory mixed-integer cuts. We provide details of our implementation of the strong Chvátal-Gomory cuts and $\{0, \frac{1}{2}\}$-cuts in Section 8.7 and 8.8 respectively. Finally, in Section 8.9 we report on computational experiments indicating the effectiveness of our implementations.

8.1. Gomory Mixed-Integer Cuts

In this section we outline our implementation of the Gomory mixed-integer cuts. A detailed discussion of the theory of GMI cuts and their connection to intersection and split cuts can be found in Section 4.4.3.

The main steps in the separation of a GMI cut are shown in Algorithms 8.1 and 8.2. In Step 1 of Algorithm 8.1 a set of basic integer variables which take

Algorithm 8.1. A round of Gomory mixed-integer cuts

```
        Input: A basis $B$ and the corresponding basic solution $x^*$ of the
               LP relaxation of the IMR (7.2).
        Output: A list $\mathcal{L}$ of GMI cuts.
(Step 1) Initialize
        Set $\mathcal{L} := \emptyset$ and $k := 0$.
        Choose a subset $S$ of the basic integer variables which have a
        fractional value, i.e. $S \subseteq \{j \in B_I : x^*_j \notin \mathbb{Z}\}$.
(Step 2) Select a variable
        Set $k := k + 1$.
        if $|S| < k$ then exit.
        Otherwise, let $i$ be the $k^{th}$ element of $S$.
(Step 3) Compute the row of the simplex tableau
        Compute the row of the simplex tableau (8.3) associated with $x_i$,
        i.e. $x_i + \sum_{j \in J} \bar{a}'_{ij} x_j = 0$.
(Step 4) Separate the cut
        Compute a GMI cut from this tableau row using Algorithm 8.2.
(Step 5) Store the cut
        Add the cut to the list $\mathcal{L}$ and goto Step 2.
```

a fractional value in the solution x^* to the LP relaxation of the IMR (7.2) is selected. The tableau rows associated with these variables are then used to derive GMI cuts. In our implementation we restrict cut generation to the set $B_I \cap \{1, \ldots, n\}$ of structural basic integer variables. While deriving GMI cuts from tableau rows associated with logical basic integer-constrained variables is possible, such variables are mostly only present in pure integer programs. Moreover, preliminary computational experiments lead to the conclusion that the GMI cuts generated from these tableau rows are not effective. Concerning the set of structural basic integer variables the two main questions are which and how many of these variables should be used for cut generation. While cut generation is likely to be fast on small problems, it can be very expensive on large problems. To rank the variables we compute the fractional part $f_{i0} = x^*_i - \lfloor x^*_i \rfloor$ for all $i \in B_I \cap \{1, \ldots, n\}$. Note that the value f_{i0} is precisely the right-hand side of the GMI cut (see Equation (4.47)) and also represents its violation. A small value of f_{i0} corresponds to a GMI cut which is only slightly violated whereas a small value of $1 - f_{i0}$ leads to a significant increase in some of the coefficients in the GMI cut. Accordingly, we only accept variables satisfying

$$f_{i0} \in \,]\epsilon, 1-\epsilon[\,, \quad i \in B_I \cap \{1, \ldots, n\}. \tag{8.1}$$

The tolerance parameter ϵ bounds the fractionalities. In our code we set ϵ to 0.05. The number of variables satisfying (8.1) can still be too large to compute GMI cuts for all of them. Thus we sort the remaining variables in non-ascending order of their fractional parts f_{i0} and store them in a stack. We only compute cuts for a number of the most fractional variables.

The next steps of the algorithm are the selection of a variable and the construction of the associated row of the simplex tableau (see Step 2 and 3 of Algorithm 8.1). Given a basis B of the LP relaxation of the IMR (7.2), the simplex tableau reads

$$A_B x_B + A_J x_J = 0, \tag{8.2a}$$

$$x_B = -\left(A_B^{-1} A_J\right) x_J, \tag{8.2b}$$

where $A = (A_B, A_J)$ and $x = (x_B, x_J)$. State-of-the-art simplex engines do not explicitly work on the simplex tableau. Let $B(i)$ denote the position of the variable x_i in the basis B. The row of the simplex tableau associated with a basic variable x_i is calculated by multiplying the $B(i)^{th}$ row of the basis inverse A_B^{-1} with the matrix A. Let the result of this multiplication be

$$x_i + \sum_{j \in J} \bar{a}'_{ij} x_j = 0, \tag{8.3}$$

where $i \in B$.

The row of the tableau (8.3) contains non-basic variables which are either at their upper or lower bound. In order to derive a GMI cut all non-basic variables have to be transformed such that they are at their lower bound of zero. We partition the set of non-basic variables J into (J^l, J^d) such that J^d contains the indices of the non-basic variables which are at their upper bound and J^l consists of the indices of the non-basic variables which are at their lower bound. Let

$$x_j = \begin{cases} s_j + l_j & \text{if } j \in J^l, \\ d_j - s_j & \text{if } j \in J^d, \end{cases} \quad \text{and} \quad \bar{a}_{ij} = \begin{cases} \bar{a}'_{ij} & \text{if } j \in J^l, \\ -\bar{a}'_{ij} & \text{if } j \in J^d, \end{cases} \tag{8.4}$$

for $j \in J$. By complementing the variables in J^d and shifting the variables in J^l we obtain the transformed tableau row

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j, \tag{8.5}$$

where the variables s are the surplus variables from the constraints $x_j \geq l_j$ for $j \in J^l$ and slack variables from the constraints $x_j \leq d_j$ for $j \in J^d$, and

$$\bar{a}_{i0} = -\sum_{j \in J^d} \bar{a}'_{ij} d_j - \sum_{j \in J^l} \bar{a}'_{ij} l_j. \tag{8.6}$$

This transformation is also performed in Step 1 of Algorithm 8.2.

In Step 2 the GMI cut is then generated using Proposition 4.7. However, in our code we generate a *complemented mixed-integer rounding cut* from the untransformed row (8.3). Although MIR and GMI cuts are theoretically equivalent, there are some algorithmic differences. One of the main reasons for generating MIR cuts instead of GMI cuts is that an efficient and well-tested implementation of the MIR separation heuristic (see Marchand and Wolsey [135]) is available in Mops. The MIR separation heuristic consists of the steps aggregation, bound substitution and separation. The aggregation heuristic searches for linear combinations of the constraints of the problem which give a violated MIR cut. It can be easily adapted to construct rows of the simplex tableau. In the second phase of the algorithm the bound substitution heuristic transforms the aggregated constraint (tableau row) in a way similar to that discussed above. Moreover, it allows for using so-called variable bound constraints

$$l_k x_k \leq x_j \leq d_l x_l, \tag{8.7}$$

```
Algorithm 8.2. Separating a Gomory mixed-integer cut
          Input: A row of the simplex tableau $x_i + \sum_{j\in J} \bar{a}'_{ij} x_j = 0$
                 associated with a basis $B$ of the LP relaxation of the
                 IMR (7.2) where $x_i$ is a basic fractional integer variable.
          Output: A GMI cut $\alpha x \geq \beta$.
(Step 1)  Shift and complement
          (Shift.) Set $\bar{a}_{ij} := \bar{a}'_{ij}$ and $x_j := l_j + s_j$ for all $j \in J^l$.
          (Complement.) Set $\bar{a}_{ij} := -\bar{a}'_{ij}$ and $x_j := d_j - s_j$ for all $j \in J^d$.
          (Compute right-hand side.) Set $\bar{a}_{i0} := -\sum_{j\in J^d} \bar{a}'_{ij} d_j - \sum_{j\in J^l} \bar{a}'_{ij} l_j$.
          The transformed tableau row has the form $x_i + \sum_{j\in J} \bar{a}_{ij} s_j = \bar{a}_{i0}$.
(Step 2)  Compute the GMI cut
          Compute the GMI cut $\sum_{j\in J} \alpha'_j s_j \geq \beta'$ (see Proposition 4.7).
(Step 3)  Unshift and uncomplement
          (Unshift.) Set $\alpha_j := \alpha'_j$ and $s_j := x_j - l_j$ for all $j \in J^l$.
          (Uncomplement.) Set $\alpha_j := -\alpha'_j$ and $s_j := d_j - x_j$ for all $j \in J^d$.
          (Compute right-hand side.) Set $\beta := \beta' - \sum_{j\in J^d} \alpha'_j d_j + \sum_{j\in J^l} \alpha'_j l_j$.
          The GMI cut reads $\sum_{j\in J} \alpha_j x_j \geq \beta$.
(Step 4)  Remove logicals
          Use the definition of the logical variables $x_L = -\tilde{A} x_S$ to write the
          cut in the space of the structural variables $x_S$.
```

where x_j with $j \in N \setminus N_I$ is a continuous variable and x_k, x_l with $k, l \in N_I$ are integer variables. More precisely, the bound substitution heuristic substitutes a continuous variable x_j either by the standard bounds as shown in Equation (8.4) or by the variable bounds

$$x_j = \begin{cases} l_k x_k + s_j & \text{or} \\ d_l x_l - s_j. \end{cases} \tag{8.8}$$

There are different criteria for deciding whether to use a variable lower or upper bound (see [51, 135]). An integer variable x_k or x_l which is substituted into a tableau row needs to be treated again by bound substitution since it may have an arbitrary lower or upper bound. Moreover, note that the algorithm has to remember whether a simple or variable bound (lower or upper bound) was substituted for a variable.

We then generate a cut using the separation heuristic. All the variables in the resulting GMI cut are slack or surplus variables s from the simple and variable bound constraints. To write the cut in the space of the x variables only we re-substitute for s. Specifically, depending on which bound was used in the initial transformation, we have $s_j = x_j - l_j$, $s_j = d_j - x_j$ from the simple bounds or $s_j = x_j - l_k x_k$, $s_j = d_l x_l - x_j$ from the variable bounds. In addition, the GMI cut has to be written in the structural variables x_S so as to add it to the matrix $\tilde{A}$ in the IMR (7.2). This is accomplished by using the definition of the logical variables $x_L = -\tilde{A}x_S$. The final cut is saved in an intermediate data structure. After GMI cuts have been generated for the selected basic variables all of the cuts are typically added to the LP relaxation at the same time. This set of cuts is also called a *round* of cuts and Algorithm 8.1 is said to generate cuts in rounds.

8.2. K-Cuts

In Section 4.4.4 we introduced k-cuts. A k-cut is a GMI cut generated from the tableau row

$$kx_i = k\bar{a}_{i0} - \sum_{j \in J} (k\bar{a}_{ij})\, s_j, \tag{8.9}$$

where $k \in \mathbb{Z}$ and $k \neq 0$. Equation (8.9) is simply the transformed tableau row (8.5) multiplied by k. The motive for the multiplication with k is to obtain GMI cuts with different violations. On the other hand, the coefficients of the (non-basic) continuous variables in a GMI cut are increased by a factor of k. Integrating k-cuts into an existing separation routine for GMI cuts is quite easy. We stick to the strategy for generating GMI cuts we discussed in Section 8.1. The main question is which values should be used for k. In our implementation we generate k-cuts for $k = 2^i$ with $i = 0, \ldots, 3$.

Finally, we offer some comments on the integrality requirement on k. The integrality of k is required for the validity of the k-cuts. For instance, consider the disjunction $(kx_i \leq \lfloor k\bar{a}_{i0} \rfloor) \vee (kx_i \geq \lfloor k\bar{a}_{i0} \rfloor + 1)$ on a basic integer variable x_i and let $\bar{a}_{i0} = \frac{1}{2}$ and $k = \frac{3}{4}$. We obtain $x_i \leq 0 \vee x_i \geq \frac{4}{3}$ which is not a valid disjunction. Moreover, an integral value of k produces an integral coefficient on the basic variable in the tableau row. In turn an integer variable with an integral coefficient k in the tableau row also obtains an integral coefficient $\alpha_i = k$ in the GMI or MIR cut. Thus the violation of the GMI cut is $f_{i0} = k\bar{a}_{i0} - \lfloor k\bar{a}_{i0} \rfloor > 0$. However, for non-integral multipliers k the situation is different. Choosing a non-integral multiplier k produces the coefficient α_i in the GMI cut which is dependent on the size of $k - \lfloor k \rfloor$ and f_{i0}. In some situations the GMI cut may not be violated, i.e. $\alpha_i\bar{a}_{i0} - \lfloor k\bar{a}_{i0} \rfloor \leq 0$, even though $k\bar{a}_{i0} \notin \mathbb{Z}$. For the same reasons the separation algorithms for combined Gomory mixed-integer cuts and reduce-and-split cuts also compute integral multipliers for the tableau rows.

8.3. Combined Gomory Mixed-Integer Cuts

In Section 4.4.5 we outlined an approach for generating GMI cuts from linear combinations of the rows of the simplex tableau. The idea behind this approach is to find a combined tableau row which gives a GMI cut with a maximal violation and minimal coefficients on certain non-basic integer variables.

We generate combined GMI cuts in rounds using an algorithm similar to Algorithm 8.1. The main input parameter for this algorithm is the number r of tableau rows to be combined. The larger this number is, the more computationally expensive the separation becomes. In our code we consider three tableau rows at a time, i.e. $r = 3$. This number of rows is also suggested Ceria et al. [46]. To avoid expensive re-computation, we store the rows of the simplex tableau associated with structural basic integer variables in a matrix T. The data structure used to store this matrix is shown in Figure 7.3. Following the ideas described in Section 8.1 we generate cGMI cuts for the most fractional structural basic integer variables. More precisely we select r structural basic integer variables in the order of decreasing fractionality. Once these variables are selected the associated rows of the simplex tableau (8.3) need to be computed. We extract these rows from the matrix T.

Algorithm 8.3 sketches the separation of a cGMI cut. We shall only briefly outline the algorithm since it was already discussed in Section 4.4.5. The input data for the algorithm are the selected rows of the simplex tableau. The first step of the algorithm is to write the coefficients of the simplex tableau and the solution values in rational numbers where D is the common denominator. We use the

Algorithm 8.3. Separating a combined Gomory mixed-integer cut

```
Input: The rows of the simplex tableau $x_i + \sum_{j\in J} \bar{a}'_{ij} x_j = 0$
       associated with a basis $B$ of the LP relaxation of the
       IMR (7.2) where $x_i$ is a basic fractional integer variable for
       $i = 1, \ldots, r$.
Output: A combined GMI cut $\alpha x \geq \beta$.
(Step 1) Compute rational representation
    Compute a rational representation $x_i + \sum_{j\in J}(\frac{e_{ij}}{D})x_j = 0$ and
    $x_i^* = \frac{e_{i0}}{D}$ for $i = 1, \ldots, r$.
(Step 2) Maximize $\sum_{i=1}^r \pi_i(\frac{e_{i0}}{D}) - \lfloor \sum_{i=1}^r \pi_i(\frac{e_{i0}}{D}) \rfloor$
    Find a family of multipliers $p_i = p_i^1 + \sum_{k=2}^{r+1} p_i^k y_k$ for $i = 1, \ldots, r$
    solving the diophantine equation
    $< e_{10}, e_{20}, \ldots, e_{r0}, D > = e_{10}p_1 + e_{20}p_2 + \ldots + e_{r0}p_r + qD$.
    Set $\pi_i := D - p_i = D - p_i^1 - \sum_{k=2}^{r+1} p_i^k y_k$ for $i = 1, \ldots, r$.
(Step 3) Check termination criteria
    if $p_i^k \equiv 0 \pmod D$ for all $i = 1, \ldots, r$ and $k = 2, \ldots, r+1$ then
        goto Step 6.
(Step 4) Select a non-basic integer variable
    Select a non-basic integer variable $x_j$, $j \in J_I$, whose coefficient was
    not optimized so far.
(Step 5) Minimize $\sum_{i=1}^r \pi_i(\frac{e_{ij}}{D}) - \lfloor \sum_{i=1}^r \pi_i(\frac{e_{ij}}{D}) \rfloor$
    Substitute for the family of multipliers $\pi$ and rewrite the result in
    the form (4.69), i.e. $\frac{e_0}{D} + \sum_{k=2}^{r+1}(\frac{e_k}{D})y_k - \lfloor \frac{e_0}{D} + \sum_{k=2}^{r+1}(\frac{e_k}{D})y_k \rfloor$.
    Find a family of multipliers $p_i = p_i^1 + \sum_{k=2}^{r+1} p_i^k y'_k$ for
    $i = 2, \ldots, r+1$ solving the diophantine equation
    $< e_2, e_3, \ldots, e_{r+1}, D > = e_2p_2 + e_3p_3 + \ldots + e_{r+1}p_{r+1} + qD$.
    Set $y_k := -\lfloor \frac{e_0}{<e_2,e_3,\ldots,e_{r+1},D>} \rfloor p_k$ for $k = 2, \ldots, r+1$ and obtain a
    new improved family of multipliers $\pi_i := D - p_i^1 - \sum_{k=2}^{r+1} p_i^k y'_k$.
    Goto Step 3.
(Step 6) Compute the combined GMI cut
    Construct the combined tableau row
    $\sum_{i=1}^r \pi_i x_i + \sum_{j\in J}(\sum_{i=1}^r \pi_i \bar{a}'_{ij})x_j = 0$.
    Compute a GMI cut $\alpha x \geq \beta$ using Algorithm 8.2.
```

Euclidean algorithm to compute the rational representation of the tableau rows. In Step 2 of the algorithm a diophantine equation is solved in order to compute a family of multipliers which maximizes the right-hand side of the cGMI cut. As noted in Section 4.4.5 we use the algorithm of Rosser [151] to solve such type of equations. The study of other algorithms that could be used for this purpose such as the LLL algorithm [128] is left to future work.

In the second part of the algorithm the left-hand side of the cGMI cut is optimized. More specifically a non-basic integer variable is selected in Step 4 and the algorithm seeks to find a new family of multipliers which maximizes the right-hand side of the GMI cut and minimizes the coefficient of this variable in Step 5. To obtain the new family of multipliers a second diophantine equation is solved and a set of integers y_k is computed using Proposition 4.11. The assumption of Proposition 4.11 is not, however, always satisfied. As suggested by Ceria et al. [46] we perform a simple rounding and nevertheless apply Proposition 4.11. We thus only approximately minimize the coefficient if Proposition 4.11 is not applicable. The process is then iterated, i.e. the algorithm selects the next non-basic integer variable and optimizes its coefficient. The algorithm stops when the termination criterion in Step 3 is satisfied. If the integers p_i^k are all zero modulo D, all products $p_i^k y_k$ will also always be zero modulo D regardless of how the integers y_k are chosen. It is therefore not possible to optimize an additional coefficient. The procedure is stopped and the cGMI cut is generated (see Step 6).

Finally we point to the weaknesses of the above procedure. A first major drawback is that the number of coefficients on the left-hand side of a GMI cut that can be optimized is constrained by the number of tableau rows involved. A resulting cut is therefore likely to have strong coefficients on a few variables and large, weak coefficients on the majority of the non-basic variables. Secondly, while the transformation of all entries of a simplex tableau into a rational representation can theoretically always be obtained by choosing a sufficiently large D, it is cumbersome from a computational point of view. The size of the multipliers π is proportional to the size of D. Large multipliers produce large coefficients on variables which are not optimized by the procedure. In particular, recall that the procedure does not at all take into consideration the coefficients of the continuous non-basic variables in the cGMI cut. In our code we set $D = 100$ and round the coefficients of the rows of the simplex tableau accordingly.

8.4. Reduce-and-Split Cuts

In this section we detail our implementation of the reduce-and-split cuts. R&S cuts aim at reducing the size of the coefficients of the continuous variables in the GMI cuts. This is accomplished by an algorithm which constructs integer linear combinations of the rows of the simplex tableau. We call this algorithm the R&S reduction algorithm. For a discussion on the theory of R&S cuts we refer the reader to Section 4.4.6. The separation of R&S cuts is presented in Algorithm 8.4.

A set S of variables is selected in Step 1. The R&S reduction algorithm is carried out on the tableau rows associated with these variables. In our code we select the complete set $B_I \cap \{1, \dots, n\}$ of structural basic integer variables. Then the tableau rows associated with these variables are computed. To save computational effort we simply deploy the matrix T of tableau rows we constructed for the separation of the combined GMI cuts (see Section 8.3). The compact data structure which is used to store this matrix is shown in Figure 7.3.

The R&S reduction algorithm performs row operations on the matrix T. As the tableau rows stored in T are of the form (8.3) we do not have to consider right-hand sides in any part of the algorithm. The compact storage of T has the drawback that row operations can not be carried out efficiently directly on T. Most notably, it is not possible to determine whether a row contains a variable without iterating over the array of variable indices (similar to a packed vector). As an alternative one could store the matrix T in a dense form where each of the rows is a dense vector as shown in Figure 7.4(a). However, iterating over the elements of a dense matrix is a computationally very expensive operation. Accordingly, whenever the algorithm intends to perform an operation on two rows, we load these rows into an indexed data structure (see Figure 7.4(c)). Using this data structure the row operations can be implemented very efficiently. Also the action of replacing a row of the matrix T by its reduced version can be implemented very efficiently using the free space area and the pointers for the length and offset of a row. When the free space area is exhausted a compression routine is executed which re-locates the rows inside the data structure with the objective of regaining free space.

In Step 2 of Algorithm 8.4 the actual R&S reduction algorithm is performed. Given two basic integer variables x_i and x_k and the associated rows of the simplex

Algorithm 8.4. Separating reduce-and-split cuts

```
Input: A basis $B$ of the LP relaxation of the IMR (7.2) and the
       corresponding basic solution $x^*$.
Output: A list $\mathcal{L}$ of reduce-and-split cuts.

(Step 1) Construct the tableau
Select a set $S$ of the structural basic integer variables. Compute
the tableau rows $x_i + \sum_{j \in J} \bar{a}'_{ij} x_j = 0$ for all $i \in S$ and store them
in a matrix $T$. Set $done := false$.

(Step 2) Perform the reduction algorithm
while $done = false$ do
    Set $done := true$.
    for $i = 1, \dots, |S| - 1$ do
        for $k = i + 1, \dots, |S|$ do
            (Get tableau rows.) Let $x_i + \sum_{j \in J} \bar{a}'_{ij} x_j = 0$ and
            $x_k + \sum_{j \in J} \bar{a}'_{kj} x_j = 0$ be the $i^{th}$ and $k^{th}$ row of $T$.
            (Check for a reduction.) Let $\delta^*$ minimize
            $f(\delta) = \sum_{j \in J \setminus J_I} (\bar{a}'_{ij} + \delta \bar{a}'_{kj})^2$ with $\delta \in \mathbb{Z}$.
            if $f(\delta^*) < f(0)$ then
                (Update.) Set $x_i := x_i + \delta^* x_k$ and $\bar{a}'_{ij} := \bar{a}'_{ij} + \delta^* \bar{a}'_{kj}$
                for all $j \in J$.
                (Replace.) Replace the $i^{th}$ row of $T$ by the updated
                row and set $done := false$.
            end
            (Check for a reduction.) Let $\delta^*$ minimize
            $f(\delta) = \sum_{j \in J \setminus J_I} (\bar{a}'_{kj} + \delta \bar{a}'_{ij})^2$ with $\delta \in \mathbb{Z}$.
            if $f(\delta^*) < f(0)$ then
                (Update.) Set $x_k := x_k + \delta^* x_i$ and $\bar{a}'_{kj} := \bar{a}'_{kj} + \delta^* \bar{a}'_{ij}$
                for all $j \in J$.
                (Replace.) Replace the $k^{th}$ row of $T$ by the updated
                row and set $done := false$.
            end
        end
    end
end

(Step 3) Compute reduce-and-split cuts
Iterate over the matrix $T$ and compute a reduce-and-split cut for
each row using Algorithm 8.2. Add these cuts to the list $\mathcal{L}$.
```

tableau in the form (8.3), the algorithm evaluates if the combined tableau row associated with $x_i + \delta x_k$ has smaller coefficients on the continuous variables. More formally, we minimize the function

$$f(\delta) = \sum_{j \in J \setminus J_I} \left(\bar{a}'_{ij} + \delta \bar{a}'_{kj}\right)^2, \tag{8.10a}$$

$$= \sum_{j \in J \setminus J_I} \left(\left(\bar{a}'_{ij}\right)^2 + 2\delta \bar{a}'_{ij}\bar{a}'_{kj} + \left(\delta \bar{a}'_{kj}\right)^2\right), \tag{8.10b}$$

$$= \sum_{j \in J \setminus J_I} \left(\bar{a}'_{ij}\right)^2 + 2\delta \sum_{j \in J \setminus J_I} \bar{a}'_{ij}\bar{a}'_{kj} + \delta^2 \sum_{j \in J \setminus J_I} \left(\bar{a}'_{kj}\right)^2. \tag{8.10c}$$

The first and last terms of Equation (8.10c) contain the squared Euclidean norm of the coefficients of the continuous variables of the tableau rows. Therefore before starting the algorithm we calculate this norm for all tableau rows in the matrix T and store the norms in a separate data structure. The second term of Equation (8.10c) contains the inner product of two rows on the continuous variables. When the algorithm selects two rows in Step 2 these rows are loaded into an indexed data structure and we compute the inner product. Recall that the optimal solution minimizing $f(\delta)$ is either

$$\delta^* = -\left\lfloor \frac{\sum_{j \in J \setminus J_I} \bar{a}'_{ij}\bar{a}'_{kj}}{\sum_{j \in J \setminus J_I} \left(\bar{a}'_{kj}\right)^2} \right\rfloor \quad \text{or} \quad \delta^* = -\left\lceil \frac{\sum_{j \in J \setminus J_I} \bar{a}'_{ij}\bar{a}'_{kj}}{\sum_{j \in J \setminus J_I} \left(\bar{a}'_{kj}\right)^2} \right\rceil. \tag{8.11}$$

These optimal values can be calculated easily from the aforementioned data. To speed up the algorithm and to avoid large multipliers δ^* (i.e. large coefficients in the split disjunction, see Equation (8.11)), we stop considering rows if their norm is not larger than a threshold. Specifically, if

$$\sum_{j \in J \setminus J_I} \left(\bar{a}'_{ij}\right)^2 \leq \epsilon, \tag{8.12}$$

we skip the row associated with the basic variable x_i in Step 2 of the algorithm. In our code we set ϵ to 10^{-5}. Moreover, to achieve a faster convergence of the algorithm, we do not accept linear combinations $x_i + \delta^* x_k$ which only slightly reduce the size of the coefficients of the continuous variables. We reject a linear combination $x_i + \delta^* x_k$ if the percentage reduction satisfies

$$1 - \frac{f(\delta^*)}{f(0)} \leq \epsilon, \tag{8.13}$$

where ϵ is set to 0.05 (i.e. 5%) in our implementation. The algorithm performs several iterations (see Step 2). It terminates if no improving reduction is found in an iteration. Further to enhance the speed of the algorithm we store the iteration in which a reduction on a specific row was performed. Similarly, we store the iteration in which a combination of two rows was last tried. Thus we avoid checking the same (unchanged) rows for a reduction over and over again.

In Step 3 the R&S cuts are generated. Note that our implementation is a slight variation of the R&S cuts as described in [8]. After the rows of the tableau have been reduced by forming the linear combinations, Andersen et al. [8] strengthen the underlying split disjunctions and then generate intersection cuts. In our implementation we directly generate GMI cuts from the reduced rows of the simplex tableau.

8.5. Lift-and-Project Cuts

In this section we discuss our implementation of the Balas-Perregaard algorithm [30] for generating lift-and-project cuts. The theory of L&P cuts is presented in Section 4.4.7. For a very detailed discussion of L&P cuts and their implementation we refer the reader to Perregaard [148].

We assume that an MIP is given in the IMR (7.2). As in Section 4.4.7 we denote by (CGLP$_i$) the cut generating linear program (4.82) and by (LP) the LP relaxation of the IMR (7.2). We generate L&P cuts in the same fashion as Gomory mixed-integer cuts (see Section 8.1). Algorithm 8.5 shows the main steps in the separation of L&P cuts.

8.5.1. Working with Bounded Variables

The correspondence between solving (CGLP$_i$) and performing pivots on (LP) relies on the fact that bounds are explicitly stored in the constraint set such that

Algorithm 8.5. Separating a lift-and-project cut

```
Input: A basis $B$ of the LP relaxation of the IMR (7.2) and the
       corresponding basic solution $x^*$. A basic integer variable $x_i$
       which has a fractional value $x_i^*$. The parameter
       $max\_pivots$ indicating the maximum number of pivots.
Output: An L&P cut $\alpha x \geq \beta$.
(Step 1) Initialize
         Set $num\_pivots := 0$.
(Step 2) Check termination criteria
         if $num\_pivots \geq max\_pivots$ then goto Step 8.
(Step 3) Compute the tableau row
         Compute the transformed tableau row (8.5) associated with $x_i$, i.e.
         $x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j$.
         if $x_i$ is a general-integer variable then set $\bar{a}_{i0} := \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor$.
(Step 4) Compute the partition $J = (M_1, M_2)$
         Let $M_1 := \{j \in J : \bar{a}_{ij} < 0\}$, $M_2 := \{j \in J : \bar{a}_{ij} > 0\}$ and
         $M_3 := \{j \in J : \bar{a}_{ij} = 0\}$.
         Assign the variables in $M_3$ to the sets $M_1$ or $M_2$ at random.
(Step 5) Find the leaving variable
         Compute the reduced cost $r^l_{u_k}, r^l_{v_k}, r^d_{u_k}, r^d_{v_k}$ for all $k \notin J \cup \{i\}$.
         Let $k := \arg\min_{t \notin J \cup \{i\}} \{r^l_{u_t}, r^l_{v_t}, r^d_{u_t}, r^d_{v_t}\}$.
         if $\min\{r^l_{u_k}, r^l_{v_k}, r^d_{u_k}, r^d_{v_k}\} \geq 0$ then goto Step 8.
(Step 6) Find the incoming variable
         Let $J' = \{j \in J : |\bar{a}_{kj}| \geq \epsilon\}$ be the set of admissible pivots.
         Let $J^- = J' \cap \{j \in J : \gamma_j = -\frac{\bar{a}_{ij}}{\bar{a}_{kj}} < 0\}$ and $J^+ = J' \setminus J^-$.
         Let $p := \arg\min\{\arg\min_{j \in J^+}\{f^+(\gamma_j)\}, \arg\min_{j \in J^-}\{f^-(\gamma_j)\}\}$.
(Step 7) Perform the pivot
         Pivot the variable $x_k$ out of the basis and pivot the variable $x_p$ into
         the basis $B$. Set $B := (B \cup \{p\}) \setminus \{k\}$ and $J := (J \cup \{k\}) \setminus \{p\}$.
         Set $num\_pivots := num\_pivots + 1$ and goto Step 2.
(Step 8) Compute the L&P cut
         Compute the row of the simplex tableau (8.3) associated with $x_i$
         and generate the L&P cut $\alpha x \geq \beta$ using Algorithm 8.2.
```

a row of an (LP) tableau row can exclusively be written in the space of the slack or surplus variables respectively. The theory assumes that all structural variables are unrestricted. However, in state-of-the-art MIP solvers bounds on variables are stored separately from the constraint matrix. Reconsider the transformed row of the simplex tableau

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j, \tag{8.14}$$

where s_j are slack or surplus variables of the bound constraints $x_j \leq d_j$ and $x_j \geq l_j$. The transformation ensures that all non-basic variables are slack or surplus variables which are at their lower bound. To be able to handle a basic variable x_k which is bounded, we can again use the corresponding slack or surplus variables from the bound constraints. We get

$$s_k^l = (\bar{a}_{k0} - l_k) - \sum_{j \in J} \bar{a}_{kj} s_j, \tag{8.15a}$$

$$s_k^d = (d_k - \bar{a}_{k0}) - \sum_{j \in J} (-\bar{a}_{kj}) \, s_j. \tag{8.15b}$$

Pivoting the surplus variable s_k^l of the lower bound constraint $x_k - s_k^l = l_k$ out of the basis is equivalent to making x_k non-basic at its lower bound l_k. Thus we have to transform the tableau row into the form (8.15a). On the other hand, pivoting the slack variable s_k^d of the upper bound constraint $x_k + s_k^d = d_k$ out of the basis is equivalent to making x_k non-basic at its upper bound d_k. In this case we have to transform the tableau row into the form (8.15b).

8.5.2. Handling General Integer Variables

In Step 3 of Algorithm 8.5 the row of the simplex tableau associated with the selected basic integer variable x_i is computed. The procedure for generating L&P cuts as outlined in Section 4.4.7 is designed exclusively to deal with 0-1 variables. Only cuts for fractional 0-1 variables are generated, i.e. x_i with $0 < \bar{a}_{i0} < 1$.

By rewriting the row of the simplex tableau (8.14) associated with a fractional general-integer variable x_i as

$$\begin{aligned} x_i - \lfloor \bar{a}_{i0} \rfloor &= \underbrace{(\bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor)} - \sum_{j\in J} \bar{a}_{ij} s_j, \\ x'_i &= \qquad f_{i0} \qquad - \sum_{j\in J} \bar{a}_{ij} s_j, \end{aligned} \tag{8.16}$$

we obtain a form that the method can handle because $0 < f_{i0} < 1$. Basically we derive L&P cuts from the elementary disjunction

$$(-x_i \geq -\lfloor \bar{a}_{i0} \rfloor) \vee (x_i \geq \lceil \bar{a}_{i0} \rceil)\,. \tag{8.17}$$

In the remainder of this section we shall for simplicity assume that $x_i := x_i - \lfloor \bar{a}_{i0} \rfloor$, $\bar{a}_{i0} := \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor$ and $x^*_i := x^*_i - \lfloor x^*_i \rfloor$ if the reference row i of the simplex tableau used for cut generation is associated with a basic general-integer variable x_i.

8.5.3. Computing the Reduced Cost

We next consider the computation of the reduced cost which is of crucial importance in finding the variable which leaves the basis (see Step 5 of Algorithm 8.5). We assume that the general normalization constraint (4.98) is given (see Section 4.4.7). As we have the choice either to pivot x_k to its lower bound l_k or to its upper bound d_k, we have slightly to modify the computation of the reduced cost (see Equation (4.99)). We get

$$r^l_{u_k} = -\sigma\lambda_k + (\bar{a}_{k0} - l_k)(1 - x^*_i) - \tau_k, \tag{8.18a}$$
$$r^l_{v_k} = -\sigma\lambda_k - (\bar{a}_{k0} - l_k)(1 - x^*_i) + (x^*_k - l_k) + \tau_k, \tag{8.18b}$$
$$r^d_{u_k} = -\sigma\lambda_k + (d_k - \bar{a}_{k0})(1 - x^*_i) + \tau_k, \tag{8.18c}$$
$$r^d_{v_k} = -\sigma\lambda_k - (d_k - \bar{a}_{k0})(1 - x^*_i) + (d_k - x^*_k) - \tau_k, \tag{8.18d}$$

where, as in Section 4.4.7,

$$\sigma = \frac{\sum\limits_{j\in M_2} \bar{a}_{ij} s^*_j - \bar{a}_{i0}(1 - x^*_i)}{1 + \sum\limits_{j\in J} |\bar{a}_{ij}|\,\lambda_j}, \tag{8.19}$$

and

$$\tau_k = \sum_{j \in M_1} \sigma \bar{a}_{kj} \lambda_j + \sum_{j \in M_2} \left(s_j^* - \sigma \lambda_j \right) \bar{a}_{kj}. \tag{8.20}$$

A specialized simplex code is necessary efficiently to separate L&P cuts in the framework of Balas and Perregaard. A point which is of crucial importance is a fast computation of the reduced cost, particularly the values τ_k. One way to compute the values τ_k is as follows. First, recompute the (LP) tableau row associated with x_k and then perform the calculations based on the partitioning (M_1, M_2) of the entries of the tableau row as shown in Equation (8.20). As this approach involves computing a large number of tableau rows, it is computationally very expensive. However, observe that each tableau row is multiplied with the same vector. We can hence rewrite the expression for τ_k in the form

$$\begin{aligned} \tau_k &= \sum_{j \in J} \bar{a}_{kj} y_j, \\ &= \sum_{j \in J} \left(A_B^{-1} A_J \right)_{kj} y_j, \\ &= \sum_{j \in J} \left(A_{B(k)}^{-1} \left(A_J \right)_j \right) y_j, \\ &= A_{B(k)}^{-1} \sum_{j \in J} \left(A_J \right)_j y_j, \end{aligned} \tag{8.21}$$

where $B(k)$ is the position of x_k in the basis B, $A_{B(k)}^{-1}$ is the row of the basis inverse associated with x_k, $(A_J)_j$ is the j^{th} non-basic column of A, $y_j = \sigma \lambda_j$ for $j \in M_1$ and $y_j = s_j^* - \sigma \lambda_j$ for $j \in M_2$. To obtain the vector τ, i.e. the value of τ_k for all (LP) tableau rows all at once, we have to solve the system

$$A_B \tau = \sum_{j \in J} \left(A_J \right)_j y_j \tag{8.22}$$

for τ. Fortunately, solving this system is a standard operation in state-of-the-art simplex engines. It involves one forward transformation operation (or ftran) which is carried out in highly efficient manner (see Chvátal [52]).

Recall that in the presence of zero elements in row i the partition $J = (M_1, M_2)$ of the non-basic variables is not unique. Let $M_3 = \{j \in J : \bar{a}_{ij} = 0\}$. We can either assign the elements of M_3 to the sets M_1 or M_2. Given a pivot row $k \neq i$, let $M_3^+ = \{j \in J : \bar{a}_{ij} = 0 \wedge \bar{a}_{kj} > 0\}$ and $M_3^- = \{j \in J : \bar{a}_{ij} = 0 \wedge \bar{a}_{kj} < 0\}$. Balas and Perregaard [30] suggest the rule $M_1 = \{j \in J : \bar{a}_{ij} < 0\} \cup M_3^+$, $M_2 = J \setminus M_1$ for computing $r^l_{u_k}$ and $r^d_{u_k}$, and $M_1 = \{j \in J : \bar{a}_{ij} < 0\} \cup M_3^-$, $M_2 = J \setminus M_1$ for computing $r^l_{v_k}$ and $r^d_{v_k}$. If a negative reduced cost is identified, this rule is guaranteed to lead to a strictly better cut. On the other hand, each row $k \neq i$ of the tableau needs to be computed to obtain the values $\bar{a}_{kj}$ for all $j \in J$ satisfying $\bar{a}_{ij} = 0$. Nor can we conclude that the current basis of (LP) is optimal with respect to (CGLP$_i$) if all reduced cost are non-negative, since the reduced cost of each row k is computed with respect to a different partition (M_1, M_2). To decide whether the current solution is optimal, it is necessary to re-compute all the reduced cost with respect to a unique partition. Balas and Perregaard thus propose to perturb the row i of the simplex tableau in such a way that all non-basic variables have a non-zero coefficient. In our code we randomly assign the non-basic variables with a zero coefficient to the sets M_1 or M_2.

Typically, in implementations of the simplex algorithm a threshold value is used to decide if all reduced cost are non-negative. Let $r_k = \min\{r^l_{u_k}, r^l_{v_k}, r^d_{u_k}, r^d_{v_k}\}$. In our code we conclude that a basis B of (LP) corresponds to an optimal basis of (CGLP$_i$), if the reduced cost satisfy

$$r_k \geq -\epsilon, \quad \forall k \in B \setminus \{i\}, \tag{8.23}$$

where $\epsilon = 5 \cdot 10^{-4}$.

8.5.4. Finding the Incoming Variable

In Step 6 of Algorithm 8.5 the L&P procedure selects a variable which enters the basis. This variable is identified by finding the minima of the functions $f^+(\gamma)$ and $f^-(\gamma)$ (see Equation (4.102)). A procedure for finding the minima of these functions is described by Perregaard [148]. Suppose that we want to perform a pivot on the element $\bar{a}_{kp}$ in row $k \neq i$ that makes x_k non-basic at its lower bound l_k. An important detail is that it is not necessary to identify the minima of both functions $f^+(\gamma)$ and $f^-(\gamma)$. Using the bounds on x_k, we can rewrite the corresponding row of the simplex tableau in the form (8.15a). The pivot then has the following effect on (8.14)

$$x_i = \bar{a}_{i0} + \gamma_p \left(\bar{a}_{k0} - l_k\right) - \left(\sum_{j \in J} \left(\bar{a}_{ij} + \gamma_p \bar{a}_{kj}\right) s_j + \gamma_p s_k^l \right), \tag{8.24}$$

where $\gamma_p = -\frac{\bar{a}_{ip}}{\bar{a}_{kp}}$. Observe that γ_p is the coefficient of the variable s_k^l in the new combined simplex tableau row associated with x_i. In case we pivot x_k to its upper bound d_k, we obtain

$$x_i = \bar{a}_{i0} + \gamma_p \left(d_k - \bar{a}_{k0}\right) - \left(\sum_{j \in J} \left(\bar{a}_{ij} - \gamma_p \bar{a}_{kj}\right) s_j + \gamma_p s_k^d \right), \tag{8.25}$$

where $\gamma_p = -\frac{\bar{a}_{ip}}{(-\bar{a}_{kp})} = \frac{\bar{a}_{ip}}{\bar{a}_{kp}}$. Again, γ_p is the new coefficient of the non-basic variable s_k^d. Now, if we update the partition (M_1, M_2) of the non-basic indices, the sign of γ_p will determine whether s_k^l or s_k^d is linked to the basic variable u_k or v_k in (CGLP$_i$). Therefore if we select x_k due the reduced cost $r_{u_k}^l$ or $r_{u_k}^d$, it follows that $\gamma < 0$. On the other hand, choosing x_k based on reduced cost $r_{v_k}^l$ or $r_{v_k}^d$ implies that $\gamma > 0$. The practical consequence of this correlation is that one only needs to minimize either $f^+(\gamma)$ or $f^-(\gamma)$ once the leaving variable has been selected.

To enhance the numerical stability of the algorithm we do not consider pivots on elements $\bar{a}_{kj}$ of the pivot row k which are smaller than a threshold. Precisely, in our code, the set of admissible pivots is

$$J' = \{j \in J : |\bar{a}_{kj}| > \epsilon\} \tag{8.26}$$

where $\epsilon = 10^{-5}$.

8.5.5. Disjunctive Modularization

In order to approximate the generation of an optimal split cut, Balas and Bonami [22] iteratively strengthen the considered (LP) tableau row after each pivot by disjunctive modularization. After each pivot the reference row of the (LP) tableau (8.14) is replaced by the row

$$y_i = \bar{a}_{i0} - \sum_{j \in J} \bar{\varphi}_{ij} s_j \tag{8.27}$$

where y_i is a new unrestricted integer-constrained variable and

$$\bar{\varphi}_{ij} = \begin{cases} f_{ij} & \text{for } j \in J_I \text{ and } f_{ij} \leq \bar{a}_{i0} \\ f_{ij} - 1 & \text{for } j \in J_I \text{ and } f_{ij} > \bar{a}_{i0} \\ \bar{a}_{ij} & \text{otherwise} \end{cases} \tag{8.28}$$

with $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$ for $j \in J$. It is easy to see that the intersection cut derived from (8.27) is the strengthened intersection cut (or GMI cut) derived from (8.14).

8.5.6. Additional Considerations

For L&P cuts to be competitive with plain GMI cuts, the pivoting procedure outlined above must be implemented efficiently. There are a number of factors which have a large impact on the overall performance. Starting from the optimal (LP) tableau row a number of pivots are performed for each basic integer variable that is fractional. After pivoting based on the reduced cost of (CGLP$_i$), one has to restore the initial optimal (LP) basis. Of course, this can be done by running the dual simplex algorithm. Although only a few pivots of the dual simplex algorithm may be necessary to return to (LP) optimality, it has to be executed after each call to the L&P algorithm (see Algorithm 8.5). Therefore we initially save the optimal (LP) basis which can then be restored very quickly by filling the appropriate data structures. To be able to work with the optimal (LP) tableau we only need to re-factorize the basis. We found that this approach is considerably faster, especially when L&P cuts are generated for a large number of basic variables. As previously discussed, the calculation of the reduced cost of (CGLP$_i$) is a critical part of the L&P algorithm. Moreover, once the reduced costs are computed, they are scanned for the most negative element several times in each iteration. To reduce the effort needed for this operation we maintain a stack which contains the indices of the (LP) tableau rows corresponding to basic variables which have negative reduced cost.

Balas et al. [24, 25] demonstrate that L&P cuts can more efficiently be generated in a subspace obtained by fixing all structural non-basic variables. With respect to the cut generating linear program (CGLP$_i$) this means that the columns associated with these variables can be removed which considerably reduces the size of (CGLP$_i$). After (CGLP$_i$) has been optimized, the fixed variables are lifted into the L&P cut. In the framework of Balas and Perregaard the same effect can be achieved by ignoring the structural non-basic variables, in the sense that none of these variables is selected as an entering variable. Implementing the lifting step is then very easy. Once the algorithm has found the basis of (LP) giving the optimal L&P cut, we “lift” this cut by computing cut coefficients for all non-basic variables.

8.6. A new Pivoting Procedure for Strengthening Gomory Mixed-Integer Cuts

The Balas-Perregaard procedure [30] for generating lift-and-project cuts guides the search for an LP basis where the tableau row associated with a specific basic integer variable gives a stronger GMI cut than the corresponding row of the optimal LP tableau. In contrast, the reduce-and-split reduction algorithm [8] tries to improve the disjunction on which a GMI cut is based by forming integer linear combinations of LP tableau rows while keeping the basis fixed. A natural question is whether it is possible to integrate both approaches in order simultaneously to improve the basis and the disjunction.

We approach this question by generating GMI cuts from alternative bases of the LP relaxation. These bases need be neither optimal nor feasible. Consider the row of the simplex tableau (8.5) associated with a basic integer variable x_i. At each iteration of our algorithm we perform a pivot in row $k \neq i$, which produces a linear combination

$$x_i + \gamma x_k = \bar{a}_{i0} + \gamma \bar{a}_{k0} - \sum_{j \in J} (\bar{a}_{ij} + \gamma \bar{a}_{kj})\, s_j \tag{8.29}$$

where $\gamma = -\frac{\bar{a}_{ip}}{\bar{a}_{kp}}$ for some $p \in J$ such that the squared Euclidean norm of the coefficients of the continuous variables decreases. We want to obtain a basis where the representation of the simplex tableau row corresponding to x_i is "better" with respect to size of the coefficients of the continuous variables.

We thereby also modify the underlying split disjunction $D(\pi, \pi_0)$ as defined in Equation (4.48). Suppose that the pivot simulated in Equation (8.29) makes the integer-constrained variable x_k non-basic at its lower bound and the integer-constrained variable x_p basic. We have $B := (B \cup \{p\}) \setminus \{k\}$ and $J := (J \cup \{k\}) \setminus \{p\}$. Define $f_{ij} = \bar{a}_{ij} + \gamma \bar{a}_{kj} - \lfloor \bar{a}_{ij} + \gamma \bar{a}_{kj} \rfloor$ for $j \in J$ and $f_{i0} = \bar{a}_{i0} + \gamma \bar{a}_{k0} - \lfloor \bar{a}_{i0} + \gamma \bar{a}_{k0} \rfloor$. The new split disjunction $D(\pi, \pi_0)$ is given by

$$\pi_j = \begin{cases} \lfloor \bar{a}_{ij} + \gamma \bar{a}_{kj} \rfloor & \text{if } j \in J_I \text{ and } f_{ij} \le f_{i0}, \\ \lceil \bar{a}_{ij} + \gamma \bar{a}_{kj} \rceil & \text{if } j \in J_I \text{ and } f_{ij} > f_{i0}, \\ 1 & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases} \tag{8.30}$$

and $\pi_0 = \lfloor \bar{a}_{i0} + \gamma \bar{a}_{k0} \rfloor$. In particular this means that $\pi_p = 0$ and that π_k either has the value $\lfloor \gamma \rfloor$ or $\lceil \gamma \rceil$.

In the following we describe our pivoting algorithm in detail. As mentioned above, the objective of our algorithm is to reduce the squared Euclidean norm on the continuous variables in the selected reference row i of the simplex tableau. To state this objective more formally, we introduce some additional notation. Given $k \in B$, define the vector $\bar{a}_k^C$:

$$\bar{a}_{kj}^C = \begin{cases} \bar{a}_{kj} & \text{if } j \in J \setminus J_I, \\ 1 & \text{if } j = k \text{ and } j \in B \setminus B_I, \\ 0 & \text{otherwise.} \end{cases} \tag{8.31}$$

We want to select a basic variable x_k and a non-basic variable x_p such that pivoting on the element $\bar{a}_{kp}$ minimizes

$$\begin{aligned} f(\gamma) &= \left(\bar{a}_i^C + \gamma \bar{a}_k^C\right)\left(\bar{a}_i^C + \gamma \bar{a}_k^C\right), \\ &= \left\|\bar{a}_i^C\right\|^2 + \gamma^2 \left\|\bar{a}_k^C\right\|^2 + 2\gamma\left(\bar{a}_i^C \bar{a}_k^C\right), \end{aligned} \tag{8.32}$$

where $\gamma = -\frac{\bar{a}_{ip}}{\bar{a}_{kp}}$. It follows that a pivot reduces the squared Euclidean norm if the inequality

$$-2\gamma\left(\bar{a}_i^C \bar{a}_k^C\right) > \gamma^2 \left\|\bar{a}_k^C\right\|^2 \tag{8.33}$$

holds. Observe that the right-hand side of Inequality (8.33) is non-negative. Thus there is no improving pivot in row k with respect to the size of the coefficients of the continuous variables in row i if $\gamma(\bar{a}_i^C \bar{a}_k^C) \geq 0$.

The first main step of the procedure is to identify a variable x_k which leaves the basis, i.e. a pivot row $k \notin J \cup \{i\}$ which we want to combine our reference row i with. The heuristic we use to select this row measures the similarity between the reference row and possible pivot rows in terms of the coefficients of the continuous variables. Specifically, we compute the absolute value of the cosine of the angle between the vectors $\bar{a}_i^C$ and $\bar{a}_k^C$

$$\theta\left(\bar{a}_i^C, \bar{a}_k^C\right) = \frac{\left|\bar{a}_i^C \bar{a}_k^C\right|}{\|\bar{a}_i^C\| \, \|\bar{a}_k^C\|} \tag{8.34}$$

for each $k \notin J \cup \{i\}$. Note that $0 \leq \theta(\bar{a}_i^C, \bar{a}_k^C) \leq 1$. The larger $\theta(\bar{a}_i^C, \bar{a}_k^C)$ is, the larger is the likelihood that there exists a pivot in row k that reduces the size of the coefficients of the continuous variables in row i. On the other hand, if we have $\theta(\bar{a}_i^C, \bar{a}_k^C) = 0$ (i.e. $\bar{a}_i^C \bar{a}_k^C = 0$), every possible pivot will not be improving with respect to the size of the coefficients of the continuous variables (see Inequality (8.33)). Therefore the pivot row $k \neq i$ is chosen such that

$$k = \arg\max_{l \notin J \cup \{i\}} \left\{\theta(\bar{a}_i^C, \bar{a}_l^C)\right\}. \tag{8.35}$$

The second important step of the procedure is choosing a variable x_p which enters the basis. Again the critical question is how to select this variable. Remember that, given a pair (i, k) of rows of the simplex tableau, we want to minimize Equation (8.32). Therefore the effect that pivoting the non-basic variable x_p into the basis has on the size of the coefficients of the continuous variables can be measured by computing $f(\gamma_p)$ where $\gamma_p = -\frac{\bar{a}_{ip}}{\bar{a}_{kp}}$. Let $J' = \{j \in J : \bar{a}_{kj} \neq 0\}$ be the set of admissible pivots in tableau row k. From all of these pivots we select the one which brings about the largest possible reduction

$$p = \arg\min_{j \in J'} \left\{ f\left(\gamma_j\right) : \gamma_j = -\frac{\bar{a}_{ij}}{\bar{a}_{kj}} \right\}. \tag{8.36}$$

After the leaving and entering variable have been selected the pivot is performed. The method is iterated until a pivot limit is reached or no improving reduction is found. We call the resulting cuts *pivot-and-reduce (P&R) cuts.*

8.6.1. Implementation

An outline of our implementation of the previously discussed pivoting procedure is presented in Algorithm 8.6. There are a few aspects one has to consider to implement this procedure efficiently.

In Step 1 we perform an initialization of the data structures. We then enter the main loop of the procedure and check whether the termination criteria are satisfied (see Step 2). If the pivot limit is reached, we exit the main loop and generate a P&R cut in Step 7. Otherwise the algorithm computes the reference row of the simplex tableau in Step 3. We stop the search for an improving pivot if the coefficients of the continuous variables in the reference row i are relatively small, i.e. if the norm satisfies $\|\bar{a}_i^C\|^2 \leq \epsilon$ with $\epsilon = 10^{-5}$.

Given a basic integer variable x_i, we have to identify a pivot row $k \notin J \cup \{i\}$ which is likely to contain pivots which reduce the size of the coefficients on the continuous variables in row i (see Step 4). We want to choose the index k that maximizes the value of Equation (8.34). It turns out that the numerator of Equation (8.34) can be computed very efficiently. We get

$$\begin{aligned}
\lambda_k = \bar{a}_i^C \bar{a}_k^C &= \sum_{j \in J} \bar{a}_{kj} \bar{a}_{ij}, \\
&= \sum_{j \in J} \left(A_B^{-1} A_J\right)_{kj} \bar{a}_{ij}, \\
&= \sum_{j \in J} \left(A_{B(k)}^{-1} \left(A_J\right)_j\right) \bar{a}_{ij}, \\
&= A_{B(k)}^{-1} \sum_{j \in J} \left(A_J\right)_j \bar{a}_{ij},
\end{aligned} \tag{8.37}$$

Algorithm 8.6. Separating a pivot-and-reduce cut

```
Input: A basis $B$ of the LP relaxation of the IMR (7.2) and the
       corresponding basic solution $x^*$. A basic integer variable $x_i$
       which has a fractional value $x_i^*$. The parameter
       $max\_pivots$ indicating the maximum number of pivots.
Output: A P&R cut $\alpha x \geq \beta$.
(Step 1) Initialize
    Set $num\_pivots := 0$.
(Step 2) Check termination criteria
    if $num\_pivots \geq max\_pivots$ then goto Step 7.
    Let $S := B \setminus \{i\}$.
(Step 3) Compute the tableau row
    Compute the tableau row (8.5) associated with $x_i$, i.e.
    $x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j$.
    if $\|\bar{a}_i^C\|^2 \leq \epsilon$ then goto Step 7.
(Step 4) Find the leaving variable
    if $S = \emptyset$ then goto Step 7.
    Compute the inner product $|\bar{a}_i^C \bar{a}_k^C|$ for all $k \in S$.
    Let $k := \arg\max_{l \in S}\{|\bar{a}_i^C \bar{a}_l^C|\}$.
    if $|\bar{a}_i^C \bar{a}_k^C| \leq \epsilon$ then goto Step 7.
(Step 5) Find the incoming variable
    Let $f(\gamma) = \|\bar{a}_i^C\|^2 + \gamma^2\|\bar{a}_k^C\|^2 + 2\gamma(\bar{a}_i^C \bar{a}_k^C)$.
    Let $J' = \{j \in J : |\bar{a}_{kj}| \geq \epsilon\}$ be the set of admissible pivots.
    Compute $f(\gamma_j)$ with $\gamma_j = -\frac{\bar{a}_{ij}}{\bar{a}_{kj}}$ for all $j \in J'$.
    Let $p := \arg\min_{j \in J'}\{f(\gamma_j = -\frac{\bar{a}_{ij}}{\bar{a}_{kj}})\}$.
    Set $S := S \setminus \{k\}$.
    if $f(\gamma_p) \geq f(0)$ then goto Step 4.
(Step 6) Perform the pivot
    Pivot the variable $x_k$ out of the basis and pivot the variable $x_p$ into
    the basis $B$. Set $B := (B \cup \{p\}) \setminus \{k\}$ and $J := (J \cup \{k\}) \setminus \{p\}$.
    Set $num\_pivots := num\_pivots + 1$ and goto Step 2.
(Step 7) Compute the P&R cut
    Compute the row of the simplex tableau (8.3) associated with $x_i$
    and generate the P&R cut $\alpha x \geq \beta$ using Algorithm 8.2.
```

where $B(k)$ is the position of x_k in the basis B, $A^{-1}_{B(k)}$ is the row of the basis inverse associated with x_k and $(A_J)_j$ is the j^{th} non-basic column of A. To obtain the vector λ, i.e. the inner product for all candidate rows $k \notin J \cup \{i\}$, we have to solve the system

$$A_B \lambda = \sum_{j \in J} (A_J)_j \, \bar{a}_{ij} \tag{8.38}$$

for λ. Again this system can be solved by a single ftran operation.

The denominator of Equation (8.34) can not be calculated so efficiently. The Euclidean norm of the tableau row associated with x_i is of course a constant in all calculations. On the other hand, the Euclidean norms of the remaining rows of the simplex tableau may change after each pivot. Therefore there are two ways to compute the norms for all rows $k \notin J \cup \{i\}$. Firstly, one can construct each tableau row in order to calculate its Euclidean norm. This is a computationally very expensive operation. Secondly, one can initially compute the squared Euclidean norm for all rows of the simplex tableau and then update them after each pivot using Equation (8.32). This is also quite costly from a computational point of view. Accordingly, we only compute the numerator for all candidates and select k such that

$$k = \arg \max_{l \notin J \cup \{i\}} \left\{ \left| \bar{a}_i^C \bar{a}_l^C \right| \right\}. \tag{8.39}$$

Suppose that we have computed the index k according to the above formula. The algorithm is stopped if the inner product with the largest absolute value is sufficiently small. More precisely, the algorithm is stopped if

$$\left| \bar{a}_i^C \bar{a}_k^C \right| \leq \epsilon, \tag{8.40}$$

where $\epsilon = 5 \cdot 10^{-4}$ in our code.

The second main step of the procedure is selecting the variable which will enter the basis (see Step 5 of Algorithm 8.6). It can be implemented in a straightforward way. As elaborated above, we select the entering variable x_p from row k such that the resulting pivot will produce the largest possible reduction in the coefficients on the continuous variables in row i. In our current implementation the algorithm does not, however, check whether the P&R cut derived from the updated row i obtained after the pivot (see Step 6) is violated by the optimal LP solution x^*.

As in our implementation of the L&P cuts, we reject pivots on elements $\bar{a}_{kj}$ which are smaller than the pivot tolerance 10^{-5}.

The overall computational effort necessary to select the entering and leaving variable is comparable to the work carried out by the Balas-Perregaard method for generating L&P cuts. But the L&P algorithm has to maintain a lot of data, e.g. the value of the objective function of the cut generating linear program and the partition of the non-basic variables. An advantage of the P&R heuristic is that it only has to retain the values of the inner products between the vectors $\bar{a}_i^C$ and $\bar{a}_k^C$ for each possible pivot row $k \notin J \cup \{i\}$ (see Equation (8.39)) while the L&P algorithm has to hold four reduced cost values. Moreover, the inner products are the direct result of the ftran operation and can therefore be scanned and handled very efficiently, e.g. by sorting according to descending absolute values. After a pivot has been performed no additional work is necessary for updates of existing data.

8.7. Strong Chvátal-Gomory Cuts

In this section we outline our implementation of the strong Chvátal-Gomory cuts. These cuts are a strengthened variant of the well known CG cuts. We have already dealt with the theory of strong CG cuts in Section 4.3.2.

In Chapter 4 we mentioned that a Gomory fractional cut is a CG cut derived from a row of the simplex tableau. We also discussed the relation between mixed-integer rounding cuts and Gomory mixed-integer cuts, namely that any GMI cut can be derived as a MIR cut from a tableau row. Our implementation of the strong CG cuts is based on suggestions made by Achterberg [3]. Inspired by the connection between CG cuts and Gomory fractional cuts, Achterberg proposes to generate strong Gomory fractional cuts, i.e. strong CG cuts from rows of the simplex tableau.

In this regard the separation of a strong CG cut works very similarly to the separation of a GMI cut (see Section 8.1). The details of the separation are outlined in Algorithms 8.7 and 8.8. Proposition 4.3 assumes that all variables are non-negative. Thus we need to shift or complement the variables. As strong CG cuts are not able to handle continuous variables we also have to ensure that all continuous variables have a non-negative coefficient in the transformed row of

Algorithm 8.7. Separating a strong Chvátal-Gomory cut (part 1)

```
Input: A basis $B$ of the LP relaxation of the IMR (7.2) where $x_i$
       is a basic fractional integer variable.
Output: An inequality $\sum_{j \in N} \bar{a}_j s_j \leq \bar{a}_0$ or transformation failed.

(Step 1) Compute the base row
Let $w = e_i A_B^{-1}$ be the $i^{th}$ row of the basis inverse.
for $k = 1, \ldots, m$ with $w_k \neq 0$ do
    if $n + k \in N \setminus N_I$ then
        if $(w_k > 0$ or $d_{n+k} = \infty)$ and $(w_k < 0$ or $l_{n+k} = -\infty)$
        then
            $w_k := 0$.
        end
    end
end
Compute $wA$ and let the result be $\sum_{j \in N} \bar{a}'_j x_j = 0$.

(Step 2) Shift and complement
Set $\bar{a}_0 := 0$.
for $j = 1, \ldots, n + m$ with $\bar{a}'_j \neq 0$ do
    if $j \in N_I$ then
        if $l_j > -\infty$ then
            (Shift.) Set $\bar{a}_j := \bar{a}'_j$, $x_j := l_j + s_j$, and $v := l_j$.
        else if $d_j < \infty$ then
            (Complement.) Set $\bar{a}_j := -\bar{a}'_j$, $x_j := d_j - s_j$, and
            $v := d_j$.
        else
            exit.
        end
    else
        if $\bar{a}_j > 0$ and $l_j > -\infty$ then
            (Shift.) Set $\bar{a}_j := \bar{a}'_j$, $x_j := l_j + s_j$, and $v := l_j$.
        else if $\bar{a}_j < 0$ and $d_j < \infty$ then
            (Complement.) Set $\bar{a}_j := -\bar{a}'_j$, $x_j := d_j - s_j$, and
            $v := d_j$.
        else
            exit.
        end
    end
    (Update the right-hand side.) Set $\bar{a}_0 := \bar{a}_0 - \bar{a}'_j \cdot v$.
end
(Relax.) Set $\bar{a}_j := 0$ for all $j \in N \setminus N_I$.
The transformed row has the form $\sum_{j \in N} \bar{a}_j s_j \leq \bar{a}_0$.
```

Algorithm 8.8. Separating a strong Chvátal-Gomory cut (part 2)

```
Input: An inequality of the form $\sum_{j \in N} \bar{a}_j s_j \leq \bar{a}_0$ generated by
       Algorithm 8.7.
Output: A strong CG cut $\alpha x \leq \beta$.

(Step 1) Compute the strong CG cut
         Compute the strong CG cut $\sum_{j \in N} \alpha'_j s_j \leq \beta'$ (see Proposition 4.3).
(Step 2) Unshift and uncomplement
         Undo the shifts and complementations on the variables using the
         same bounds as in Step 2 of Algorithm 8.7.
(Step 3) Remove logicals
         Use the definition of the logical variables $x_L = -\tilde{A} x_S$ to write the
         cut in the space of the structural variables $x_S$.
```

the tableau. Therefore we choose the substitution that produces a non-negative coefficient (see Step 2 of Algorithm 8.7). If this is possible, the continuous variables can be removed from the tableau row, thereby relaxing it to a $\leq$-inequality. On the other hand, the substitution may fail due to missing bounds on some continuous variables. In this case we can not generate a strong CG cut. In particular, every continuous logical variable is required to have a non-negative coefficient. Note that the coefficient of a logical variable in a row of the simplex tableau is directly given by the associated row of the basis inverse. Thus it is easy to check whether a continuous logical variable can be transformed such that is has a non-negative coefficient. If this is not possible, we zero out the corresponding position in the row of the basis inverse (see Step 1 of Algorithm 8.7). This means that we are not necessarily computing rows of the simplex tableau associated with a basis. Rather we use the row of the basis inverse to guide the search for a violated strong CG cut.

The resulting row is of the form $\sum_{j \in N} \bar{a}'_j x_j = 0$. After the transformation discussed above, we obtain

$$\sum_{j \in N} \bar{a}_j s_j \leq \bar{a}_0. \tag{8.41}$$

A strong CG cut can then be generated using Proposition 4.3 (see Step 1 of Algorithm 8.8). As in Section 4.3.2, let $f_0 = \bar{a}_0 - \lfloor \bar{a}_0 \rfloor$ and $f_j = \bar{a}_j - \lfloor \bar{a}_j \rfloor$ for

$j \in N$. First we have to choose an integer $k \geq 1$ that satisfies $\frac{1}{k+1} \leq f_0 < \frac{1}{k}$. As this is equivalent to $\frac{1}{f_0} - 1 \leq k < \frac{k+1}{kf_0} - 1$, we set

$$k = \left\lceil \frac{1}{f_0} \right\rceil - 1 \tag{8.42}$$

in our code. According to the value of k a partition $(N_0, \ldots, N_k)$ of N is constructed. Recall that the set N_0 is defined as $N_0 = \{j \in N : f_j \leq f_0\}$ and the set N_p is defined as $N_p = \{j \in N : f_0 + \frac{(p-1)(1-f_0)}{k} < f_j \leq f_0 + \frac{p(1-f_0)}{k}\}$ for $p = 1, \ldots, k$. We do not work with the partition $(N_0, \ldots, N_k)$ explicitly. Observe that we can also write $N_p = \{j \in N : p - 1 < \frac{k(f_j - f_0)}{1-f_0} \leq p\}$ for $p = 1, \ldots, k$. It follows that we can restate the strong CG cut as

$$\sum_{j \in N} \left(\lfloor \bar{a}_j \rfloor + \frac{p_j}{k+1} \right) s_j \leq \lfloor \bar{a}_0 \rfloor, \tag{8.43}$$

where the non-negative integer p_j for $j \in N$ is given by

$$p_j = \begin{cases} 0 & \text{if } f_j \leq f_0, \\ \left\lceil \frac{k(f_j - f_0)}{1-f_0} \right\rceil & \text{if } f_j > f_0. \end{cases} \tag{8.44}$$

To write the cut in the space of the original structural variables we undo the shifts and complementations and substitute for the logical variables (see Steps 2 and 3 of Algorithm 8.8).

8.8. $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts

In this section we discuss an algorithm for generating $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts. Our implementation of this algorithm is based on the work of Koster et al. [126]. We briefly introduced $\{0, \frac{1}{2}\}$-cuts in Section 4.3.1. We assume that an IP of the form

$$\min \{cx : Ax \leq b, x \geq 0, x \in \mathbb{Z}^n\} \tag{8.45}$$

is given, where A is an integral matrix of dimension $m \times n$ and b is an integral vector of dimension m. Let x^* be a non-integral solution to the LP relaxation of (8.45). Let $s = b - Ax$ be a vector of slack variables. A general IP (7.2) can be brought into the form (8.45) by transforming all inequalities into $\leq$-inequalities and shifting or complementing the variables to obtain $x \geq 0$.

Now, let $\bar{A} = A \bmod 2$ and $\bar{b} \bmod 2$ where the modulo operation is applied component-wise. There exists a violated $\{0, \frac{1}{2}\}$-cut if and only if there exists a binary vector $v \in \{0,1\}^m$ such that

$$v\bar{b} \bmod 2 = 1 \text{ and } \left(v\bar{A} \bmod 2\right) x^* + vs^* < 1. \tag{8.46}$$

A vector v satisfying the constraints presented in Equation (8.46) indicates which inequalities of $Ax \leq b$ to combine with weights $\frac{1}{2}$ such that the CG cut from this linear combination is violated (see Section 4.3.1).

8.8.1. Preprocessing

Our code performs three basic operations on the system $(\bar{A}, \bar{b})$. Not all columns and rows of $(\bar{A}, \bar{b})$ affect the two conditions presented in Equation (8.46). A number of transformations can be performed which reduce the size of $(\bar{A}, \bar{b})$.

Proposition 8.1 (**Reduction** [126]). *The reductions below do not influence the set of undominated $\{0, \frac{1}{2}\}$-cuts for the original system (A, b):*

1. *All columns in $\bar{A}$ corresponding to variables $x^*_j = 0$ can be removed.*
2. *Zero rows in $(\bar{A}, \bar{b})$ can be removed.*
3. *Zero columns in $\bar{A}$ can be removed.*
4. *Identical columns in $\bar{A}$ can be replaced by a single representative with associated variable value as sum of the merged variables.*
5. *Any unit vector (or singleton) column $\bar{a}_j = e_i$ in $\bar{A}$ with $j \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, m\}$ can be removed provided that x^*_j is added to the slack s^*_i of row i.*
6. *Any row $i \in \{1, \ldots, m\}$ with slack $s^*_i \geq 1$ can be removed.*
7. *Rows identical in $(\bar{A}, \bar{b})$ can be eliminated except for one with smallest slack value.*

For simplicity, let the reduced system again consist of m rows and n columns. Up to now we assumed that each row of $(\bar{A}, \bar{b})$ is associated with a single inequality from the original system. In order to obtain additional reductions we now consider row operations on $(\bar{A}, \bar{b})$. To this end, let R_i contain the indices of the original inequalities of which row i is currently composed. We initialize R_i by setting $R_i = \{i\}$ for $i = 1, \ldots, m$. Let us assume we add row k to row i. We get

$$\begin{aligned} \bar{a}_{ij} &:= \bar{a}_{ij} + \bar{a}_{kj} \bmod 2, \quad \forall j \in \{1, \ldots, n\}, \\ \bar{b}_i &:= \bar{b}_i + \bar{b}_k \bmod 2, \\ s_i^* &:= s_i^* + s_k^*, \\ R_i &:= (R_i \cup R_k) \setminus (R_i \cap R_k), \end{aligned} \tag{8.47}$$

where $\bar{a}_{ij}$ is the coefficient of the matrix $\bar{A}$ in row i and column j. The following proposition plays a major role in obtaining additional reductions.

Proposition 8.2 (**Elimination** [126])**.** *Let i be the index of a row and j the index of a column of $\bar{A}$ such that $\bar{a}_{ij} = 1$ and $s_i^* = 0$. Then adding row i to all other rows $k \neq i$ with $\bar{a}_{kj} = 1$ does not influence the set of undominated $\{0, \frac{1}{2}\}$-cuts for the original system (A, b).*

Proposition 8.2 creates new singleton columns. These columns can then be removed using rule 5 of Proposition 8.1, thereby reducing the size of $(\bar{A}, \bar{b})$. In addition, other preprocessing rules may be applicable after the elimination of these singleton columns. In particular, certain zero rows of $\bar{A}$ directly correspond to violated $\{0, \frac{1}{2}\}$-cuts.

Proposition 8.3 (**Separation** [126])**.** *Let i be the index of a zero row in $\bar{A}$ with $\bar{b}_i = 1$. If $s_i < 1$, then the weight vector u defined by $u_k = \frac{1}{2}$ for all $k \in R_i$ and 0 otherwise defines a violated $\{0, \frac{1}{2}\}$-cut on the original system (A, b).*

8.8.2. Implementation

In this section we present a heuristic procedure for obtaining $\{0, \frac{1}{2}\}$-cuts which is implemented in our code (see Algorithm 8.9). An algorithm for the exact separation of $\{0, \frac{1}{2}\}$-cuts is discussed by Koster et al. [126]. This exact algorithm uses the above reduction and elimination rules to reduce the size of $(\bar{A}, \bar{b})$ and then solves a reduced-size separation problem. In our code we primarily use Proposition 8.3 to obtain violated $\{0, \frac{1}{2}\}$-cuts.

Algorithm 8.9. Separating $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts

```
         Input: An IP (8.45) and a fractional solution $x^*$ of the associated
                LP relaxation. An integer $k$ indicating the maximum
                number of rows combined in the heuristic search.
         Output: A list $\mathcal{L}$ of $\{0, \frac{1}{2}\}$-cuts.
(Step 1) Initialize
         Compute the slack values $s^* = b - Ax^*$.
         Construct the system $(\bar{A}, \bar{b})$.
(Step 2) Reduce
         Perform the reductions presented in Proposition 8.1.
(Step 3) Separate
         Find violated $\{0, \frac{1}{2}\}$-cuts using Proposition 8.3.
         Add these cuts to the list $\mathcal{L}$.
(Step 4) Eliminate
         if $s_i^* > 0$ holds for all rows $i$ of $(\bar{A}, \bar{b})$ then goto Step 5.
         Create new singleton columns by the elimination algorithm
         outlined in Proposition 8.2.
         Goto Step 2.
(Step 5) Search
         Try to find combinations of $l = 1, \dots, k$ rows of $(\bar{A}, \bar{b})$ that give a
         violated $\{0, \frac{1}{2}\}$-cut and add them to the list $\mathcal{L}$.
```

In Step 1 of Algorithm 8.9 we compute the matrix $\bar{A}$ and the vector $\bar{b}$ and store them in a compact form (see Figure 7.3). We then try to reduce the size of the system $(\bar{A}, \bar{b})$ using the preprocessing rules presented in Proposition 8.1 (see Step 2). Since checking for duplicate columns and rows is very time-consuming, we only apply rules 1 to 3 and 5 to 6 in our code. In Step 3 we apply Proposition 8.3 and search for zero rows i of $\bar{A}$ satisfying $s_i^* < 1$ and $\bar{b}_i = 1$. If such a row is found, we first extract the set original inequalities which is associated with this row. We then combine these original inequalities with weights $\frac{1}{2}$ and generate a CG cut. In fact, to obtain stronger cutting planes, we generate a complemented MIR cut. Alternatively, we could also generate a strong CG cut. If a violated $\{0, \frac{1}{2}\}$-cut is found by Proposition 8.3, we also remove the corresponding row from the system $(\bar{A}, \bar{b})$ in the hope of achieving further reductions. In Step 4 of the algorithm the elimination procedure outlined in Proposition 8.2 is performed. We create column singletons which are then removed by reduction rule 5 of Proposition 8.1. This procedure can only be carried out as long as there are rows with a slack of zero. A natural question is which columns should be preferred. When removing a singleton column j which has its only non-zero entry in row i, the value x_j^* is added to the slack of row i. In our case we have $s_i^* := x_j^*$ since we select a row with slack zero. In addition, row i can be removed as well if the updated slack satisfies $s_i^* \geq 1$. It is thus desirable to create singletons in columns with a large value of x_j^*. After the elimination step a new iteration of the procedure is started. If no row has zero slack, we exit the main loop of the procedure and perform a final heuristic search for violated $\{0, \frac{1}{2}\}$-cuts (see Step 5). First we try to identify single rows of $(\bar{A}, \bar{b})$ giving a violated $\{0, \frac{1}{2}\}$-cut. Then we check combinations of two rows of $(\bar{A}, \bar{b})$. This process is iterated up to k rows. However, as this search is rather expensive, we set $k = 1$ in our code.

8.9. Computational Results

In the preceding sections we discussed our implementation of separation algorithms for several classes of split cuts. We particularly focussed on Gomory mixed-integer cuts as the most prominent split cuts and also considered Chvátal-Gomory cuts for pure integer programs. We furthermore discussed several algorithms designed to improve the performance of the GMI cuts. This section reports on our computational experience. We investigate whether the performance of the GMI cuts can in practice be improved by the algorithms discussed earlier (see Sections 8.2 to 8.5). Moreover, we study the effect that the separation of CG cuts (see Sections 8.7 and 8.8) has on the performance of Mops. We perform two types of experiments. In the first we enable particular cut separators and measure the performance improvement or deterioration as compared with a reference setting. In the second type of experiment we again consider a reference setting and disable particular cut separators in order to evaluate each separator's contribution to the overall performance. Concerning cut separators working on the simplex tableau, we limit cut generation to the fifty most fractional structural basic integer variables. We use a cut selection algorithm which we shall discuss in more detail in Chapter 10. Details of our experimental setup, test set and the evaluation methods used can be found in Appendix A.

Most of the improvement algorithms for GMI cuts we discussed are heuristic in nature. The only exact algorithm we considered is the lift-and-project algorithm which is guaranteed to find a better (more violated) GMI cut from an alternative basis of the LP relaxation if one exists. We therefore decided to use the L&P cut separator as a replacement for the GMI cut separator in our benchmarks. By contrast, the combined GMI, reduce-and-split and pivot-and-reduce cut separators are executed on top of the GMI cut separator. As mentioned earlier (see Section 8.1) we use a separation routine for mixed-integer rounding cuts to generate the cutting planes discussed in this chapter. Our k-cut separator is integrated into this separation routine in the sense that a tableau row which is fed to our MIR separation routine is automatically scaled with different non-negative integer values k (see Section 8.2). We generate a MIR cut for each value of k and keep the one with the largest distance cut off relative to the optimal solution to the LP relaxation.

Concerning the R&S, P&R, L&P, cGMI and k-cut separators, Table 8.1 presents the results of our first experiment, i.e. the performance gain achieved by applying each individual separator in conjunction with the GMI cut separator. Exceptions are, as noted above, the L&P cut separator which replaces the GMI cut separator and the k-cut separator which is integrated into our separation framework. The detailed results can be found in Tables B.1 to B.5 in Appendix B. Enabling the k-cut separator (“GMI + k-cuts”) leads to a slight improvement in the performance on the CORAL test set. On the other hand, the performance deteriorates significantly on the ACC test set. For instance the running times and number of nodes increase by more than 100%. Similarly, applying the cGMI separator (“GMI + cGMI”) does not yield a reduction in the running times on any test set in terms of the relative shifted geometric means. Again a slight increase in the amount of integrality gap closed and a reduction in the branching nodes computed can be observed on the test set CORAL. The R&S cut separator (“GMI + R&S”) is very effective on the MIPLIB test set where it closes 4% more integrality gap and reduces the number of branching nodes and the running times by about 10%. Concerning the MILP test set, the GMI cut separator

	test set	GMI + k-cuts	GMI + cGMI	GMI + R&S	GMI + P&R	L&P
time	ACC	**+101**	0	0	**+12**	**-8**
	CORAL	-1	+1	-2	0	0
	MIPLIB	+2	+1	**-10**	**-19**	**-6**
	MILP	+2	+2	**+7**	**-7**	**+11**
	total	+3	+1	-2	**-5**	0
nodes	ACC	**+124**	0	0	**+18**	**-11**
	CORAL	-3	-1	**-17**	**-6**	**-7**
	MIPLIB	**+7**	-2	**-11**	**-30**	**-24**
	MILP	+3	**+6**	**+7**	**-10**	**+7**
	total	+4	0	**-12**	**-13**	**-11**
gap	ACC	0	0	0	0	0
	CORAL	-1	-1	-3	**-10**	**-12**
	MIPLIB	+2	+3	-4	**-9**	**-18**
	MILP	+2	**+9**	-3	**-8**	**+5**
	total	0	+1	-3	**-9**	**-11**

Table 8.1. Performance impact of enabling particular single-row tableau cut separators. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which only the GMI cut separator is activated. Positive values indicate a deterioration while negative values signify an improvement.

outperforms the R&S cut separator with respect to running times and number of branching nodes. The performance ratios are heavily influenced by the instance `swath1` where the R&S cut separator closes 35% more integrality gap. At the same time, however, the running time and the number of branching nodes increase by a factor of 5 and 10 respectively which indicates that the amount of integrality gap closed might not be a good performance indicator. The P&R cut separator ("GMI + P&R") can significantly improve upon the performance of the GMI cut separator on the test sets MIPLIB and MILP. For the MIPLIB instances the running time and the number of branching nodes decrease by about 20% and 30% respectively. Like the R&S cut separator the P&R cut separator is not effective on the ACC instances. Both separators apply computationally quite expensive algorithms to obtain cuts with small coefficients on the continuous variables. Since the ACC test set only contains pure 0-1 instances it is not surprising that the R&S and P&R cut separators are not effective. The L&P cut separator ("L&P") outperforms the other separators on the test sets CORAL and MIPLIB with respect to the amount of integrality gap closed. In particular, for the MIPLIB instances the L&P cut separator closes 18% more integrality gap than the GMI cut separator. Along with this reduction in the integrality gap the number of branching nodes decreases by about 25%. Table 8.1 also indicates, however, that the L&P cut separator deteriorates the performance on the MILP test set. The running time and the number of branching nodes increase by 11% and 7% respectively. An extreme example is again the instance `swath1` where the running time increases by a factor of about 28. Concerning the total test set the R&S, P&R and L&P cut separators reduce the number of branching nodes by about 10% and significantly increase the amount of integrality gap closed. Compared with these improvements the decrease in the running times, on the other hand, is quite small.

We conclude from the above discussion that the R&S, P&R and L&P cut separators are the winners of our first experiment, while the k-cut separator and the cGMI cut separator seem to be largely ineffective on our test set. For many instances R&S, P&R and L&P cut separators obviously succeed in obtaining cuts which are of higher quality than those produced by the GMI cut separator. On the other hand, the results also reflect that the R&S and P&R cut separators are heuristic algorithms which are not guaranteed to generate improved cuts. By contrast the L&P cut separator is guaranteed to find the most-violated intersection cut (with respect to a fixed disjunction). A different question is whether these improved cuts cause a gain in performance. The L&P cut separator computes at least 10% fewer branching nodes than the GMI cut separator on 77 instances and closes at least 10% more integrality gap than the GMI cut separator on 48 instances in the entire test set (see Table B.5). The results also show, however, that adding improved cuts does not invariably result in an improved dual bound and a reduction in the running times or the number of branching nodes. In general, our computational experiments indicate that it is difficult to compete with the GMI cut separator with respect to running times. The main reason is the computational cost of the R&S, P&R and L&P cut separation algorithms. On easy or moderately difficult instances the speed-up created by improved cuts is overcompensated by the running times of the separators (i.e. the time needed to compute these improved cuts). It is known that cutting planes play a more important role in solving hard instances than in solving easy

		test set	GMI + *k*-cuts	GMI + cGMI	GMI + R&S	GMI + P&R	L&P
easy	time	ACC	**+530**	+1	+1	**+38**	-13
		CORAL	**+6**	+4	**+29**	**+32**	**+31**
		MIPLIB	+2	-3	**+9**	**-15**	**+60**
		MILP	+1	**-12**	**+79**	**+113**	**+279**
		total	**+17**	+1	**+24**	**+23**	**+46**
	nodes	ACC	**+618**	0	0	**+52**	**-23**
		CORAL	0	+2	**-7**	**+6**	**+8**
		MIPLIB	**+11**	-3	+1	**-27**	**-8**
		MILP	**+6**	+1	**+82**	**+79**	**+222**
		total	**+12**	-1	0	**-6**	**+5**
	gap	ACC	0	0	0	0	0
		CORAL	+2	0	-4	**-5**	-2
		MIPLIB	0	+2	**-5**	**-12**	-2
		MILP	**+17**	**+29**	**-7**	**-13**	**+54**
		total	+1	+2	-4	**-8**	+1
hard	time	ACC	0	0	0	0	**-7**
		CORAL	-4	0	**-11**	**-9**	**-10**
		MIPLIB	+3	+4	**-25**	**-33**	**-37**
		MILP	+3	+4	0	**-19**	**-9**
		total	-2	+1	**-11**	**-15**	**-14**
	nodes	ACC	0	0	0	0	-3
		CORAL	-4	-3	**-23**	**-13**	**-16**
		MIPLIB	+1	+1	**-31**	**-39**	**-50**
		MILP	+3	**+7**	-3	**-21**	**-13**
		total	-2	0	**-20**	**-19**	**-21**
	gap	ACC	0	0	0	0	0
		CORAL	-2	-2	-2	**-14**	**-20**
		MIPLIB	**+6**	**+5**	0	-3	**-68**
		MILP	0	**+6**	-2	**-8**	-4
		total	0	+1	-2	**-10**	**-22**

Table 8.2. Performance impact of enabling particular single-row tableau cut separators on easy and hard instances. We consider an instance to be easy if it can be solved to optimality using only the GMI cut separator in 60 seconds or less. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which only the GMI cut separator is activated. Positive values indicate a deterioration while negative values signify an improvement.

	test set	only GMI	no k-cuts	no cGMI	no R&S	no P&R	no L&P
time	ACC	-3	+1	0	0	**-8**	+2
	CORAL	-2	**-13**	**-6**	**-10**	**-6**	**+9**
	MIPLIB	**+19**	**+6**	**+6**	**-11**	**+14**	**+25**
	MILP	-1	**+9**	+4	-1	**+16**	0
	total	+2	**-5**	-2	**-8**	+1	**+10**
nodes	ACC	+3	0	0	0	**-8**	**+8**
	CORAL	**+15**	**-17**	**-6**	**-7**	**-15**	**+14**
	MIPLIB	**+103**	**+8**	**+10**	**-5**	**+49**	**+47**
	MILP	**+13**	**+6**	+3	-1	**+22**	**+11**
	total	**+33**	**-8**	-1	**-5**	+4	**+21**
gap	ACC	0	0	0	0	0	0
	CORAL	**+18**	-1	0	+2	+1	**+11**
	MIPLIB	**+25**	+2	0	+4	+4	**+10**
	MILP	**+18**	**+5**	+3	**+5**	**+9**	**+21**
	total	**+19**	+1	+1	+3	+3	**+12**

Table 8.3. Performance impact of disabling particular single-row tableau cut separators. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row tableau cut separators are activated. Positive values indicate a deterioration while negative values signify an improvement.

instances, e.g instances which can be solved efficiently by pure branch-and-bound. In our context this means that the R&S, P&R and L&P cut separators should be applied on instances which are hard to solve using only the GMI cut separator. Accordingly, we partition our test instances into two sets: those instances that can be solved using only the GMI cut separator in 60 seconds or less, and those that can not. Table 8.2 is similar to Table 8.1 and presents the performance of the cut separators on the easy and hard instances respectively. Concerning the hard instances, the R&S, P&R and L&P cut separators, as expected, outperform the GMI cut separator. Concerning the total test set, the running times decrease by between 11% and 15% and the number of branching nodes is reduced by about 20%. On the other hand, a deterioration can be observed on the easy instances. The running times on the test set MILP, for instance, increase by 279% when activating the L&P cut separator. The detailed results can be found in Appendix B, more precisely in Tables B.6 to B.10 for the easy instances and in Tables B.11 to B.15 for the hard instances.

In our second experiment we consider a reference setting in which we apply the GMI cut separator together with the cGMI, R&S, P&R, L&P and k-cut separators. We then disable one of last-mentioned separators at a time and record the performance improvement or deterioration. We are interested in the contribution of each separator to the overall performance and the interaction of the different separators. Table 8.3 shows a summary of the results of this experiment. The detailed results are presented in Tables B.16 to B.20 in Appendix B. The column headed "only GMI" shows the relative performance of the solver if only the GMI cut separator is activated, i.e. a setting in which all "improved" separators are deactivated. When using only the GMI cut separator the number of branching nodes computed increases considerably (e.g. by 103% on the MIPLIB instances) and the amount of integrality gap closed clearly reduces. With respect to running times the results are not so clear. While a slight reduction in the running times can be observed on the ACC and CORAL instances if GMI cuts only are separated, the performance clearly deteriorates on the MIPLIB instances. Deactivating the cGMI ("no cGMI") or k-cut separator ("no k-cuts") improves the performance of the solver on the CORAL instances. Interestingly, CORAL was the only test set we detected the k-cut separator to be effective on in our first experiment. Although the R&S cut separator turned out to be effective when applied individually in our first experiment, Table 8.3 reveals that disabling the R&S cut separator ("no R&S") does not result in a performance deterioration. On the contrary, deactivating the R&S cut separator reduces the running times and the number of branching nodes on any of our test sets (in terms of the relative shifted geometric means). A possible explanation of this behavior is that the cuts generated by the R&S cut separator coincide to a large extent with the cuts generated by the remaining separators (e.g. those generated by the P&R cut separator). In addition, the R&S cut separation algorithm is computationally very demanding. The deactivation of the P&R cut separator ("no P&R") interferes with the performance of the solver on the MIPLIB and MILP instances. The running time increases by about 15% and the number branching nodes by about 50% and 20% respectively. In our first experiment the P&R cut separator yielded the largest improvement over the GMI cut separator precisely on these two test sets. As an additional similarity to our first experiment, disabling the P&R cut separator improves the performances on the ACC test set (see Table 8.1). The

L&P cut separator greatly affects the overall performance of the solver. All values in the column headed “no L&P” are non-negative, indicating that deactivating the L&P cut separator mainly leads to performance deteriorations with respect to the shifted geometric means of running times, branching nodes and amounts of integrality gap closed. On the MIPLIB instances the running time increases by 25% and the number of branching nodes by 47% if the L&P cut separator is disabled. In addition the amount of integrality gap closed decreases by 21%. It is further interesting that the running times increase by 19% if only the GMI cut separator is activated (“only GMI”) and by 25% if we apply our reference setting with the L&P cut separator deactivated (“no L&P”). A similar observation can be made regarding the number of branching nodes. The deterioration obtained when disabling the P&R cut separator (“no P&R”) is larger than that obtained when applying only the GMI cut separator. Concerning the total test set the sum of the relative changes of the number of nodes is, moreover, smaller than the deterioration obtained when disabling all “improved” separators. These observations point to the interaction of the different cut separators. With respect to the total test set the P&R and L&P cut separators are the only separators whose deactivation leads to an increase in the shifted geometric means of the running times and number of branching nodes.

In order to obtain satisfactory performance of the L&P cut separator it requires a sophisticated implementation and fine-tuning. All computational results for the L&P cut separator we presented so far were obtained with a standard version (cf. [30]) which uses the standard normalization (see Equation (4.83)) and a pivot limit of 10. We now consider improving the performance of this standard L&P cut separator by applying disjunctive modularization (see Section 8.5.5) and the Euclidean normalization (see Section 4.4.7). Following our previous approach to evaluate the performance of cut separators, we use the standard L&P cut separator as the reference setting and enable the two last-mentioned techniques. The results of this experiment are summarized in Table 8.4. The detailed results can be found in Tables B.26 to B.30 in Appendix B. Activating disjunctive modularization (“L&P + DM”) improves the performance of the L&P cut separator on the ACC and MILP instances. The running time decreases by 1% and 5% respectively and the number of branching nodes by 5% for both test sets. For the MIPLIB instances the L&P cut separator with disjunctive modularization closes 2% more

	test set	L&P + DM	L&P + EN	L&P + DM + EN
time	ACC	-1	**-20**	**+9**
	CORAL	+4	+2	+2
	MIPLIB	+3	**-10**	**-8**
	MILP	**-5**	**-11**	+2
	total	+2	-4	0
nodes	ACC	**-5**	**-22**	**+9**
	CORAL	+2	+4	+3
	MIPLIB	**+5**	**-13**	**-9**
	MILP	**-5**	**-13**	**-6**
	total	+2	**-5**	-1
gap	ACC	0	0	0
	CORAL	+1	-1	0
	MIPLIB	-2	-2	-2
	MILP	+3	**-9**	**-12**
	total	0	-2	-2

Table 8.4. Performance impact of enabling disjunctive modularization (DM) and Euclidean normalization (EN). The values represent percentage changes in the shifted geometric mean compared with a reference setting in which the standard L&P cut separator is activated. Positive values indicate a deterioration while negative values signify an improvement.

integrality gap than the standard L&P cut separator. The variant of the L&P cut separator applying the Euclidean normalization (“L&P + EN”) is particularly effective on the ACC instances. The running time decreases by 20% and the number of branching nodes by 22%. Fischetti et al. [87] found the Euclidean normalization to be well suited for instances whose constraints have only non-negative coefficients (e.g. set covering or set partitioning instances). Since the scheduling problem from which the ACC instances arise contains, among others, set partitioning constraints, it is not surprising that the Euclidean normalization is successful. The Euclidean normalization also works well on the MIPLIB and MILP instances where it reduces the running time by about 10% and the number of branching nodes by 13%. Concerning the amount of integrality gap closed, applying the Euclidean normalization yields an improvement of about 9% on the MILP test set. Another observation that can be made from Table 8.4 is that, in comparison with the standard L&P cut separator and with regard to running times, applying disjunctive modularization and Euclidean normalization together

	test set	SR tableau + strong CG	SR tableau + $\{0, \frac{1}{2}\}$	SR tableau + strong CG + $\{0, \frac{1}{2}\}$
time	ACC	**+9**	**-10**	**+50**
	CORAL	**-7**	**-19**	**-19**
	MIPLIB	**+6**	**+11**	**+8**
	MILP	+4	**+10**	**+20**
	total	-2	**-9**	**-6**
nodes	ACC	**-7**	**+11**	**+42**
	CORAL	**-7**	**-25**	**-25**
	MIPLIB	-1	+1	**-5**
	MILP	0	-2	+1
	total	**-5**	**-15**	**-15**
gap	ACC	0	0	0
	CORAL	0	**-7**	**-9**
	MIPLIB	-3	-2	**-5**
	MILP	**+5**	**-7**	**-11**
	total	0	**-5**	**-8**

Table 8.5. Performance impact of enabling particular CG cut separators. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row tableau cut separators are activated. Positive values indicate a deterioration while negative values signify an improvement.

("L&P + DM + EN") is not effective on any test set except for MIPLIB. The L&P cut separator which applies only the Euclidean normalization, however, performs just as well or better than the combined variant with respect to the relative shifted geometric means. The only exception is the amount of integrality gap closed on the MILP instances. We conclude from the above discussion that combining disjunctive modularization and Euclidean normalization interferes with the positive effects that each of these techniques has when applied individually. This behavior can especially be seen on the ACC instances where the combination of both techniques performs worse than any other configuration.

In the preceding part of this section we evaluated our GMI cut separator and other separators which try to generate improved GMI cuts. In the remainder of this section we study the computational effectiveness of CG cuts and consider the strong CG cut separator (see Section 8.7) and the $\{0, \frac{1}{2}\}$-cut separator (see Section 8.8). We consider a reference setting in which all previously discussed tableau cut separators are activated, then enable the CG cut separators and measure the relative improvement or deterioration in performance. We are interested in whether the CG cut separators can improve the performance of the solver or whether it is enough to generate (improved) GMI cuts. Table 8.5 presents a summary of this experiment. Tables B.21 to B.25 in Appendix B show the detailed results. For the CORAL instances enabling the strong CG cut separator ("SR tableau + strong CG") reduces the shifted geometric mean of the running times and the number of branching nodes by 7%. The strong CG cut separator, on the other hand, deteriorates the performance on the other test sets with regard to running times. Concerning our entire assembly of test instances, activating the strong CG cut separator reduces the running times by 2% and the number of branching nodes by 5%. Activating the $\{0, \frac{1}{2}\}$-cut separator ("SR tableau + $\{0, \frac{1}{2}\}$") results in a reduction in the running times on the ACC and CORAL instances. The running times, more precisely, reduce by 10% on the ACC instances and by 19% on the CORAL instances. In addition the number of branching nodes computed decreases by 25% for the CORAL instances. On the test sets MIPLIB and MILP the running times, however, increase by about 10%. Concerning the amount of integrality gap closed at the root node, the variant in which the $\{0, \frac{1}{2}\}$-cut separator is activated consistently outperforms the reference setting. On the total test set enabling the $\{0, \frac{1}{2}\}$-cut separator yields a reduction in the running times and the number of branching nodes by 9% and 15% respectively. Furthermore, the amount of integrality gap closed increases by 5%. Applying the $\{0, \frac{1}{2}\}$-cut separator together with the strong CG cut separator ("SR tableau + strong CG + $\{0, \frac{1}{2}\}$") yields the largest increases in the amounts of integrality gap closed at the root node. Adding the strong CG cut separator interferes with the good performance of the $\{0, \frac{1}{2}\}$-cut separator on the ACC instances. While there is a 10% decrease in the running times if only the $\{0, \frac{1}{2}\}$-cut separator is activated, using both separators in conjunction increases the running times by 50% and the number of branching nodes by 42%. The combination of both CG separators, as mentioned above, closes the largest amounts of integrality gap. Yet the running times on the MILP instances increase by 20% and the number of branching nodes by 1%. Concerning the CORAL test set, the relative shifted geometric means obtained using both CG cut separators are equal to those obtained with the $\{0, \frac{1}{2}\}$-cut separator. All entries in the rows headed "total" are non-positive, indicating that, concerning the entire collection of test instances, the CG cut separators can improve upon the performance of the GMI cut separators.

Chapter 9.

Multi-Row Cutting Plane Separators

This chapter complements Chapter 5 which was concerned with the derivation of cutting planes from multiple rows of a simplex tableau. In this chapter we present a separation algorithm for multi-row cuts and discuss the key issues for an efficient implementation. We define the maximal lattice-free convex sets we use to derive cuts and show how we construct the multi-row relaxations. We also discuss how stronger cuts can be obtained by using the integrality conditions on the non-basic integer variables. Finally, we report on our computational experience with multi-row cuts.

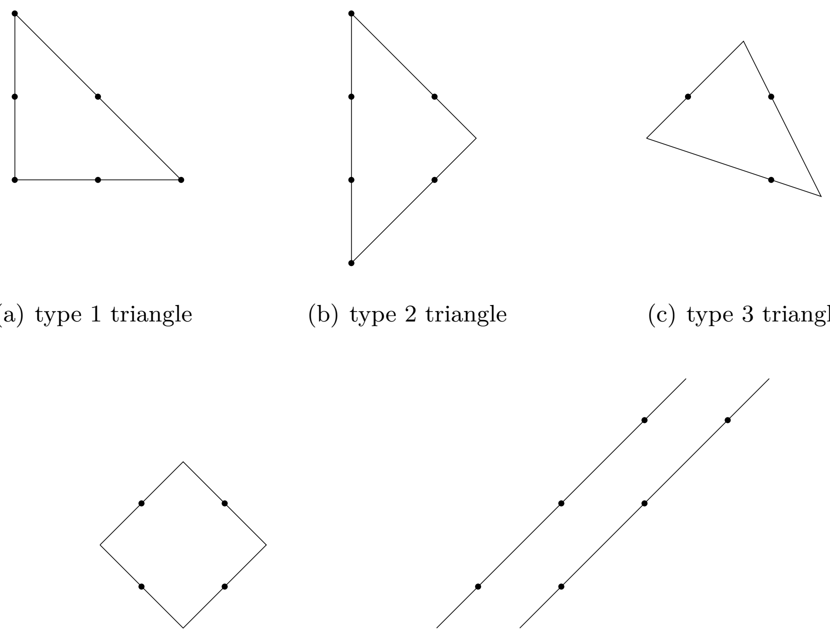
This chapter is organized as follows. Section 9.1 deals with maximal lattice-free convex sets. The construction of a multi-row relaxation is described in Section 9.2. In Section 9.3 we discuss how we generate intersection cuts and highlight some implementation details. Computational results are presented in Section 9.4.

9.1. Maximal Lattice-Free Convex Sets

Given the results presented in Section 5.4, particularly Theorem 5.1, the main task regarding the generation of multi-row cuts is to select a maximal lattice-free convex set.

9.1.1. Maximal Lattice-Free Convex Sets in the Plane

In the plane there exist three general classes of maximal lattice-free convex sets [133]. A maximal lattice-free *quadrilateral* contains four integral points each of which lies in the relative interior of one edge. An unbounded maximal lattice-free convex set which consists of two edges is called a *split*. Finally, following


(a) type 1 triangle (b) type 2 triangle (c) type 3 triangle

(d) quadrilateral (e) split

Figure 9.1. Non-empty maximal lattice-free convex sets in $\mathbb{R}^2$

Dey and Wolsey [78], we distinguish between three classes of maximal lattice-free *triangles* (see Section 5.4).

9.1.2. Selected Maximal Lattice-Free Convex Sets

In the following we define the families of maximal lattice-free convex sets we use in our computational experiments. A characterization of all maximal lattice-free convex sets in arbitrary dimension is unknown. Therefore we use higher-dimensional generalizations of the maximal lattice-free convex sets in $\mathbb{R}^2$ we discussed in the previous section. Some of the following definitions are taken from Espinoza [83, 84].

Definition 9.1 ($T1_n$ [84])**.** *We define*

$$T1_n := \{x \in \mathbb{R}^n : x \geq 0, ex \leq n\} \tag{9.1}$$

where e is the vector of all ones.

The set $T1_n$ is defined by $n+1$ inequalities. Each of these inequalities has one integral point in its relative interior. Moreover, we have $\operatorname{int}(T1_n) \cap \mathbb{Z}^n = \emptyset$ and the volume of $T1_n$ is given by $\frac{n^n}{n!}$. Thus $T1_n$ is a bounded maximal lattice-free convex set in $\mathbb{R}^n$. The set $T1_n$ has $n+1$ vertices which are given by $v_n^1, \dots, v_n^{n+1} \in \mathbb{Q}^n$ with $(v_n^k)_i = n$ if $i = k$ and $(v_n^k)_i = 0$ otherwise. Note that the set $T1_2$ is a maximal lattice-free triangle of type 1 as shown in Figure 9.1(a).

Definition 9.2 ($T2_n$ [84])**.** *We define*

$$T2_n := \left\{ x \in \mathbb{R}^n : \begin{array}{ll} R_j : & \sum_{i=1}^{j-1} x_i - x_j \leq j-1, \forall j = 1, \dots, n \\ R_{n+1} : & ex \leq n \end{array} \right\} \tag{9.2}$$

The set $T2_n$ is defined by $n+1$ inequalities. An integral point lies in the relative interior of each of these inequalities. The volume of $T2_n$ is $(2^{\frac{n+1}{2}} - 2^{\frac{1-n}{2}})^n / n!$ which is substantially larger than the volume of $T1_n$. In addition we have $\operatorname{int}(T2_n) \cap \mathbb{Z}^n = \emptyset$. It follows that the set $T2_n$ is a bounded maximal lattice-free convex set in $\mathbb{R}^n$. The $n+1$ vertices of $T2_n$ are given by $v_n^1, \dots, v_n^{n+1} \in \mathbb{Q}^n$ with $(v_n^k)_i = 1 - 2^{i-1}$ if $i < k$, $(v_n^k)_i = 2^k - 2^{i-1} - 2^{k-n} + 1$ if $i = k$ and $(v_n^k)_i = 1 - 2^{i-1-n}$ if $i > k$. The set $T2_2$ is a maximal lattice-free triangle of type 2 as shown in Figure 9.1(b).

Definition 9.3 ($T3_n$)**.** *We define*

$$T3_n := \left\{ x \in \mathbb{R}^n : \begin{array}{lll} R_1 : & \gamma x \leq -1 - \lambda_n & \\ R_j : & \sum_{i=1}^{j-1} x_i - x_j \leq j-1, & \forall j = 2, \dots, n \\ R_{n+1} : & \delta x \leq n + 1 + \lambda_n & \end{array} \right\} \tag{9.3}$$

where $\gamma \in \mathbb{Z}^n$ and $\delta \in \mathbb{Z}^n$ are the vectors $\gamma = (-3, -1, 0, \dots, 0)$, $\delta = (1, 2, 1, \dots, 1)$ and $\lambda_n = \min\{n-2, 0\}$.

Proposition 9.4. *The set $T3_n$ is lattice-free, i.e. there is no point $x \in \mathbb{Z}^n$ in the interior of $T3_n$. Furthermore, each facet of $T3_n$ has an integer point in its relative interior.*

Proof. We prove the claim by showing that every integral point $x \in \mathbb{Z}^n$ in $T3_n$ lies in the relative interior of a facet R_j with $j \in \{1, \dots, n+1\}$. Suppose that $n = 1$, then $T3_1 = \text{conv}\{0, 1\}$ and the statement holds. We assume that $n \geq 2$ in what follows. As $0 \leq x_1 \leq 1$ for $x \in T3_n \cap \mathbb{Z}^n$, we have $T3_n \cap \mathbb{Z}^n = P_1 \cup P_2$ where

$$\begin{aligned} P_1 &= T3_n \cap \{x \in \mathbb{Z}^n : x_1 = 0\}, \\ P_2 &= T3_n \cap \{x \in \mathbb{Z}^n : x_1 = 1\}. \end{aligned} \tag{9.4}$$

First we consider the set P_1 where $x_1 = 0$. In this case, R_1 forces all feasible integral solutions to have the characteristic that $x_2 \geq 1$. All solutions having $x_2 = 1$ satisfy R_1 at equality. Note that for $n \geq 3$ there are also solutions with $x_2 = 2$. Furthermore, the variable x_2 is upper bounded, i.e. we have that $x_2 \leq 2$ for $x \in T3_n \cap \mathbb{Z}^n$. Now, assume that $n \geq 3$ and consider the set

$$P_{1k} = (T3_n \cap \mathbb{Z}^n) \cap \left\{x \in \mathbb{Z}^n : x = (0, 2, e^{k-3}, 0, t), t \in \mathbb{Z}^{n-k}\right\}, \tag{9.5}$$

where e^k is the vector of all ones with dimension k and $3 \leq k \leq n$. All solutions $x \in P_{1k}$ satisfy constraint R_k at equality.

Next we investigate the set P_2 where $x_1 = 1$. All of the solutions having $x_1 = 1$ and $x_2 = 0$ satisfy R_2 at equality. Consider the set

$$P_{2k} = (T3_n \cap \mathbb{Z}^n) \cap \left\{x \in \mathbb{Z}^n : x = (e^{k-1}, 0, t), t \in \mathbb{Z}^{n-k}\right\} \tag{9.6}$$

where $2 \leq k \leq n$. It is easy to see that all solutions in P_{2k} satisfy R_k at equality. Finally, the solution e^n satisfies R_{n+1} at equality which completes the proof. $\square$

The set $T3_n$ is defined by $n+1$ inequalities, each of them containing an integer point in its relative interior and $\text{int}(T3_n) \cap \mathbb{Z}^n = \emptyset$. The volume of $T3_n$ is

$$\frac{2^{1-\frac{n(n-1)}{2}} (3 \cdot 2^n + 2)^n}{(2^{2n} + 2^{n+3} + 12)\, n!} \tag{9.7}$$

for $n \geq 2$. Thus $T3_n$ is a bounded maximal lattice-free convex set in $\mathbb{R}^n$. The volume of $T3_n$ is smaller than the volume of $T2_n$. On the other hand, for increasing values of n the volume of $T3_n$ is clearly larger than the volume of $T1_n$. Note that the set $T3_2$ is a maximal lattice-free triangle of type 3 (see Figure 9.1(c)).

The set $T3_n$ has $n+1$ vertices. For $n = 1$ we obtain the two vertices $v^1 = 0$ and $v^2 = 1$. For $n \geq 2$ the vertices of $T3_n$ are given by the vectors $v_n^1, \dots, v_n^{n+1} \in \mathbb{Q}^n$

$$(v_n^1)_i := \begin{cases} \frac{2^{n+1}+2}{2^n+2} & \text{if } i = 1, \\ \frac{2^n-2^{i-1}+2}{2^n+2} & \text{otherwise,} \end{cases} \tag{9.8}$$

$$(v_n^2)_i := \begin{cases} -\frac{2^n-2}{2^n+6} & \text{if } i = 1, \\ \frac{2^{n+2}}{2^n+6} & \text{if } i = 2, \\ \frac{2^n-2^{i+1}+6}{2^n+6} & \text{otherwise,} \end{cases} \tag{9.9}$$

$$(v_n^k)_i := \begin{cases} 1 - 2^{i-1} + 2^{\max\{-1,i-3\}} & \text{if } i < k, \\ 1 + 2^{i-n-1} + 3 \cdot 2^{i-3} & \text{if } i = k, \\ 1 + 2^{i-n-2} & \text{if } i > k, \end{cases} \tag{9.10}$$

with $k = 3, \dots, n+1$. It is easy to see that these vertices are non-integral.

Definition 9.5 (G_n [83, 84])**.** *We define*

$$G_n := \frac{1}{2}e + \left\{ x \in \mathbb{R}^n : \delta x \leq \frac{n}{2}, \forall \delta \in \{-1,1\}^n \right\}. \tag{9.11}$$

The set G_n is defined by 2^n inequalities. Each of these inequalities has one integer point in its relative interior and $\text{int}(G_n) \cap \mathbb{Z}^n = \emptyset$. Moreover, the set G_n has a volume of $\frac{n^n}{n!}$. Thus G_n is a bounded maximal lattice-free convex set in $\mathbb{R}^n$. In particular, the set G_2 is a maximal lattice-free quadrilateral (see Figure 9.1(d)). Note also that the set G_n has $2n$ vertices which are given by $\frac{1}{2}e + \{V\}$ where the set V contains all permutations of the vector $(\pm\frac{n}{2}, 0, \dots, 0)$.

Definition 9.6 (SP_n)**.** *We define*

$$SP_n := \frac{1}{2}e + \left\{ x \in \mathbb{R}^n : \begin{array}{c} \delta x \leq \dfrac{n}{2} \\ \delta x \geq \dfrac{n}{2} - 1 \end{array} \right\} \tag{9.12}$$

with $\delta \in \{-1,1\}^n$.

Proposition 9.7. *The set SP_n is lattice-free, i.e. there is no point $x \in \mathbb{Z}^n$ in the interior of SP_n. Furthermore, each facet of SP_n has an integer point in its relative interior.*

Proof. Let $S = \{1, \dots, n\}$. Observe that SP_n can alternatively be written as

$$SP_n := \left\{ x \in \mathbb{R}^n : \begin{array}{c} \delta x \leq \rho \\ \delta x \geq \rho - 1 \end{array} \right\}, \tag{9.13}$$

where $\delta \in \{-1, 1\}^n$ and $\rho = |S^+|$ with $S^+ = \{i \in S : \delta_i = 1\}$. As the vector δ and the right-hand side ρ are integral, SP_n is a split. By construction, a split cannot contain integral points in its interior. For instance, the vectors t^+ with $t_i^+ = 1, \forall i \in S^+$, $t_i = 0$ otherwise, and $t^- = t^+ - e_k$ with $k \in S \setminus S^+$ satisfy one of the inequalities defining SP_n at equality respectively. $\square$

Finally, we offer some comments on the characteristics of the selected maximal lattice-free convex sets. The sets $T1_n$, $T2_n$ and G_n completely contain the 0-1 hypercube in $\mathbb{R}^n$ while the sets $T3_n$ and SP_n do not. We can rotate each axis around $\frac{1}{2}e$ by 180 degrees and again obtain maximal lattice-free convex sets. There are 2^n of these *orientations*. Moreover, additional variations are generated by permuting the sequence of the variables defining a set. There exist $n!$ of these *permutations*. Therefore each set discussed above in fact gives rise to a *family of lattice-free convex sets*. Following Espinoza [84], we shall denote, for a fixed dimension n, these families by $T1_n$, $T2_n$, $T3_n$, G_n or SP_n respectively. A family altogether yields $2^n n!$ maximal lattice-free convex sets for a fixed dimension n. However, not all families are affected by these variations, i.e. not all of the generated sets differ from each other. In particular, the family G_n is completely symmetric, and thus all variations correspond to the same lattice-free convex set. The families $T1_n$ and SP_n are not affected by permuting the variables and each of them generates 2^n different sets. Concerning $T2_n$, we obtain 2^{n-1} different sets for a fixed permutation, and $2^{n-1}n!$ different sets in total. The family $T3_n$ represents an extreme in terms of variations and generates $2^n n!$ different sets.

9.2. Selecting a Multi-Row Relaxation

Another important step in generating multi-row cuts is the construction of an appropriate multi-row relaxation. We first compute the rows of the simplex tableau associated with the basic integer variables. We then rewrite these rows in the form

$$x_i = x_i^* + \sum_{j \in J} (-\bar{a}_{ij})\, s_j, \quad i \in B_I. \tag{9.14}$$

Recall that the variables s are the slack or surplus variables from the simple lower or upper bound constraints respectively (see Section 8.1). We store these transformed tableau rows in a matrix. As we want to experiment with different multi-row relaxations we thereby avoid expensive re-computations which slow down the overall separation algorithm. Let $Q \subseteq B_I$ be a subset of the basic integer variables. For simplicity, let us assume that $Q = \{1, \dots, q\}$. With the help of the tableau rows (9.14) we are able efficiently to construct any multi-row relaxation of the form

$$\begin{aligned} x_Q &= f + \sum_{j \in J} r^j s_j, \\ s_j &\geq 0, \quad j \in J, \\ s_j &\in \mathbb{Z}, \quad j \in J_I, \\ x_Q &\in \mathbb{Z}^q. \end{aligned} \tag{9.15}$$

The system (9.15) can be viewed as a group relaxation (see Section 5.3). We therefore let f be a vector with components $f_i = x_i^* - \lfloor x_i^* \rfloor$ for $i \in Q$ and replace any column vector r^j associated with a non-basic integer variable by the vector of its fractional parts, i.e. $r_i^j := r_i^j - \lfloor r_i^j \rfloor$ for all $j \in J_I$ and $i \in Q$.

Concerning the separation of multi-row cuts, any set Q of basic integer variables with at least one fractional (i.e. $\exists i \in Q$ such that $f_i > 0$) provides a relaxation from which a violated cut can be derived. Nevertheless, in practice the question is which and how many of the tableau rows should be selected. To obtain a q-row relaxation of the form (9.15), we select the tableau rows associated with the q most fractional structural basic integer variables in our implementation. A similar strategy is also used to derive Gomory mixed-integer cuts (see Section 8.1).

9.3. Generating Intersection Cuts

Intersection cuts are derived from a basis of the LP relaxation and a violated disjunction. Given a multi-row relaxation (9.14) consisting of q rows, we derive violated disjunctions by means of maximal lattice-free convex sets. More precisely, once a maximal lattice-free convex set $S = \{x \in \mathbb{R}^q : \pi^i x_Q \leq \pi_0^i, i = 1, \ldots, l\}$ with f in its interior has been selected, we can derive the violated disjunction

$$\bigvee_{i=1}^{l} \left(\pi^i x_Q \geq \pi_0^i\right). \tag{9.16}$$

The intersection cut is then given by

$$\sum_{j \in J} \max_{i=1,\ldots,l} \left\{ \frac{\pi^i r^j}{\pi_0^i - \pi^i f} \right\} s_j \geq 1, \tag{9.17}$$

and can be written in the space of the original variables by substituting for the variables s.

To derive an intersection cut we compute the points at which the extreme rays r^j and the inequalities defining a maximal lattice-free convex set intersect. Since the family G_q is defined by 2^q inequalities, the work to derive an intersection cut from G^q is exponential in q. The families $T1_q$, $T2_q$ and $T3_q$ are in contrast defined by $n + 1$ inequalities which makes their separation less expensive. As suggested by Espinoza [84], we use gray-code enumeration [122] to generate all tuples $\{-1, 1\}^q$ (or orientations respectively) and a plain changes algorithm [122] to iterate over all permutations of $\{1, 2, \ldots, q\}$.

We generate intersection cuts from all families of maximal lattice-free convex sets discussed in Section 9.1.2 (see Definition 9.1 to 9.6). Given a family of maximal lattice free convex sets, we iterate over all variants obtained by reorientating the axes or permuting the variables. Since the families $T1_q$, $T2_q$ and G_q contain the 0-1 hypercube in $\mathbb{R}^q$, all variants obtained by applying these operations yield valid intersection cuts. The situation is different for the families $T3_q$ and SP_q. Concerning a specific orientation and permutation, we have to check if the solution f lies in the interior of the resulting maximal lattice-free convex set. We generate an intersection cut $\alpha x \geq 1$ for each possible variant and keep the cut with the largest value of $\frac{1}{\|\alpha\|}$.

The reason for using gray-code enumeration to generate the tuples $\{-1,1\}^q$ is that two consecutive tuples only differ by a single element. For example, consider again the set G_q (see Definition 9.5) which is defined by 2^q inequalities whose coefficients are given by the tuples $\{-1,1\}^q$. Now, suppose two vectors $\pi, \hat{\pi} \in \{-1,1\}^q$ are given which only differ on the k^{th} component with $1 \leq k \leq q$. Thus we can obtain $\hat{\pi}$ by either setting $\hat{\pi} := \pi + 2e_k$ or $\hat{\pi} := \pi - 2e_k$ depending on whether the k^{th} element changed from 1 to -1 or vice versa. It follows that we can perform a simple update to switch from one to the next inequality defining G_q. Suppose that the k^{th} component is changed from 1 to -1. We obtain

$$\begin{aligned} \hat{\pi} r^j &= (\pi - 2e_k)\, r^j = \pi r^j + 2\bar{a}_{kj}, \quad \forall j \in J, \\ \hat{\pi} f &= (\pi - 2e_k)\, f = \pi f - 2f_k. \end{aligned} \tag{9.18}$$

The opposite case works analogously. Using this trick we can speed up the computation of the intersection cut (9.17) generated from G_q. In particular, we can work with the rows of the simplex tableau (9.14) instead of the columns r^j. Since gray-code enumeration is also used to iterate over the orientations of the sets $T1_q$, $T2_q$, $T3_q$ and SP_q, we can also use this trick to speed up this process.

9.3.1. Strengthening

In this section we discuss improving the performance of the intersection cut by strengthening the disjunction (9.16). We consider the so-called trivial fill-in function (or trivial strengthening) discussed in Section 5.4. Given a q-row relaxation of the form (9.15), the strengthened intersection cut generated by the trivial fill-in function reads

$$\sum_{j \in J_I} \min_{u^j \in \mathbb{Z}^q} \left\{ \max_{i=1,\ldots,l} \left\{ \frac{\pi^i \left(r^j - u^j\right)}{\pi_0^i - \pi^i f} \right\} \right\} s_j + \sum_{j \in J \setminus J_I} \max_{i=1,\ldots,l} \left\{ \frac{\pi^i r^j}{\pi_0^i - \pi^i f} \right\} s_j \geq 1. \tag{9.19}$$

To strengthen the intersection cut a minimization problem is solved for each non-basic integer variable. We are not, however, forced to solve these problems to optimality. Since the intersection cut is a $\geq$-inequality and we are minimizing the coefficients of the integer variables, any set of integral vectors $\{u^j\}_{j \in J_I}$ generates a valid intersection cut.

Concerning the maximal lattice-free triangles $T1_2$ and $T2_2$, Dey and Wolsey [78, 79] showed that the trivial strengthening is optimal, i.e. the resulting intersection cuts (9.19) are minimal. In general, the trivial strengthening does not always produce the strongest possible intersection cuts. Even for the maximal lattice-free triangle $T3_2$ the trivial strengthening only generates minimal inequalities under additional conditions. On the other hand, while the trivial strengthening function may not reach the optimum, it may still succeed in improving the coefficients of some of the integer variables. In our view, this justifies applying the (strengthened) intersection cut (9.19) while overlooking the fact that the trivial fill-in function does not produce minimal inequalities in general.

Our implementation is as follows. For each non-basic integer variable we simply iterate over vectors $u^j \in \mathbb{Z}^q$ and select the one producing the best (smallest) coefficient. In the previous section we noted that the cut generation process can be improved by saving parts of previous computations and by working with the rows of the simplex tableau. However, when computing the trivial fill-in function we instead have to work with the columns r^j of the simplex tableau. Concerning the orientations of the lattice-free bodies, we can not derive the intersection cut produced by the current orientation from the intersection cut generated by the previous one using simple updates. Therefore separation algorithms using the trivial fill-in function to strengthen the intersection cuts are likely to be slower than those not using any strengthening.

Finally, we offer some comments on intersection cuts generated from lattice-free split bodies. Any intersection cut generated from a lattice-free split body is a single-row cut in the sense that it can also be obtained by applying integrality arguments and rounding to a linear combination of the rows in the multi-row relaxation. On the other hand, split cuts have the advantage that they can be strengthened using the closed-form formula presented in Proposition 4.5. Although intersection cuts derived from SP_n (see Definition 9.6) are single-row cuts, we generate these cuts in order to compare them with the multi-row cuts generated using the remaining families of maximal lattice-free convex sets.

9.4. Computational Results

In the preceding sections we described a practical implementation of multi-row cut separators. We concentrated particularly on the two key decisions in generating intersection cuts from multiple rows of the simplex tableau: the selection of a multi-row relaxation and the selection of a violated disjunction (or, in other words, the selection of a maximal lattice-free convex set containing the fractional LP solution f). In this section we report on our computational experience with multi-row cut separators. We show that the separation of multi-row cutting planes can positively affect the overall performance of an MIP solver. In line with our evaluation method in Chapter 8 we perform two types of experiments. Our experimental setup, test set and evaluation methods are covered in more detail in Appendix A.

Our separation scheme for multi-row cuts is as follows. Suppose a family of maximal lattice-free convex sets, say $T1_n$ (see Definition 9.1), is given. Let $\bar{m}$ be the maximum number of intersection cuts we allow to be generated from $T1_n$ in each round of IP preprocessing. Given the dimension n and the upper bound $\bar{m}$, define $s = ((k-1) \cdot n) + 1$ for $k = 1, \ldots, \bar{m}$. For each starting value s we construct a multi-row relaxation consisting of LP tableau rows associated with the s^{th} to the $(s+n-1)^{th}$ most fractional structural basic integer variables and generate an intersection cut. As with the single-row cut separators discussed in Chapter 8, we generate at most fifty cuts ($\bar{m} = 50$) for each selected maximal lattice-free convex set in our implementation. For instance, given the set $T2_2$ (i.e. a maximal lattice-free triangle of type 2), we first generate a triangle cut using the tableau rows associated with the first and second most fractional structural basic integer variables, then for the third and fourth most fractional structural basic integer variables and so on. We do not add all generated cuts to the LP relaxation. The cut selection algorithm we use is described in Chapter 10.

In our first experiment we individually apply the multi-row cut separators on top of the single-row cut separators discussed in Chapter 8 and record the percentage change in the shifted geometric means of the running times, number of branching nodes and amounts of integrality gap closed. This experiment is meant to answer the question of whether multi-row cut separators can contribute to improving the overall performance of an MIP solver. The results of this experiment are summarized in Tables 9.1 and 9.2. The detailed results can be found in Tables B.31 to B.35 in Appendix B. In particular Table 9.1 presents results obtained with the two-row and lifted (or strengthened) two-row cut separators. The last mentioned cut separators use the trivial fill-in function or trivial strengthening (see Equation (9.19)) to obtain intersection cuts with small coefficients on the integer-constrained variables. The only exceptions are intersection cuts generated from split disjunctions (i.e. split cuts) which we strengthen using Proposition 4.5. Columns "$T1_2$" to "SP_2" in Table 9.1 show the results obtained with plain (unstrengthened) two-row intersection cuts generated from different families of maximal lattice-free convex sets (see Section 9.1). The remaining columns "$T1L_2$" to "SPL_2" present the results obtained with the strengthened (or lifted) versions of these cuts. In the same way Table 9.2 shows the results obtained with intersection cuts from higher-dimensional maximal lattice-free convex sets and their strengthened counterparts. We emphasize again that in general the trivial strengthening is not guaranteed to yield minimal intersection cuts. In the special case where a maximal lattice-free triangle of type 1 or type 2 is used to derive an intersection cut, this strengthening, however, yields minimal coefficients on the integer variables (see [78, 79]).

We first examine the effect that activating individual two-row cut separators has on the performance of MOPS. An immediate observation that can be made from Table 9.1 is that the two-row cut separators are only in some cases competitive with the single-row cut separators in terms of the relative shifted geometric means of the times needed to solve the instances in the test sets to optimality. Concerning the ACC and MIPLIB instances, the shifted geometric means of the running times consistently increase. For these two test sets, the number of branching nodes is also consistently larger than in the reference setting which only makes use of the single-row cut separators. The situation is, on the other hand, different for the CORAL and MILP instances. Activating the unstrengthened two-row cut separators leads to a decrease in the running times on the MILP instances. Decreases in the running times of 7% and 8% can be observed when applying the type 1 ("$T1_2$") or type 3 triangle cut separator ("$T3_2$") respectively. Simultaneously the number of branching nodes reduces by 6% and 8% respectively. For the CORAL instances the largest decreases in the shifted geometric means of the running times (8%) are yielded by the type 3 triangle cut separator and the

	test set	$T1_2$	$T2_2$	$T3_2$	G_2	SP_2	$T1L_2$	$T2L_2$	$T3L_2$	GL_2	SPL_2
time	ACC	**+9**	**+16**	**+18**	**+13**	**+22**	**+15**	**+16**	**+16**	**+14**	**+12**
	CORAL	+1	-1	**-8**	**+7**	**-8**	+4	0	+2	-4	**-6**
	MIPLIB	**+11**	**+21**	**+13**	**+6**	**+8**	**+17**	**+24**	**+8**	**+6**	+3
	MILP	**-7**	-1	**-8**	**-5**	-4	**+10**	-2	**-11**	**+8**	**+6**
	total	+2	+4	-2	**+5**	-3	**+8**	**+5**	+2	+1	-2
nodes	ACC	**+22**	**+23**	**+31**	**+21**	**+30**	**+21**	**+21**	**+21**	**+21**	**+18**
	CORAL	-3	**-12**	**-13**	**+6**	**-17**	+1	**-14**	**-11**	**-13**	**-9**
	MIPLIB	**+16**	**+20**	**+12**	**+6**	**+13**	**+17**	**+15**	+2	+1	**+6**
	MILP	**-6**	-3	**-8**	-2	-3	+2	**-13**	**-18**	-1	+2
	total	+2	-2	**-5**	**+5**	**-6**	**+6**	**-5**	**-8**	**-7**	-2
gap	ACC	0	0	0	0	0	0	0	0	0	0
	CORAL	0	+1	+2	-3	0	-1	-3	-2	-4	-4
	MIPLIB	**+5**	**+6**	**+5**	+2	+2	**+7**	**+9**	**+7**	+3	+2
	MILP	**+12**	**+6**	+2	+2	**+6**	+1	+2	+4	0	+1
	total	+3	+3	+3	-1	+1	+2	+1	+1	-1	-2

Table 9.1. Performance impact of enabling particular two-row tableau cut separators. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row cut separators (see Chapter 8) are activated. Positive values indicate a deterioration while negative values signify an improvement.

split cut separator ("SP_2"). These two cut separators are also the winners with respect to the number of branching nodes: the number of branching nodes reduces by 13% and 17% respectively. At the same time, however, there is no increase in the amount of integrality gap closed at the root node. This observation again leads to the conclusion that the amount of integrality gap closed is not a reliable indicator of the performance of an MIP solver.

Table 9.1 also reveals that the strengthened two-row cut separators only rarely yield reductions of the solutions times in terms of the relative shifted geometric means. In particular, strengthened cuts are bought at the cost of additional computational effort. The time spent on generating unstrengthened type 1 triangle cuts for the instance `nw04`, for example, is about 0.8 seconds. On the other hand, the generation of strengthened type 1 triangle cuts for the same instance takes about 7.0 seconds. As mentioned above, split cuts are an exception as they can be strengthened via a closed-form formula and thus their strengthening comes virtually free. Apart from increased running times, strengthened two-row cuts lead to considerable reductions of the number of branching nodes as compared with the unstrengthened cut separators. Concerning the type 3 triangle cut separators, the reduction in the number of branching nodes computed on the MILP test set increases from 8% to 18% when the strengthening is applied (see columns "$T3_2$" and "$T3L_2$"). The running times in addition slightly decrease in terms of the relative shifted geometric means. Regarding the unstrengthened and strengthened type 2 triangle cut separators a similar development of the reductions of the number of branching nodes can be observed on the CORAL and MILP test sets (see columns "$T2_2$" and "$T2L_2$"). For the CORAL instances the strengthened quadrilateral cut separator ("GL_2") performs considerably better than its unstrengthened pendant ("G_2"). In contrast, the performance of the unstrengthened split cut separator ("SP_2") is superior to that of the strengthened version ("SPL_2") with respect to running times and number of branching nodes. The amount of integrality gap closed, however, increases when the strengthened split cut separator is applied.

In the following we analyze the computational effectiveness of intersection cuts generated from more than two rows of a simplex tableau. In other words, we consider using higher-dimensional maximal lattice-free convex sets to generate intersection cuts. The results of our evaluation are summarized in Table 9.2 whose structure is similar to that of Table 9.1. For each family of maximal lattice-free convex sets we considered (see Section 9.1) the subscripts in the column headers of Table 9.2 indicate which members of these families were used for cut generation. For instance, the column "$T2_{2-3}$" shows the results obtained by generating intersection cuts from the sets $T2_2$ and $T2_3$. As discussed above, we generate at most fifty cuts for each selected maximal lattice-free convex set. This means that to obtain the results in column "$T2_{2-3}$" we generated at most 100 intersection cuts (50 from $T2_2$ and 50 cuts from $T2_3$). Table 9.2 again indicates that, regarding solution times, the generation of multi-row cuts does not generally lead to an improved performance in terms of reduced shifted geometric means. One possible explanation is that, compared with the generation of two-row cuts, generating intersection cuts from higher-dimensional maximal lattice-free convex sets is computationally expensive. The generation of type 3 triangle cuts ("$T3_2$") for the instance `acc0`, for example, takes about 0.4 seconds while the generation of type 3 triangle and 3-simplex cuts ("$T3_{2-3}$") takes about 2.8 seconds. The

	test set	$T1_{2-6}$	$T2_{2-3}$	$T3_{2-3}$	G_{2-6}	SP_{2-6}	$T1L_{2-6}$	$T2L_{2-3}$	$T3L_{2-3}$	GL_{2-6}	SPL_{2-6}
time	ACC	**+18**	**+20**	**+57**	**+19**	**+18**	**+146**	**+64**	**+59**	**+52**	**+13**
	CORAL	+3	-2	-1	**+5**	**-10**	**+101**	**+56**	**+57**	**+38**	**-11**
	MIPLIB	**+14**	**+15**	**+10**	**+5**	-2	**+177**	**+91**	**+84**	**+52**	**+21**
	MILP	**-5**	**-6**	**-11**	**-15**	+1	**+48**	**+41**	**+47**	**+15**	+2
	total	**+5**	+2	+1	+3	**-5**	**+106**	**+59**	**+59**	**+36**	-1
nodes	ACC	**+24**	**+23**	**+71**	**+27**	**+27**	**+10**	**+16**	**+5**	**+29**	**+21**
	CORAL	**-14**	**-21**	**-14**	-3	**-16**	**-27**	**-8**	**-7**	-1	**-12**
	MIPLIB	+4	**+12**	-2	**+9**	**-12**	**-6**	+1	**+9**	**+9**	**+6**
	MILP	-4	**-7**	**-14**	**-9**	-1	**-21**	-3	+3	**-13**	-3
	total	**-7**	**-10**	**-8**	0	**-12**	**-20**	-4	-1	+1	**-5**
gap	ACC	0	0	0	0	0	0	0	0	0	0
	CORAL	-1	-3	0	-3	-1	-1	-3	-4	-3	-4
	MIPLIB	**+7**	**+7**	+4	**+6**	+3	**+7**	**+8**	**+5**	+4	+2
	MILP	**+12**	**+7**	**+8**	**+6**	+4	**+8**	+4	+1	**-5**	-1
	total	+3	+1	+2	+1	+1	+3	+1	-1	-1	-2

Table 9.2. Performance impact of enabling particular multi-row tableau cut separators. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row cut separators (see Chapter 8) are activated. Positive values indicate a deterioration while negative values signify an improvement.

increase in the separation time is caused by the number of variations (i.e. rotations and permutations) of $T3_3$ we consider. To obtain a single type 3 3-simplex cut we actually compute $2^3 \cdot 3! = 48$ intersection cuts and select the one with the largest distance cut off relative to the fractional LP solution (see Section 9.3). On the MILP test set, however, improvements in terms of running times can be achieved by generating intersection cuts from more than two rows. For example, the quadrilateral cut separator ("G_2") reduces the running times by 5% for the MILP instances. By considering up to six rows simultaneously ("G_{2-6}"), we can improve upon this result and obtain a reduction of 15%. With respect to the number of branching nodes computed, our analysis above showed that the two-row cut separators are particularly effective on the CORAL and MILP test sets. Table 9.2 also shows that generating intersection cuts from more than two rows can further reduce the number of branching nodes. For the CORAL instances the reduction in the number of branching nodes yielded by the type 1 triangle cut separator ("$T1_2$") is 3%. The reduction obtained by using a multi-row relaxation consisting of up to six rows ("$T1_{2-6}$") is 14%. On the CORAL and MILP instances a similar progress in the reduction in the number of branching nodes can be observed for all cut separators except for the split cut separator (see columns "$T1_{2-6}$" to "G_{2-6}"). Concerning the amount of integrality gap closed at the root node, those cut separators deriving intersection cuts from more than two rows of the simplex tableau perform slightly better than the two-row cut separators in terms of the relative shifted geometric mean values computed for the total test set.

Like two-row intersection cuts, general multi-row intersection cuts can also be strengthened using the integrality requirements on some of the non-basic variables (see Equation (9.19)). Columns "$T1L_{2-6}$" to "SPL_{2-6}" of Table 9.2 report on results for the strengthened multi-row intersection cuts using between two and six rows. Table 9.2 allows for the direct observation that the solution times increase considerably if the strengthening is applied. Generating type 1 simplex cuts ("$T1_{2-6}$"), for example, produces a deterioration of the solution times by about 14% on the MIPLIB instances as compared with the single-row cut separators. If the strengthening is applied ("$T1L_{2-6}$"), the relative shifted geometric mean further deteriorates to a value of 177%. This deterioration is to some extent caused by the computational expensiveness of generating strengthened intersection cuts from multiple (i.e. more than two) rows of the simplex tableau (see Section 9.3.1). For example, generating unstrengthened type 1 simplex cuts ("$T1_{2-6}$") for the instance `nw04` takes about 12.2 seconds. The time spent on separating the strengthened versions of these cuts ("$T1L_{2-6}$") is 515.5 seconds. The strengthened type 1 simplex cuts yield reductions of the number of branching nodes on all test sets except for the ACC test set. Concerning the MILP test set, the unstrengthened cuts ("$T1_{2-6}$"), for example, compute 4% fewer branching nodes than the reference setting (i.e. the single-row cut separators) while a reduction of about 20% is realized by applying the strengthening ("$T1L_{2-6}$"). Strengthened cuts are, on the other hand, not guaranteed to produce smaller node counts. On the CORAL test set the strengthened type 2 and type 3 simplex cut separators compute significantly more branching nodes than the unstrengthened versions of these separators (see columns "$T2_{2-3}$" and "$T2L_{2-3}$" or "$T3_{2-3}$" and "$T3L_{2-3}$" respectively). Similar to the results discussed above, the separation of strengthened intersection cuts from more than two rows of the simplex tableau does not significantly increase the amount of integrality gap closed at the root node.

So far in this section we have evaluated the individual computational effectiveness of various multi-row cut separators. Several conclusions can be drawn from our experiments.

Firstly, our computational experiments revealed that, regarding solution times, the multi-row cut separators are only sporadically competitive with the single-row cut separators. One may argue that this result is not surprising since the separation of multi-row cuts is computationally more demanding than, for example, the separation of Gomory mixed-integer cuts. In particular, the multi-row cut separators turned out to be ineffective on the ACC and MIPLIB instances. A possible explanation is that most instances in the MIPLIB test set are easy in the sense that they can rapidly be solved using the single-row cut separators. Concerning the ACC instances, the rather dense tableau rows especially make the generation of intersection cuts from more than two rows of the simplex tableau relatively time-consuming. Plain (unstrengthened) multi-row intersection cuts can nevertheless be obtained very efficiently as explained in Section 9.3. Our experiments suggest that the separation times are acceptable even if a multi-row relaxation which consists of more than two rows is used for cut generation. For example, if type 2 3-simplex cuts are separated in addition to type 2 triangle cuts the mean value of the overall solution times is not decisively influenced (see Tables 9.1 and 9.2) because the separation times only increase by a reasonable amount. Similar to their unstrengthened pendants, strengthened intersection cuts from multiple rows only rarely yield performance improvements in terms of reduced running times. In the two-row case the strengthening is mostly cost-efficient in the sense that the increases in the overall running times of the separators are acceptable. Our experiments, however, also indicate that computing strengthened multi-row cuts, especially from more than two rows (except for split cuts), is in general very time-consuming.

Besides computation times, our computational study indicates, secondly, that separating multi-row cutting planes can positively affect the enumeration during the branch-and-bound algorithm. Configurations applying particular multi-row cut separators compute significantly fewer branching nodes on several test sets.

Concerning the effectiveness of strengthening, recall that the only minimal intersection cuts we compute are strengthened type 1 and type 2 triangle cuts (see Section 9.3.1). Aside from the fact that the intersection cuts we compute are thus not minimal in general, our experiments demonstrate that strengthened cuts can lead to an improved performance in terms of reducing the number of branching nodes computed. In particular, we showed that intersection cuts from more than two rows of the simplex tableau can successfully be strengthened. On the other hand, the influence of the separation of multi-row cuts on the amount of integrality gap closed at the root node is unclear. On selected test sets multi-row cut separators reduce the integrality gap while they lead to increased integrality gaps on other test sets.

Thirdly, we offer some conclusions concerning the performance of multi-row cuts as compared with split cuts. The separation of split cuts was treated in Chapter 8. The difference between split cuts and the remaining classes of intersection cuts we considered in this chapter is that the latter can not directly be derived as split cuts from the current formulation of the LP relaxation. As mentioned above, it was, however, recently shown by Dey and Louveaux [76] that all triangle cuts (except for type 1 triangle cuts) and quadrilateral cuts have finite split rank, implying that these cuts can also be obtained by iteratively applying split cuts. To ensure high comparability our split cut separator was integrated into the framework used for the separation of multi-row cuts. Our computational experiments showed that split cuts are clearly superior to multi-row cuts as far as separation times are concerned. Compared with multi-row cuts, split cuts can, in addition, be strengthened very efficiently. For some test sets, larger reductions of the number of branching nodes computed can, on the other hand, be obtained by applying the multi-row cut separators.

We conclude from the previous discussion that, although multi-row cuts lead to notable improvements in performance for certain instances or test sets, their general effect on the overall performance of an MIP solver like Mops is not univocal. In particular, it is not always cost-efficient to apply the multi-row cut separators, especially for those instances where the fast single-row cut separators are effective. In Section 8.9 we showed that separating lift-and-project cuts (or reduce-and-split cuts, etc.) for instances which are hard to solve using GMI cuts is beneficial. Similarly, we suppose that multi-row cuts play a more important

		test set	$T1_2$	$T2_2$	$T3_2$	G_2	SP_2	$T1L_2$	$T2L_2$	$T3L_2$	GL_2	SPL_2
easy	time	ACC	**+33**	**+59**	**+66**	**+46**	**+81**	**+52**	**+58**	**+58**	**+49**	**+42**
		CORAL	**+13**	**+5**	+3	**+27**	-1	**+50**	**+17**	**+24**	**+10**	+3
		MIPLIB	**+18**	**+30**	**+32**	**+34**	**+18**	**+25**	**+52**	**+33**	**+16**	**+7**
		MILP	**-39**	**-36**	**-18**	**-21**	**-39**	**+145**	**+136**	**+46**	**+105**	**+63**
		total	**+14**	**+14**	**+14**	**+28**	**+7**	**+42**	**+32**	**+28**	**+15**	**+8**
	nodes	ACC	**+70**	**+74**	**+108**	**+68**	**+101**	**+68**	**+68**	**+68**	**+68**	**+58**
		CORAL	**+14**	**-8**	+4	**+35**	**-16**	**+42**	**-6**	+2	**+5**	+4
		MIPLIB	**+15**	**+18**	**+16**	**+15**	**+17**	**+16**	**+13**	**+7**	0	**+9**
		MILP	**-11**	**-11**	**-17**	**-19**	**-15**	**-18**	**-9**	**-8**	+3	**+22**
		total	**+16**	**+5**	**+11**	**+26**	+1	**+30**	+4	**+6**	+4	**+8**
	gap	ACC	0	0	0	0	0	0	0	0	0	0
		CORAL	0	+1	+2	+1	-1	0	0	+1	+1	-2
		MIPLIB	+3	**+6**	+4	0	+2	**+5**	**+6**	+4	+2	+2
		MILP	**+59**	**+59**	0	0	**+59**	**-45**	0	0	0	0
		total	+2	+4	+3	0	+1	+2	+2	+2	+1	0
hard	time	ACC	0	0	0	0	0	0	0	0	0	0
		CORAL	-3	-3	**-13**	+2	**-12**	**-10**	**-6**	**-5**	**-9**	**-10**
		MIPLIB	**+15**	**+30**	**+7**	**-11**	**+5**	**+24**	**+19**	-4	+4	+2
		MILP	**-6**	0	**-8**	**-5**	-3	**+6**	**-7**	**-14**	+4	+4
		total	-1	+2	**-9**	-2	**-8**	-2	-3	**-7**	-4	**-5**
	nodes	ACC	0	0	0	0	0	0	0	0	0	0
		CORAL	**-13**	**-15**	**-23**	**-8**	**-18**	**-18**	**-19**	**-18**	**-23**	**-16**
		MIPLIB	**+24**	**+29**	**+5**	**-11**	**+6**	**+23**	**+22**	**-6**	+2	+2
		MILP	**-6**	-3	**-8**	-1	-3	+3	**-14**	**-19**	-1	+1
		total	**-6**	**-6**	**-15**	**-7**	**-11**	**-8**	**-12**	**-16**	**-15**	**-9**
	gap	ACC	0	0	0	0	0	0	0	0	0	0
		CORAL	-1	+1	+2	**-6**	+1	-2	**-5**	-3	**-7**	**-5**
		MIPLIB	**+11**	+4	**+8**	**+8**	+2	**+11**	**+17**	**+12**	**+6**	+1
		MILP	**+10**	+3	+2	+2	+3	+3	+2	+4	0	+1
		total	+4	+2	+3	-2	+1	+1	0	+1	-3	-3

Table 9.3. Performance impact of enabling particular two-row tableau cut separators on easy and hard instances. We consider an instance to be easy if it can be solved to optimality using the single-row cut separators (see Chapter 8) in 60 seconds or less. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row cut separators are activated. Positive values indicate a deterioration while negative values signify an improvement.

		test set	$T1_{2-6}$	$T2_{2-3}$	$T3_{2-3}$	G_{2-6}	SP_{2-6}	$T1L_{2-6}$	$T2L_{2-3}$	$T3L_{2-3}$	GL_{2-6}	SPL_{2-6}
easy	time	ACC	**+65**	**+75**	**+262**	**+72**	**+66**	**+911**	**+294**	**+263**	**+225**	**+48**
		CORAL	**+34**	**+20**	**+18**	**+42**	+4	**+556**	**+256**	**+240**	**+216**	**+29**
		MIPLIB	**+46**	**+46**	**+46**	**+32**	**+17**	**+494**	**+274**	**+248**	**+151**	**+62**
		MILP	**+19**	**-21**	**-23**	**-32**	**-12**	**+1704**	**+637**	**+487**	**+299**	**+19**
		total	**+38**	**+29**	**+33**	**+36**	**+11**	**+530**	**+259**	**+238**	**+185**	**+39**
	nodes	ACC	**+78**	**+74**	**+299**	**+91**	**+90**	**+36**	**+53**	**+16**	**+100**	**+69**
		CORAL	**+5**	-4	+4	**+44**	**-12**	**+5**	**+14**	**+8**	**+54**	**+32**
		MIPLIB	**+7**	**+20**	**+8**	**+25**	-2	+2	**+11**	**+19**	**+21**	**+19**
		MILP	**-6**	**-8**	**-18**	-4	**-8**	**-8**	**+15**	**-16**	**+13**	-4
		total	**+8**	**+8**	**+11**	**+35**	**-5**	+4	**+13**	**+13**	**+39**	**+26**
	gap	ACC	0	0	0	0	0	0	0	0	0	0
		CORAL	-1	0	0	0	-2	-1	-2	-2	-1	-1
		MIPLIB	**+5**	**+7**	+3	**+6**	+3	**+7**	**+5**	+4	**+5**	+4
		MILP	**+59**	**+59**	**+59**	**+59**	0	**-45**	0	0	**-117**	**-84**
		total	+3	+4	+2	+4	0	+2	+1	+1	+1	0
hard	time	ACC	0	0	0	0	0	**+6**	+1	+1	+1	0
		CORAL	**-6**	**-9**	**-8**	**-5**	**-16**	**+21**	**+14**	**+18**	-1	**-24**
		MIPLIB	0	+2	**-10**	**-10**	**-17**	**+71**	**+24**	**+24**	**+16**	**+5**
		MILP	**-6**	**-6**	**-11**	**-15**	+2	**+26**	**+28**	**+37**	**+7**	+1
		total	**-5**	**-7**	**-9**	**-8**	**-12**	**+28**	**+18**	**+22**	+3	**-14**
	nodes	ACC	0	0	0	0	0	-4	-1	-1	-1	0
		CORAL	**-24**	**-30**	**-23**	**-23**	**-19**	**-42**	**-20**	**-15**	**-24**	**-31**
		MIPLIB	-2	-4	**-21**	**-17**	**-33**	**-22**	**-17**	**-9**	**-13**	**-17**
		MILP	**-5**	**-7**	**-15**	**-10**	-1	**-22**	-4	+4	**-14**	-3
		total	**-16**	**-21**	**-20**	**-19**	**-17**	**-34**	**-15**	**-10**	**-20**	**-23**
	gap	ACC	0	0	0	0	0	0	0	0	0	0
		CORAL	0	**-6**	0	**-6**	0	-1	-4	**-6**	**-5**	**-7**
		MIPLIB	**+12**	**+6**	**+9**	**+6**	+2	**+6**	**+17**	**+9**	+2	-1
		MILP	**+9**	+4	**+6**	+4	+4	**+11**	+4	+1	0	+3
		total	+4	-2	+2	-2	+1	+3	+1	-2	-3	-4

Table 9.4. Performance impact of enabling particular multi-row tableau cut separators on easy and hard instances. We consider an instance to be easy if it can be solved to optimality using the single-row cut separators (see Chapter 8) in 60 seconds or less. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row cut separators are activated. Positive values indicate a deterioration while negative values signify an improvement.

role in solving hard instances (in the sense hard to solve with the single-row cut separators) than in solving instances where applying the single-row cut separators suffices. We consider an instance to be hard if it can not be solved to optimality with the single-row cut separators within a time limit of 60 seconds. Tables 9.3 and 9.4 summarize the performance of the multi-row cut separators on easy and hard instances respectively. The detailed results can be found in Tables B.36 to B.40 and Tables B.41 to B.45 in Appendix B. Concerning the easy instances performance deteriorations can mainly be observed, especially if the strengthened multi-row cut separators are applied. On the other hand, a large number of improvements can be seen on the hard instances. For these reasons we suggest use of a separation scheme in which the multi-row cut separators are only applied if the single-row cut separators are not effective. For example, the multi-row cut separators should be activated if the single-row cut separators do not succeed in noticeably improving the dual bound for a number of rounds of IP preprocessing.

We should like to emphasize that we draw our conclusions from (shifted geometric) mean values. Conclusions drawn for a specific test set do not therefore necessarily apply to every single instance in this test set. This means in particular that while multi-row cut separators deteriorate the overall performance on the ACC and MIPLIB test sets in our experiments in terms of the mean values, these cut separators may nevertheless be beneficial on certain instances in these test sets.

In the preceding part of this section, we analyzed the individual computational usefulness of various multi-row cut separators. A question which has so far remained unanswered is how the multi-row cut separators interact. We are especially interested in whether an improved performance can be obtained by applying several multi-row cut separators together. To keep the number of test runs manageable we limit our investigation to the two-row cut separators. In future research we plan also to investigate the interaction of cut separators which use more than two rows of the simplex tableau simultaneously. In our second experiment, which is meant to answer the questions raised above, we therefore consider a reference setting which applies all single-row (see Chapter 8) and two-row cut separators. We then disable specific two-row cut separators and examine the performance improvement or deterioration. Table 9.5 presents a summary of the results of this experiment. The detailed results can be found in

	test set	only SR	no $T1_2$	no $T2_2$	no $T3_2$	no G_2	no SP_2	no $T1L_2$	no $T2L_2$	no $T3L_2$	no GL_2	no SPL_2
time	ACC	**-16**	**-10**	+1	**-10**	**-10**	**-6**	**-10**	**-11**	**-11**	**-10**	**-10**
	CORAL	**-5**	**+6**	-4	+4	+2	**+8**	-2	**-9**	-4	+3	-2
	MIPLIB	**-30**	-4	**-7**	-2	+2	-2	**-6**	**-7**	**-9**	-3	**-6**
	MILP	**+7**	+1	0	+3	**+5**	+3	**+5**	+1	**+22**	-3	**+5**
	total	**-10**	+2	-4	+2	+2	+4	-3	**-7**	-2	0	-3
nodes	ACC	**-18**	**-13**	+1	**-13**	**-12**	**-14**	**-12**	**-13**	**-12**	**-12**	**-12**
	CORAL	**+20**	**+6**	**-6**	+3	**+5**	**+9**	-3	**-11**	-1	+4	-3
	MIPLIB	**-22**	**-7**	-3	+3	-2	-2	**-15**	**-5**	**-9**	**-5**	**-15**
	MILP	**+20**	-4	**-9**	-3	-3	-1	+2	-4	**+23**	-3	+2
	total	**+5**	0	**-6**	+2	+1	+4	**-6**	**-9**	-1	0	**-6**
gap	ACC	0	0	0	0	0	0	0	0	0	0	0
	CORAL	**+5**	0	0	-1	+1	0	+1	0	0	0	+1
	MIPLIB	**-6**	+1	+1	+1	0	+2	+3	+2	+2	+1	+3
	MILP	-3	+2	**+5**	+1	0	0	**+6**	+4	+1	+1	**+6**
	total	+1	0	+1	0	0	+1	+2	+1	+1	0	+2

Table 9.5. Performance impact of disabling particular two-row tableau cut separators. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which all single-row (see Chapter 8) and two-row cut separators are activated. Positive values indicate a deterioration while negative values signify an improvement.

Tables B.46 to B.50 in Appendix B. As shown by the results of our first experiment, it is difficult for our two-row cut separators to compete with our single-row cut separators regarding solution times. If only the single-row cut separators ("only SR") are activated, the solution times for the ACC, CORAL and MIPLIB instances decrease by 16%, 5% and 30% respectively. Concerning the entire test set ("total"), the largest improvement in the solution times is also caused by the deactivation of all two-row cut separators. For the CORAL test set, the running times, by contrast, increase by 7% if only the single-row cut separators are executed. Compared with the reference setting, the largest deterioration in the solution times (22%) is obtained by deactivating the strengthened type 3 triangle cut separator ("no $T3L_2$") on the MILP instances. We note that this deterioration is larger than that produced by deactivating all two-row cut separators. This observation indicates that strengthened type 3 triangle cuts are especially effective on the MILP test set and also points to the interaction of the different two-row cut separators. In our study of the individual strength of the two-row cut separators the strengthened type 3 triangle cut separator was also most effective on the MILP instances (see Table 9.1). Also in line with the results of our first experiment, the two-row cut separators considerably reduce the number of branching nodes computed for the CORAL and MILP instances. The number of branching nodes reduces by 20% for both test sets. Regarding the two-row cut separators, the largest increase in the number of branching nodes (9%) is caused by the deactivation of the split cut separator ("no SP_2"). When the strengthened type 3 cut separator is disabled the number of branching nodes increases by 23%. On the other hand, disabling the strengthened type 2 triangle cut separator ("no $T2L_2$") consistently leads to improvements in terms of the shifted geometric means of the number of branching nodes computed. Table 9.5 also shows that the effect of disabling specific two-row cut separators on the amount of integrality gap closed at the root node is rather small. Regarding the entire test set ("total") only small deteriorations arise if two-row cut separators are deactivated.

Chapter 10.

Cutting Plane Selection and Management

In the preceding two chapters we described various cut separation algorithms and discussed implementation details. Typically, these algorithms generate a large number of cutting planes.

In this chapter we address the problem of selecting only a subset of the generated cuts. To this end we discuss the reliability of different cut selection rules or cut quality measures respectively and point out their merits and demerits. We also describe a cut selection algorithm and highlight important implementation details. Finally, we present computational results to emphasize the importance of cut selection algorithms and study the impact of different cut quality measures on the overall performance of MOPS.

This chapter is organized as follows. After giving a short introduction in Section 10.1, we present different cut quality measures in Section 10.2. The cut pool and our cut selection algorithm are described in Section 10.3. Finally, Section 10.4 discusses computational results.

10.1. Introduction

Cutting planes play a central role in solving MIPs. Closely related to cut generation is the problem of reducing the number of cuts that are added to the LP relaxation. This problem in particular has become increasingly important since there are numerous cutting planes available in MIP solvers like CPLEX [115], XPRESS-MP [74] or MOPS [139]. If the number of generated cuts is large, adding all of them represents a computational burden for the LP solver, possibly leading to longer (node) solution times. Thus finding cut quality measures and developing cut selection algorithms are central issues in the study of cutting plane algorithms.

10.2. Cut Selection

Crucial to solving MIPs is a strong LP relaxation. There are a number of techniques such as bound and coefficient reduction to improve the strength of the LP relaxation. However, cutting planes like Gomory mixed-integer cuts, cover cuts, flow cover cuts, flow path cuts, mixed-integer rounding cuts, etc. are very important to obtaining a tight LP relaxation.

The primary aim of cutting plane selection is to choose the "right" cuts which help to solve an MIP problem more rapidly. The essential question is how to select such a subset of cuts. To answer this question reliable cut quality measures are needed. An additional aim of cutting plane selection is to keep the number of cuts that are added to the LP relaxation small.

10.2.1. Brief Literature Review

Padberg and Rinaldi [146] emphasize that "finding a reasonable quality measure is one of the central issues in the area of polyhedral cutting-plane algorithms that is - as of today - not yet investigated satisfactorily"([146], p. 79). Although this statement was made over fifteen years ago, the situation has not fundamentally changed. Several contributions on cutting plane theory address measuring cut quality as a secondary problem. Balas et al. [26] point out that using the improvement of the objective function as a cut quality measure has certain drawbacks (e.g. zero gap problems) and suggest calculating the Euclidean distance between a given LP solution and the cutting plane instead. Christof and Reinelt [50] propose preference be given to cutting planes that are as parallel as possible to the objective function and try to estimate the improvement of the objective function. A different proposal put forward by Ferris et al. [85] is called "cut selection by usage". Here the policy is to add all generated cuts tentatively and then track the pivots performed by the dual simplex method. If a cut is used, meaning that is was pivoted on, it is marked and later on only marked cuts are added. To increase the diversity of cuts, Andreello et al. [13] check cuts pairwise for parallelism (see also [85]). Achterberg [4] proposes a sophisticated cut selection algorithm which combines several cut quality measures.

10.2.2. Cut Quality Measures

Assume that a mixed-integer program is given in the form

$$\max\{cx : Ax \leq b, x \geq 0, x_j \in \mathbb{Z}, \forall j \in N_I\} \tag{10.1}$$

where $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$, and $N_I \subseteq N = \{1,\dots,n\}$. Let x^* be an optimal solution to the corresponding LP relaxation. Suppose that some separation algorithm generated a number of cutting planes

$$\alpha^i x = \sum_{j\in N} \alpha^i_j x_j \leq \beta^i, \quad \forall i \in L, \tag{10.2}$$

where $\alpha^i \in \mathbb{R}^n$ and $\beta^i \in \mathbb{R}$ and L is an arbitrary index set. To simplify the notation we omit the index i whenever possible and denote a single cut by

$$\alpha x = \sum_{j\in N} \alpha_j x_j \leq \beta. \tag{10.3}$$

Some known quality measures for cutting planes are the following.

Violation The simplest quality measure for cutting planes is *violation* which is defined as

$$v\left(\alpha, \beta, x^*\right) = \alpha x^* - \beta. \tag{10.4}$$

Using violation to measure the quality of cutting planes has several drawbacks. For example, violation is not invariant under scaling. Given any integer $k \in \mathbb{Z}_+$, the cut $k\alpha x \leq k\beta$ is k times more violated than $\alpha x \leq \beta$. Observe that the value of $v(\alpha, \beta, x^*)$ is positive for violated cuts with this definition.

Relative Violation A straightforward approach to overcome issues related to scaling is to divide the violation value by the absolute value of the right-hand side, i.e. normalizing the right-hand side. The resulting quality measure is called *relative violation* (cf. [130]) and is defined as follows:

$$r\left(\alpha, \beta, x^*\right) = \frac{\alpha x^* - \beta}{|\beta|} = \frac{v\left(\alpha, \beta, x^*\right)}{|\beta|} \tag{10.5}$$

However, it is not possible to compute the relative violation if the right-hand side of the cut is zero. In this case we instead use the violation.

Distance A third possibility to measure the cut quality is to determine the *Euclidean distance* between a given LP solution x^* and the hyperplane $\alpha x = \beta$. In other words, we consider the distance between x^* and its orthogonal projection on this hyperplane:

$$d\left(\alpha, \beta, x^*\right) = \frac{\alpha x^* - \beta}{\|\alpha\|} = \frac{v\left(\alpha, \beta, x^*\right)}{\|\alpha\|} \tag{10.6}$$

Using the Euclidean distance can be seen as an improvement of relative violation and overcomes some of its drawbacks. Like relative violation, the Euclidean distance is independent of scaling. In addition it can also be calculated for cuts whose right-hand sides take arbitrary values.

Objective Function Parallelism Suppose that c and α are linearly dependent, for example $\alpha = kc$, $k \in \mathbb{R}$ and $k > 0$. After substitution of α the cut (10.3) reads $(kc)x \leq \beta$ which is equivalent to $cx \leq \frac{\beta}{k}$. This means that $\frac{\beta}{k}$ is an upper bound on the value of the objective function. Therefore cutting planes with α *as parallel as possible to the objective function* are preferable (cf. [50]). Parallelism is measured by inspecting the cosine of the angle between c and α:

$$o\left(\alpha\right) = \frac{\alpha c}{\|\alpha\|\,\|c\|} \tag{10.7}$$

If $o(\alpha) = 1$, the cutting plane is parallel to the objective function.

Expected Improvement An intuitive method for measuring the quality of a cut is to inspect the improvement of the value of the objective function after the cut has been added and the LP has been reoptimized. Due to its computational cost (add cut, reoptimize, remove cut) this approach is normally not practicable. Although the exact improvement of the value of the objective function is therefore unknown, the improvement can be estimated under certain assumptions. Let x^*_{exp} be the *expected next LP solution* (cf. [50]). As we are maximizing, we expect the value of the objective function to decrease (or remain the same) after a violated cut has been added, i.e. $cx^* - cx^*_{exp} \geq 0$. Taking $-\alpha$ as the direction of reoptimization, e.g. $x^*_{exp} = x^* - \delta\alpha$, and assuming that $\alpha x^*_{exp} = \beta$ gives

$$\delta = \frac{\alpha x^* - \alpha x^*_{exp}}{\|\alpha\|^2} = \frac{\alpha x^* - \beta}{\|\alpha\|^2} = \frac{v\left(\alpha, \beta, x^*\right)}{\|\alpha\|^2}. \tag{10.8}$$

The value of the objective function for x^*_{exp} takes the form

$$cx^*_{exp} = cx^* - \frac{\alpha c \cdot v\left(\alpha, \beta, x^*\right)}{\|\alpha\|^2} = cx^* - \frac{\alpha c}{\|\alpha\|} \cdot d\left(\alpha, \beta, x^*\right), \tag{10.9}$$

and the quality of a cut can, for instance, be measured by calculating

$$\begin{aligned} e\left(\alpha, \beta, x^*\right) = cx^* - cx^*_{exp} &= \frac{\alpha c}{\|\alpha\|} \cdot d\left(\alpha, \beta, x^*\right), \\ &= \|c\| \cdot o\left(\alpha\right) \cdot d\left(\alpha, \beta, x^*\right). \end{aligned} \tag{10.10}$$

It is quite easy to see from (10.10) that the expected improvement is a combination of other previously defined quality measures. Not only is the prediction of the improvement of the objective function highly dependent on the assumptions, measuring cut quality this way is problematic when the objective gap between the optimal solution of the LP relaxation and the optimal MIP solution is zero.

Support The support of the coefficient vector α is the set of its non-zero positions $\bar{N} = \{j \in N : |\alpha_j| > 0\}$. One can compute the percentage of all variables that is present in the cut

$$s\left(\alpha\right) = \frac{\left|\bar{N}\right|}{|N|}. \tag{10.11}$$

Cutting planes which have a coefficient vector α which has a large value of $s(\alpha)$ are called *dense*. Dense cuts are likely to slow down computations in the simplex algorithm (e.g. the LU factorization) and can possibly cause numerical difficulties.

Since adding these cuts is thus not desirable, we select cuts with minimal support, i.e. we use the quality measure $1 - s(\alpha)$.

Integral Support Let the set $\bar{N}$ be defined as in the previous paragraph. The set $\bar{N}_I = \bar{N} \cap N_I$ consists of those non-zero positions of α which correspond to integer-constrained variables. Using this information it is possible to calculate the ratio between the number of integer-constrained variables and the total number of variables present in a cut

$$i(\alpha) = \frac{\left|\bar{N}_I\right|}{\left|\bar{N}\right|}. \tag{10.12}$$

One may argue that a cut having non-zero coefficients on many (possibly fractional) integer variables is preferable to a cut which consists mostly of continuous variables.

Parallelism Let $\alpha^i x \leq \beta^i$ and $\alpha^k x \leq \beta^k$ be two cuts generated by some cut separation routine. If their defining hyperplanes are parallel, one of the two cuts is dominated by the other and should consequently not be added to the LP relaxation. Again, the absolute value of the cosine of the angle between the two hyperplanes (or vectors) is used to detect *parallelism*:

$$p\left(\alpha^i, \alpha^k\right) = \frac{\left|\alpha^i \alpha^k\right|}{\|\alpha^i\| \, \|\alpha^k\|} \tag{10.13}$$

If $p(\alpha^i, \alpha^k) = 1$, the hyperplanes which correspond to the two given cuts are parallel. In contrast, $p(\alpha^i, \alpha^k) = 0$ indicates that the two cuts are orthogonal.

Distance Variant I A problem of the Euclidean distance is that it does not work properly with a system of equations (cf. [56]). This is because a polyhedron whose feasible points have to satisfy a number of equations is not full-dimensional. In other words, measuring cut quality by the Euclidean distance does not take into account that a solution must exactly lie at the intersection of all existent hyperplanes. For example, consider the simple sets

$$X^R = \left\{x \in \mathbb{R}^2_+ : 6x_1 + 4x_2 = 20\right\} \text{ and } X = X^R \cap \mathbb{Z}^2_+, \tag{10.14}$$

with a fractional solution of $x_1^* = 3\frac{1}{3}$ and $x_2^* = 0$ to the relaxation X^R. The hyperplanes geometrically representing the two cuts $3x_1 + x_2 \leq 8$ and $x_1 \leq 2$ have different Euclidean distances from x^*, meaning that one of them is more favorable. But this is actually not true. Suppose that we add each cut to the formulation of X^R individually. After reoptimization X^R has an optimal integral solution $x_1' = 2$, $x_2' = 2$ in both cases. Thus each of the cuts alone excludes the fractional solution x^* from X^R. A way to solve this problem is computing the *rotated steepness* (cf. [56]).

Consider an example with two cuts with indices i and k, i.e. $L = \{i, k\}$ (see Equation (10.2)), and let $\beta^i = \beta^k$ and (N_1, N_2) be a partition of N with

$$\begin{aligned} \alpha_j^i = \alpha_j^k \geq 0, &\quad \forall j \in N_1, \\ \alpha_j^i > \alpha_j^k \geq 0, &\quad \forall j \in N_2. \end{aligned}$$

Furthermore, suppose that the LP solution x^* to the underlying problem fulfills the condition $x_j^* = 0, \forall j \in N_2$ and violates both cuts specified above. Clearly, cut i (mathematically) dominates cut k under these circumstances. But due to the structure of the example identical violations and relative violations are assigned to both cuts. Moreover, it follows from the assumed relations between α^i and α^k that $\|\alpha^i\| > \|\alpha^k\|$. Thus the Euclidean distance between the cut hyperplane of cut k and x^* exceeds the Euclidean distance between the cut hyperplane of cut i and x^*.

A variable having the characteristics that $\alpha_j > 0$ and $x_j^* = 0$ for some $j \in N$ does not affect the numerator of (10.6). The denominator of (10.6), namely the Euclidean norm of α, is however increased by this variable. It is possible to obtain a simple variation of the Euclidean distance by modifying the Euclidean norm for variables with index j satisfying $x_j^* = 0$. Define the vector $\bar{\alpha} \in \mathbb{R}^n$ with

$$\bar{\alpha}_j = \begin{cases} \alpha_j & \text{if } x_j^* \neq 0, \\ 0 & \text{otherwise}, \end{cases} \tag{10.15}$$

for $j \in N$. We obtain

$$\tilde{d}\,(\alpha, \beta, x^*) = \frac{v\,(\alpha, \beta, x^*)}{\|\bar{\alpha}\| + 1}. \tag{10.16}$$

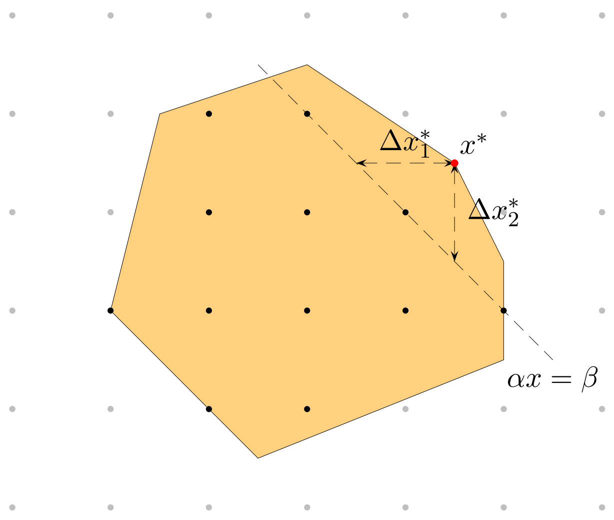


Figure 10.1. Computing the deviation in each coordinate direction

In order to avoid division by zero, we increase the denominator by one.

Distance Variant II We now consider the deviation of the LP solution x^* from a given cut hyperplane in each variable direction and multiply them. In other words, given the LP solution x^*, let the point p_k be the intersection of the ray starting in x^* in coordinate direction k with the cut hyperplane $\alpha x = \beta$ (see Figure 10.1). The vectors x^* and p_k only differ in the k^{th} component by the amount Δx_k^* which can be obtained by calculating

$$\Delta x_k^* = \frac{\alpha x^* - \beta}{|\alpha_k|}, \tag{10.17}$$

where, as above, $k \in \bar{N} = \{j \in N : |\alpha_j| > 0\}$. To simplify the notation we assume that $\bar{N} = \{1, \dots, p\}$ in the following. Multiplying the deviations Δx_k^* for all $k \in \bar{N}$ yields

$$\prod_{k \in \bar{N}} \Delta x_k^* = \Delta x_1^* \cdot \Delta x_2^* \cdots \Delta x_p^* = \frac{(\alpha x^* - \beta)^p}{|\alpha_1 \cdot \alpha_2 \cdots \alpha_p|}. \tag{10.18}$$

In two-dimensional space, the product of the deviations computed in Equation (10.18) is twice the area of the triangle described by x^* and its projections p_k on the cut hyperplane $\alpha x = \beta$ (see Figure 10.1). Typically, not all generated cutting planes have non-zero coefficients on the same number of variables. In order to improve the comparability we extract the p^{th} root and raise the resulting expression to the power of $k \geq 1$. We obtain

$$\hat{d}\left(\alpha, \beta, x^*, k\right) = \frac{v\left(\alpha, \beta, x^*\right)^k}{\left(\left|\alpha_1 \cdot \alpha_2 \cdots \alpha_p\right|\right)^{\frac{k}{p}}}. \tag{10.19}$$

Note that we assume that x^* violates the inequality $\alpha x \leq \beta$. Otherwise even values of k would change the sign of the numerator of Equation (10.19). It is easy to show that (10.19) is invariant to scaling.

10.3. Cut Pool and Cut Selection Algorithm

The term *cut pool* (cf. [146]) is used for a data structure which is designed to offer an intermediate storage area for generated cutting planes. In our implementation the cut pool uses a packed storage, meaning that only non-zero entries and their indices are stored (see Figure 7.3). All cuts are stored as less-or-equal inequalities, i.e. $\alpha x \leq \beta$.

Several cutting plane techniques are available in MOPS. So far, all generated cuts which are violated by the current solution are added to the LP relaxation. That this policy dramatically increases the problem size motivated the idea of studying and implementing cut management and selection techniques.

The MOPS MIP solver usually generates cuts at the root node of the branch-and-bound tree (cut-and-branch). Prior to the branch-and-bound algorithm the so-called supernode processing (cf. [157]) or IP preprocessing is performed. Besides the classes of cutting planes treated in this thesis, MOPS separates clique [65], bound implication [157], cover [107], flow cover [166] and mixed-integer rounding cuts [135] with the objective of tightening the LP relaxation. The cut generation process is iteratively continued until there is no improvement in the dual bound or a maximum number of rounds (typically twenty) is reached. There is no additional scheme for deactivating specific cut separators. Cut pool and cut selection are at the moment only used at root node. However, it should not take too much effort to adapt the implementation in such a way that it can be used at any branching node.

In the following we discuss the details of our implementation which is also described in Algorithm 10.1. A detailed description of a cut selection algorithm for $\{0, \frac{1}{2}\}$-cuts can be found in [13].

- In order to minimize work related to changes in the existing implementation of supernode processing, we changed the cut separation routines in such a way that they optionally add violated cuts to the cut pool instead of adding them directly to the LP relaxation. If possible, coefficients are scaled to integer values whenever a cut is added to the cut pool to avoid numerical problems. The (Euclidean) norm of a cut and the parallelism to the objective function (see Equation (10.7)) are computed and stored.

- After each round of cut generation, in which different cutting plane separators are executed, the cut selection is performed and the cut pool is scanned for cutting planes which seem to be promising with respect to some quality measure. If a cut is selected and *activated*, i.e. added to the problem formulation, it still remains in the cut pool. But in order to avoid adding this cut again, we mark it as activated. Having added the selected cuts, the LP is reoptimized and a new round of cut generation starts.

- We use the algorithm of Tomlin and Welch [164] to identify duplicate (i.e. exactly parallel) cuts. This algorithm is a very fast and specialized method which was originally developed to identify duplicate rows in an LP matrix in LP preprocessing. As this algorithm only works on a columnwise representation of the cut pool, we need to perform a sparse transpose.

- As the solution to the LP relaxation may have changed since the last call to the cut selection algorithm, quality measures whose values depend on this solution need to be updated. In our implementation we just recompute the violation (see Equation (10.4)) and the different norms (for instance, see denominator in Equation (10.16)). All other quality measures can be calculated from these basic values.

- The cut quality values are computed and stored in the array `poolqul` and the index of each cut is stored in the array `poolsrt`. We sort `poolsrt` according to the quality values in `poolqul`. This strategy makes direct access to all cuts in order of their quality possible.

- Suppose l cuts have been added to the cut pool and the cut pool has been sorted by non-ascending quality values. Furthermore, given $\delta = 1, \dots, l-1$, let
$$\alpha^i x \leq \beta^i, \quad i = \text{poolsrt}[\delta] \tag{10.20}$$
be the cut of δ^{th} best quality and let
$$\alpha^k x \leq \beta^k, \quad k \in \{\text{poolsrt}[\delta+1], \dots, \text{poolsrt}[l]\}, \tag{10.21}$$
be the set of cuts with smaller quality values. To reduce the size of the cut pool and increase the diversity of cuts we perform a check for parallelism which exploits the order of the cut pool. Thus we require all pairs (i,k) of cuts as defined in Equations (10.20) and (10.21) to satisfy the inequality
$$p\left(\alpha^i, \alpha^k\right) \leq p\_max. \tag{10.22}$$
We thereby remove all cutting planes with index k which are (almost) parallel to a better cut with index i from the cut pool. But in order to avoid removing high-quality cuts from the cut pool, we additionally check whether the quality of such a cut is relatively large. If the quality value is larger than *skip_factor* times *best_qual* and the value of $p(\alpha^i, \alpha^k)$ is not larger than *p_max_ub* cut k is not removed.

- If $\alpha^i = w\alpha^k$ with $w \in \mathbb{R}$, then the hyperplanes $\alpha^i x = \beta^i$ and $\alpha^k x = \beta^k$ are parallel. We obtain the equation
$$p\left(\alpha^i, c\right) = \frac{|\alpha^i c|}{\|\alpha^i\| \, \|c\|} = \frac{\left|w\alpha^k c\right|}{\|w\alpha^k\| \, \|c\|} = \frac{\left|w\alpha^k c\right|}{|w| \, \|\alpha^k\| \, \|c\|} = p\left(\alpha^k, c\right) \tag{10.23}$$
and the implication
$$p\left(\alpha^i, \alpha^k\right) = 1 \Longrightarrow p\left(\alpha^i, c\right) = p\left(\alpha^k, c\right). \tag{10.24}$$

Algorithm 10.1. Cut selection algorithm

```
Input: A fractional basic LP solution $x^*$. A number of $l$ cutting planes
       in the cut pool.
Output: A selected list $\mathcal{L}$ of cutting planes.
(Step 1) Initialize
         if first pass of cut selection routine then
             Set $fail := 0$ and $total\_cuts := 0$.
             Moreover, set $max\_cuts\_total := total\_factor \cdot m$ and
             $max\_cuts\_round := round\_factor \cdot m$.
         end
         if $l = 0$ then exit.
         Set $\mathcal{L} := \emptyset$.
(Step 2) Remove duplicates
         Perform a sparse transpose and remove duplicate cuts using the
         algorithm of Tomlin and Welch [164].
(Step 3) Update quality measures
         Calculate the values of the quality measures dependent on the LP
         solution $x^*$.
(Step 4) Sort cut pool according to quality
         Sort the array of cut indices poolsrt in non-ascending order of the quality
         values in the array poolqul.
(Step 5) Remove parallel cuts
         Check for pairs of cuts $(i, k)$ which violate $p(\alpha^i, \alpha^k) \leq p\_max$.
         Check also for dominated cuts.
(Step 6) Check cut age
         For all cuts in the cut pool, set $age_i := age_i + 1$ and remove cuts with
         $age_i \geq max\_age$.
(Step 7) Calculate required minimal quality
         Set $best\_qual := poolqul[poolsrt[0]]$.
         if first pass of cut selection routine then set
         $min\_qual := \min\{min\_qual\_ub, 0.5 \cdot best\_qual\}$.
         if $best\_qual < min\_qual$ then
             Set $fail := fail + 1$.
             if $fail = 2$ then set $fail := 0$ and $min\_qual := min\_qual - 0.05$.
             exit.
         end
(Step 8) Select cuts
         Set $k := 0$.
         while $poolqul[poolsrt[k]] \geq min\_qual$ and $k \leq l$ do
             if $total\_cuts \geq max\_cuts\_total$ then exit.
             Add cut $poolsrt[k]$ to the list $\mathcal{L}$.
             Set $total\_cuts := total\_cuts + 1$ and $k := k + 1$.
             if $k \geq max\_cuts\_round$ then exit.
         end
```

Consequently, being immediately parallel to the objective function is a required condition for the parallelism of two cutting planes. This detail can be used to save computational effort needed for the computation of the inner products between coefficient vectors.

- To remove dominated cuts we test whether the set of indices of a cut is completely contained in the set of indices of another cut in the cut pool. Then cut coefficients are inspected to decide whether a simple dominance relationship holds.

- Whenever a cutting plane is found to be parallel to, a duplicate of or dominated by another cut it is marked as *deleted* instead of being directly removed from the cut pool. All parts of the algorithm simply ignore cuts which are marked as deleted. If an additional cut cannot be added to the cut pool because the maximum number of cuts in the cut pool is reached, a compression routine is executed. This compression routine efficiently removes all cuts which are marked as deleted from the cut pool.

- We require cuts which are inserted into the LP relaxation to have a minimum quality. Following suggestions described in [13] we do not fix this minimum quality min_qual a priori. At the first call of the selection routine min_qual is set to the minimum of min_qual_ub and 50% of the quality of the best cut in the cut pool. Whenever the cut pool does not contain cuts having the required minimum quality, the counter $fail$ is incremented. If $fail$ takes the value two, meaning that the cut pool did not contain adequate cuts in two rounds, min_qual is reduced to allow for cuts with smaller quality values.

- The parameters $total\_factor$ and $round\_factor$ restrict the number of cuts added to the LP relaxation during one round of cut generation and the whole supernode processing respectively. As illustrated in Algorithm 10.1, our implementation does not allow for the total number of cuts (or number of cuts per round) to be larger than $total\_factor$ (or $round\_factor$) times m, which is the number of constraints in the initial LP formulation.

- Cutting planes are *aged out* of the cut pool, meaning that cuts whose quality values were not large enough to be added to the LP relaxation in max_age rounds are deleted.
- Non-binding constraints are removed from the LP relaxation. Additional techniques such as aging are not used in the LP context. In particular, this means that cuts are not transferred from the LP formulation back to the cut pool.

10.4. Computational Results

In the preceding sections we discussed various cut quality measures and presented a cut selection algorithm. In this section we evaluate computationally the effectiveness of these cut quality measures. We further assess the effect that our cut selection algorithm has on the overall performance of Mops.

Apart from choosing a reliable cut quality measure, a number of parameters need to be set for our cut selection algorithm. In a series of preliminary experiments, we tested different default settings for these parameters while keeping the cut quality measure fixed. We first only limited the number of cutting planes added to the LP relaxation by setting the parameters $total\_factor$ and $round\_factor$ to the values 1.0 and 0.1 respectively. The results obtained with this parameter setting were not satisfying. After some tuning we found the following default setting which we used to compute all results presented in this thesis. In particular, we pursue a quite aggressive strategy to reduce the number of cutting planes in the cut pool. Duplicate cuts are identified and even fairly parallel cutting planes ($p\_max = 0.1$) are not considered. We avoid removing cuts with a relatively high quality ($skip\_factor = 0.9$, $p\_max\_ub = 0.5$). In addition, we try to reduce the size of the cut pool by restricting the maximum age of a cut to three ($max\_age = 3$). The minimum quality value we require a cut to have is 0.01 ($min\_qual\_ub = 0.01$). It turned out to be unnecessary further to limit the number of cuts added to the LP relaxation ($total\_factor = \infty$, $round\_factor = \infty$). Note that we do not examine the effect of each single parameter on the overall performance. These settings are therefore still first choices likely to be improved by further experiments.

	test set	violation	rel. violation	distance	obj. parallelism	exp. improvement	support	int. support	distance var. I	distance var. II
time	ACC	**-39**	**-34**	-3	**-38**	**-44**	**-64**	**-27**	**-44**	**-26**
	CORAL	**-30**	**-25**	**-36**	**-20**	**-31**	**-45**	**-38**	**-42**	**-30**
	MIPLIB	**-17**	**-17**	**-38**	**+8**	**-34**	**-33**	**-26**	**-24**	**-14**
	MILP	**-31**	**-37**	**-52**	**-44**	**-47**	**-31**	**-43**	**-52**	**-46**
	total	**-27**	**-25**	**-37**	**-18**	**-34**	**-40**	**-35**	**-39**	**-28**
nodes	ACC	**+41**	**+58**	**+87**	**+48**	**+39**	**-19**	**+72**	**+23**	**+73**
	CORAL	**+22**	**+22**	-2	**+61**	**+21**	**-13**	+1	**-13**	**+16**
	MIPLIB	**+26**	**+25**	**-14**	**+197**	**-12**	**-19**	-3	**+8**	**+66**
	MILP	**+23**	**-6**	**-15**	**+18**	-2	**+16**	-2	**-18**	**+10**
	total	**+23**	**+20**	**-5**	**+82**	**+9**	**-11**	+1	**-8**	**+28**
gap	ACC	0	0	0	0	0	0	0	0	0
	CORAL	-2	0	**-7**	**+18**	**+6**	**-5**	0	**-6**	+2
	MIPLIB	0	**-8**	0	**+25**	0	-3	**-10**	**-8**	**+7**
	MILP	+3	**-8**	**+5**	**+12**	+1	**-7**	**-8**	**-5**	**+7**
	total	-1	-3	-3	**+19**	+4	-4	-3	**-6**	+4

Table 10.1. Performance impact of using particular cut quality measures. The values represent percentage changes in the shifted geometric mean compared with a reference setting in which no cut selection scheme is activated. Positive values indicate a deterioration while negative values signify an improvement.

As in Chapters 8 and 9, we perform two experiments. In our first experiment, we investigate the individual computational effectiveness of the cut quality measures we presented in Section 10.2. We compare the variants using a specific quality measure with a reference setting which does not apply a cut selection algorithm. The results of this experiment are summarized in Table 10.1. Tables B.51 to B.55 in Appendix B present the detailed results. An immediate observation from Table 10.1 is that, independent of the cut quality measure used, the performance of an LP-based cut-and-branch MIP solver like MOPS can considerably be improved by carefully selecting only a subset of the cutting planes generated during IP preprocessing. Concerning the time spent on solving the instances in our test sets to optimality, our cut selection algorithm almost exclusively leads to performance improvements in terms of the relative shifted geometric mean values. In fact the only increase in the solution times that Table 10.1 reports on arises from using the objective parallelism ("obj. parallelism") to measure cut quality for the MIPLIB instances. Measuring cut quality by the Euclidean distance between the cut hyperplane and the fractional LP solution ("distance") yields the best results on the MIPLIB and MILP instances. For the ACC and CORAL instances the largest decreases in the running times are obtained by evaluating cutting planes according to their density ("support"). The effect of our cut selection algorithm on the enumeration during the branch-and-bound algorithm varies across the test sets and quality measures. The reference setting which adds all cutting planes generated during IP preprocessing to the LP relaxation in many cases computes the smallest number of nodes during the branch-and-bound search. Increases in the node counts are not surprising since our cut selection algorithm strongly reduces the number of cutting planes added to the LP relaxation, thereby affecting its tightness. The search space the branch-and-bound algorithm needs to explore thus increases if the MIP formulation produced by our cut selection algorithm has a lower quality. The number of branching nodes, for example, increases by 197% for the MIPLIB instances if cut quality is measured by the objective parallelism. This increase corresponds to the only increase in the running times we mentioned above. The remaining, sometimes dramatic, increases in the number of branching nodes do not, however, lead to increased running times. The greater part of the branch-and-bound algorithm's running time is typically spent on solving LP relaxations. Cutting planes increase the size of these LP relaxations, making them more difficult to solve. Our computational experiments indicate that the number of constraints in the LP relaxation is considerably increased by adding all cutting planes generated during IP preprocessing, which repeatedly leads to higher node solution times during the branch-and-bound process. If, on the other hand, only a careful subset of the generated cuts is added, solving the LP relaxation becomes computationally less expensive. Since the LP relaxation is solved for every branching node, the performance gain in LP optimization achieved by adding only selected cuts is likely to overcompensate the loss in the quality of the MIP formulation. This is also shown by the results in Table 10.1 where the overall solution times decrease even if the number of branching nodes increases. Concerning the amounts of integrality gap closed at the root node, our

	test set	no cut selection	no int. support	no obj. parallelism	only distance
time	ACC	**-21**	0	**-24**	**-24**
	CORAL	**+90**	**+28**	**+8**	**+29**
	MIPLIB	**+40**	**-8**	-4	**-15**
	MILP	**+125**	-2	**+15**	**+6**
	total	**+73**	**+13**	**+5**	**+12**
nodes	ACC	**-66**	0	**-36**	**-36**
	CORAL	**+39**	**+49**	**+19**	**+48**
	MIPLIB	**+18**	-2	**-9**	+1
	MILP	**+25**	+2	**+17**	**+9**
	total	**+25**	**+24**	**+7**	**+24**
gap	ACC	0	0	0	0
	CORAL	**+7**	+2	0	+1
	MIPLIB	**+15**	+1	0	**+15**
	MILP	+3	+2	+1	**+8**
	total	**+9**	+2	0	**+6**

Table 10.2. Performance impact of changing the default cut quality measure. The values represent percentage changes in the shifted geometric mean compared with the default setting which uses a weighted sum of the distance, objective function parallelism and integral support to measure cut quality. (The first quantity is weighted with 1.0 and the latter two with 0.1.) Positive values indicate a deterioration while negative values signify an improvement.

results show that most cut quality measures produce dual bounds of good quality. Considerable losses in the quality of the dual bound can only be observed if cut quality is measured by the objective parallelism. On the other hand, some quality measures close more integrality gap than the reference setting although fewer cuts are added. For the MIPLIB instances the first distance variant ("distance var. I") increases the shifted geometric mean of the amounts of integrality gap closed at the root node by 10%.

In its default settings Mops uses a combined measure for the quality of cutting planes. This measure is given by a weighted sum of the Euclidean distance, objective function parallelism and integral support, where the first quantity is weighted with 1.0 and the latter two with 0.1. In our second experiment, we assess the effectiveness of this combined quality measure. Table 10.2 presents a summary of the results of this experiment. The detailed results can be found in Tables B.56 to B.60 in Appendix B. For example, the column headed "no obj. parallelism" in

Table 10.2 shows the development of the performance if the criterion objective parallelism is disabled, i.e. if the corresponding weight in the weighted sum is changed to zero. Table 10.2 indicates that, in comparison with the naive policy which adds all generated cuts (“no cut selection”), the combined quality measure performs very well. Concerning the total test set (“total”) the solution times and the number of branching nodes increase by 73% and 25% respectively if no cut selection is applied. In addition, the amount of integrality gap closed at the root node decreases by 9%. Column “only distance” reveals that, in comparison with the quality measure distance (see Equation (10.6)), the combined quality measure also performs quite well although it is outperformed on the ACC and MIPLIB instances with respect to solution times. For the CORAL instances disabling the integral support (“no int. support”) leads to large performance deteriorations. The effect of deactivating the objective parallelism (“no obj. parallelism”) is not so large, but nevertheless the shifted geometric mean of the running times and number of branching nodes clearly increases for the MILP instances.

Chapter 11.

Summary and Concluding Remarks

In this thesis we have investigated the generation of general-purpose cutting planes from single- and multiple-constraint relaxations of mixed-integer programs, with an emphasis on computational aspects. Cutting planes play a crucial role in accelerating the solution process of MIPs and are responsible for significant improvements in the performance of MIP solvers (cf. Bixby et al. [38]). Developing efficient implementations of cutting plane techniques can yet be a challenging task. For example, although proposed already in the late 1950s and early 1960s, Gomory cuts [97–99] were first successfully implemented in a branch-and-cut framework in the middle of the 1990s by Balas et al. [26].

Part I of this thesis offered an introduction to mixed-integer programming. In Chapter 2 we showed how valid inequalities for MIPs can be derived from disjunctive arguments and how the LP relaxation of MIPs can be strengthened by using these valid inequalities as cutting planes. Chapter 3 discussed basic solution methods for MIPs such as cutting plane and branch-and-bound algorithms.

In Part II of this thesis we reviewed the state-of-the-art in cutting plane technology. Chapter 4 considered split cuts, i.e. cutting planes generated from single-row relaxations of MIPs. In practice the split closure provides a tight approximation of the convex hull of the feasible solutions of MIPs (see Balas and Saxena [31] and Dash et al. [73]). On the other hand, it is also known that optimizing over the split closure is $\mathcal{NP}$-hard (see Caprara and Letchford [44]). We therefore discussed several families of split cuts which can be generated efficiently such as Gomory mixed-integer cuts, reduce-and-split cuts [8] and lift-and-project cuts [30]. These families of split cuts do not, however, produce an approximation of the integer hull of MIPs which is as tight as that provided by the elementary split closure (cf. Cornuéjols and Nannicini [64]). Nor is there any computational study which compares the individual effectiveness of these families of split cuts in solving MIPs. In Chapter 5 we examined the derivation of cutting planes from multi-row relaxations. We reviewed multi-row and group relaxations and elaborated on the connection between valid inequalities for the semi-infinite relaxation and maximal lattice-free convex sets. We also demonstrated how intersection cuts can be derived from maximal lattice-free convex sets other than split sets and how these intersection cuts can be strengthened by using the integrality of some of the non-basic variables. While several authors study strengthened two-row cuts (see Basu et al. [32] and Dey et al. [75]) and non-strengthened multi-row cuts (see Espinoza [83, 84]) computationally, there is no extensive computational study of multi-row and strengthened (or lifted) multi-row cuts on a large-scale test set. In general, only very few publications are concerned with implementation aspects of cut generation and document the technical details making cut separators effective in practice.

From our review of the state-of-the-art we presented the three main research objectives of this thesis in Chapter 6. The first objective addressed computational aspects of the generation of split cuts. We proposed development of a novel heuristic approach to strengthen the Gomory mixed-integer cuts or, in other words, a new algorithm which generates a family of split cuts. We also wanted to develop efficient implementations of our algorithm and existing algorithms from the literature for generating split cuts. We intended to document important technical details and, moreover, to compare the discussed families of split cuts based on computational experiments. The second objective concerned the generation of cutting planes from multiple rows of a simplex tableau, again with a focus on computational aspects. We wanted to implement multi-row cut separators and to highlight important implementation details. We wanted, moreover, to perform an extensive computational study of different configurations of these multi-row cut separators. In particular, we aimed to assess whether the generation of multi-row cuts provides a performance improvement over split cuts. We also wanted to analyze the benefits of strengthening multi-row cuts using the integrality requirements on some of the non-basic variables. The third objective of this thesis was to study techniques for cut selection and management. We wanted to develop a cut selection algorithm and to perform computational experiments with different cut quality measures.

Part III of this thesis described our implementation of different cut separators and presented computational results. In Chapter 7 we introduced the MIP solver Mops and considered some general aspects of our implementation. We discussed the main data structures used in our code and the internal and external representation of MIP models in Mops. We also addressed the inherent inaccuracy of floating-point arithmetic.

With respect to the first research objective stated in Chapter 6 we studied in Chapter 8 approaches to generate split cuts. We presented a novel algorithm for improving the performance of the Gomory mixed-integer cuts. This algorithm is based on the work of Andersen et al. [8] who observed the distance cut off by an intersection cut to be affected by the size of the coefficients of the continuous variables in the rows of the simplex tableau. To increase the distance cut off, Andersen et al. developed an algorithm which reduces the size of these coefficients by taking integer linear combinations of the rows of the simplex tableau. The idea behind our approach is very similar. However, instead of considering linear combinations of tableau rows, we proposed a reduction algorithm which performs a sequence of pivots on the simplex tableau. Our algorithm thus modifies the basis from which an intersection cut is generated while Andersen et al. change the disjunction. Given a reference row of the simplex tableau, we first select a pivot row distinct from the reference row which is likely to contain an improving pivot. Once a pivot row has been identified, we choose the pivot column which produces the largest reduction in the coefficients of the continuous variables in the reference row. We called the resulting cuts *pivot-and-reduce* cuts. According to the research objectives discussed in Chapter 6 we implemented cut separators for the new family of pivot-and-reduce cuts as well as for several other families of split cuts from the literature such as $\{0, \frac{1}{2}\}$-cuts, strong CG cuts, Gomory mixed-integer cuts, k-cuts, combined Gomory mixed-integer cuts, reduce-and-split cuts and lift-and-project cuts. We described these cut separators in detail and also highlighted important technical details. For instance, we described how to apply the reduction algorithm used to obtain reduce-and-split cuts on a sparse matrix data structure. Concerning the separation of lift-and-project cuts using the Balas-Perregaard procedure, we addressed the efficient computation of the reduced cost and other implementation techniques. Moreover, we discussed a cost-efficient way of identifying promising pivot rows in the pivot-and-reduce algorithm. As proposed in Chapter 6 we conducted extensive computational experiments with the discussed cut separators. We showed that, in comparison with the Gomory mixed-integer cut separator, the reduce-and-split, pivot-and-reduce and lift-and-project cut separators are very effective in reducing the number of branching nodes computed and in increasing the amount of integrality gap closed on a large part of our test set. Our experiments, on the other hand, also revealed that these elaborate cut separators are often not competitive with the Gomory mixed-integer cut separator in terms of solution times due to their computational expensiveness. We pointed out, however, that the improved cut separators are superior in respect of solution times on instances which are hard to solve using only the Gomory mixed-integer cut separator. We demonstrated that the performance of our lift-and-project cut separator can be improved by applying the disjunctive modularization or the Euclidean normalization. Our experiments also showed that the Chvátal-Gomory cut separators are a useful addition to the discussed tableau cut separators. Especially the $\{0, \frac{1}{2}\}$-cut separator was very effective on pure integer instances.

In Chapter 9 we discussed the generation of multi-row cutting planes. According to the research objectives presented in Chapter 6 we implemented several cut separators which derive cutting planes from multiple rows of a simplex tableau and described in detail our implementation. We defined in particular the families of lattice-free convex sets we use for cut generation and discussed how we construct and handle the multi-row relaxations. We also dealt with implementation details such as cut strengthening using the integrality of the non-basic variables and fast iteration over the elements of a family of lattice-free convex sets. In compliance with the research objectives proposed in Chapter 6 we also performed a detailed computational study of the multi-row cut separators on a large-scale test set. We showed that, compared with the split cut separators discussed in Chapter 8, two-row cuts and strengthened (or lifted) two-row cuts successfully reduce the number of branching nodes on our test set. The two-row cut separators were, on the other hand, often not competitive with the split cut separators as far as solution times are concerned. We also studied the effect of generating cutting planes from more than two rows of a simplex tableau. These multi-row cut separators improved upon the results of the two-row cut separators in terms of the reduction in the number of branching nodes. Due to our separation strategy to compute an intersection cut for each maximal lattice-free convex set in a family and to select the one with the largest distance cut off, the computational work carried out to compute multi-row cuts increases considerably with the number of tableau rows in the multi-row relaxation. Therefore enabling these multi-row cut separators mainly resulted in deteriorations of the solution times. We also showed, however, that these multi-row cut separators reduce the solution times on instances which are hard to solve for Mops if only the split cut separators are applied. In our experiments the strengthening of cuts derived from more than two rows of a simplex tableau had a positive effect on the number of branching nodes computed. As implemented in our code, the strengthening is, however, very expensive and led to large increases in the solution times.

Chapter 10 examined cutting plane selection and management. As stated as a research objective in Chapter 6, we designed and implemented a cut selection algorithm and documented the technical details. We also presented various cut quality measures and carried out computational experiments with them. The results of these experiments showed that our cut selection algorithm has a positive effect on the performance of Mops. In particular, the selection of cutting planes based on the distance cut off performed very well. We also demonstrated that a combination of quality measures can be effective.

The field of computational mixed-integer programming provides many opportunities for further research such as the efficient design and implementation of algorithms and the documentation of key technical details. Future research in the area of cutting plane techniques could address several interesting theoretical and computational issues. One research opportunity is to find further families of split cuts that can be generated efficiently and to evaluate these families computationally. Other fruitful research might lie in strengthening cutting planes derived from multiple rows of a simplex tableau using the non-negativity constraints on the basic integer variables. Future research could also be concerned with accelerating the separation process of multi-row cuts, thereby making multi-row cuts more competitive with split cuts. For instance, closed-form formulae for the trivial strengthening of multi-row cuts could be derived. Research could also be devoted to developing a deeper understanding of the interaction of cutting planes.

Appendices

Appendix A.

Benchmarking Environment

This chapter of the appendix presents details of the environment we use computationally to evaluate the algorithms presented in this thesis. In Section A.1 we discuss the hard- and software environment in which our benchmarks have been conducted. Section A.2 deals with the selection of a test set of MIP instances. Finally, Section A.3 details the methods we use to evaluate the outcome of our experiments.

A.1. Hard- and Software

Our implementations of the algorithms discussed in this thesis are written in Fortran 95. The source code was compiled by the Intel Visual Fortran Compiler 11.1.051 on a standard PC running Microsoft Windows 7 as operating system. The compiled code was linked to the Mops (static link) library. For our benchmark runs we employed Mops version 10.7.4. All of our computational experiments were conducted on Dell Optiplex 755 standard PCs with 8 GB of main memory and a 3.16 GhZ Intel Core2Duo E8500 CPU with 6 MB of second-level cache.

For our benchmarks we imposed a maximum computation time of one hour per instance. If an instance was not solved within this time limit, we assume a computation time of one hour and treat the current number of computed branch-and-bound nodes as the final node count in our evaluations. We did, however, not impose a memory limit. Concerning the branch-and-bound algorithm, we limit the amount of disk space used to store the node file during the branch-and-bound algorithm to 2 GB. If a test run exceeded the disk space limitations, we assume that it hit the time limit and scale the node count accordingly.

A.2. Test Set

It is common practice in computational mixed-integer programming to perform numerical experiments so as to assess the impact of specific algorithms on the overall performance of an MIP solver. We benchmark our code on instances from several publicly available test sets. The initial collection consisted of the instances from the following sources:

- ACC [141]
  `http://www.ps.uni-sb.de/~walser/acc/acc.html`
- CORAL [131]
  `http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/`
- MIPLIB [6, 37]
  `http://miplib.zib.de`
- MILP [138]
  `http://plato.asu.edu/ftp/milp/`

The ACC test set only contains instances of pure 0-1 (binary) programs originating from baseball scheduling. All other test sets comprise general MIP instances from various problem classes and applications. To obtain a manageable set of instances for our computational experiments we applied the following rules to these test sets.

1. Our test set MIPLIB consists of the instances from MIPLIB 3.0 and MIPLIB 2003. Some of the instances contained in MIPLIB 3.0 are also part of MIPLIB 2003. We removed these duplicates from MIPLIB 2003.
2. Some of the instances in the CORAL test set are duplicates of instances in MIPLIB and MILP. We removed these duplicates from the CORAL test set.

3. Finally, we removed all instances from the test sets that could not be solved by a preliminary version of MOPS 10.7.4 (which included our new algorithms) within one hour of computation time.

Our final test set consists of 198 instances: 7 instances from the ACC test set, 110 instances from the CORAL test set, 55 instances from the MIPLIB test set and 26 instances from the MILP test set.

A.3. Evaluation Methods

When benchmarking the performance of algorithms related to MIP solving on a large-scale test set, one is faced with the issue of analyzing the large amount of result data generated. Due to the size of our test set, reporting results for every single instance is impractical and would reduce the readability of our analysis. Thus we decided to sum up the results of our benchmark runs by means of average values. The detailed results are, however, available from the author upon request.

Given a set of non-negative values $u_1, \dots, u_n \geq 0$ we consider the arithmetic mean

$$\frac{1}{n}\sum_{i=1}^{n} u_i, \tag{A.1}$$

the geometric mean

$$\left(\prod_{i=1}^{n} \max\{u_i, l\}\right)^{\frac{1}{n}} \tag{A.2}$$

and, following Achterberg [3], the shifted geometric mean

$$\left(\prod_{i=1}^{n} \max\{u_i + s, l\}\right)^{\frac{1}{n}} - s, \tag{A.3}$$

where $s, l \geq 0$ are non-negative parameters. We compute mean values for the computation times, the number of branching nodes and the amounts of integrality gap closed at the root node. The amount of integrality gap closed is given by $(\underline{c} - c^*)/(\bar{c} - c^*)$ where $\underline{c}$ is the optimal objective value of the LP relaxation after adding cutting planes. For instances which do not have an integrality gap we assume a value of 1 (i.e. 100% gap closed) in the calculation of the mean values. The parameter l constitutes a lower bound on the values $u_1, \dots, u_n$ and allows for computing the (shifted) geometric mean if there are zero values. When computing the (shifted) geometric mean, we set $l = 1$ for node counts and running times and $l = 0.01$ (i.e. 1%) for the amounts of integrality gap closed. As proposed by Achterberg [3] we set the shifting parameter s to $s = 10$ for the running times and $s = 100$ for the node counts. We further set $s = 0.05$ (i.e. 5%) if amounts of integrality gap closed are considered.

Depending on the data given each of the above three means may take values which lead to different conclusions. Suppose two configurations A and B with two series of observations $a_1, \ldots, a_n \geq 0$ and $b_1, \ldots, b_n \geq 0$ are given, for instance solution times on a particular test set. Suppose moreover that each series contains outliers which are very small in the scale of the remaining values, i.e. instances which are very easy to solve compared with the remaining instances. We should now like to assess how configuration A performed in comparison with B. Following Achterberg [3] we shall compute ratios between the arithmetic, geometric and shifted geometric means. In our example the ratio between the arithmetic means reads $\sum_{i=1}^{n} a_i / \sum_{i=1}^{n} b_i$ and the ratio between the geometric means is given by $\prod_{i=1}^{n}(a_i/b_i)^{1/n}$. The running times on the easy instances are thus more or less ignored by the relative arithmetic mean. By contrast, a slight variation of the running times on an easy instance i greatly affects the ratio a_i/b_i. Consequently, the relative geometric mean is greatly influenced by the easy instances. In order to control the influence of easy instances on the geometric mean, the shifted geometric mean is considered which is parameterized by the shifting parameter s. The larger the parameter s chosen, the more the influence of the easy instances is reduced.

Dolan and Moré [81] propose a different approach to obtaining a comprehensive view of the performance of algorithms. Suppose that a set T of algorithms and a set S of test instances are given. Let $q_{t,s}$ be a performance measure which quantifies how algorithm $t \in T$ performs on instance $s \in S$ (e.g. running times). Also let $t_s^* \in T$ be the algorithm that performs best on instance $s \in S$. By computing the ratios $q_{t,s}/q_{t_s^*,s}$ for all $t \in T$ and $s \in S$ we see how an algorithm performed on an instance in comparison with the best algorithm. When plotting these ratios in an accumulated way we obtain the so-called performance profiles of the algorithms.

Appendix B.

Tables

This chapter of the appendix presents the detailed results of our computational experiments in tabular form. Each table shows the performance of various parameter configurations on a specific set of test instances. A detailed discussion of the collections of test instances and the evaluation methods we use can be found in Appendix A.

For each tested parameter configuration we report the arithmetic mean, geometric mean and shifted geometric mean of the solution times, the number of branching nodes and the amounts of integrality gap closed at the root node. As mentioned in Section A.3 we do not, however, present these mean values in absolute numbers, but compute the relative percentage change as compared with a reference configuration. For example, given a set of test instances, let g_R and g_T denote the geometric means of the solution times or the number of branching nodes taken by the reference setting R and an alternative parameter setting T. The relative percentage change in the geometric mean is then calculated as $100 \cdot (g_T/g_R - 1)$. A negative value thus corresponds to a performance improvement whereas a positive value indicates a performance deterioration compared with the reference setting. Concerning the amounts of integrality gap closed at the root node, the relative percentage change in the geometric mean is, for the sake of consistency, given by $100 \cdot (1 - g_T/g_R)$. The relative percentage changes in the arithmetic and shifted geometric means are obtained by analogous calculations. Information about the tested parameter configurations can be found in the column "setting". The respective mean value results can be found in the columns "total"[1],

[1]Given two series of observations, the ratio between the arithmetic mean values is equal to the ratio between the sums of the observed values. We therefore denote by "total" the columns containing the relative percentage change in the arithmetic means.

"geom. mean" and "shifted geom. mean". The changes in the solution times, node counts and gap values are shown in the sub-columns "time", "nodes" and "gap" respectively. Values printed in bold face indicate that the improvement or deterioration is larger than five percent. The reference setting is shown in the first row of each table. For example, the results in Table B.1 are given as relative percentage changes to the performance of the Gomory mixed-integer cut separator (see Chapter 8).

Besides these mean values, we provide additional information about the performance of the configurations tested. The column headed "T" indicates the number of instances in the test set which can not be solved to optimality within the time limit by the respective configuration. In the column "F" we further report the number of instances on which a configuration failed. The column "wins / losses" presents the number of instances on which a configuration performs at least 10% better or worse than the reference setting in terms of solution times, branching nodes or gap values.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + *k*-cuts	2	0	0 / 1	0 / 1	0 / 0	**+108**	**+126**	0	**+101**	**+124**	0	**+51**	**+303**	0
GMI + cGMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + R&S	1	0	0 / 0	0 / 0	0 / 0	+1	0	0	0	0	0	0	0	0
GMI + P&R	1	0	0 / 2	0 / 2	0 / 0	**+15**	**+20**	0	**+12**	**+18**	0	0	+2	0
L&P	1	0	3 / 0	2 / 1	0 / 0	**-9**	**-12**	0	**-8**	**-11**	0	**-8**	-4	0

Table B.1. Performance impact of enabling particular single-row tableau cut separators (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	6	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + *k*-cuts	7	0	8 / 7	11 / 6	1 / 3	-2	-4	0	-1	-3	-1	+2	-4	-1
GMI + cGMI	6	0	7 / 16	9 / 13	3 / 1	+1	-2	-2	+1	-1	-1	+3	+2	0
GMI + R&S	8	0	12 / 16	12 / 11	4 / 4	**-6**	**-24**	-2	-2	**-17**	-3	**+7**	**-12**	-3
GMI + P&R	8	0	29 / 35	37 / 27	13 / 3	+3	**-7**	**-13**	0	**-6**	**-10**	-2	**-7**	**-5**
L&P	8	1	32 / 43	47 / 33	21 / 6	+3	-4	**-15**	0	**-7**	**-12**	+2	-2	**-5**

Table B.2. Performance impact of enabling particular single-row tableau cut separators (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	1	0	3 / 8	4 / 7	1 / 4	+2	**+10**	+2	+2	**+7**	+2	**+14**	**+34**	+2
GMI + cGMI	2	0	2 / 9	5 / 6	0 / 4	+3	-2	+3	+1	-2	+3	**+29**	**+22**	+3
GMI + R&S	1	1	5 / 16	12 / 13	4 / 1	-2	**-10**	-4	**-10**	**-11**	-4	-4	**-16**	-3
GMI + P&R	1	0	13 / 19	20 / 11	14 / 3	**-11**	**-34**	**-8**	**-19**	**-30**	**-9**	**-6**	**-26**	**-10**
L&P	1	0	11 / 26	19 / 16	21 / 5	+4	**-28**	**-21**	**-6**	**-24**	**-18**	**+6**	**-12**	**-13**

Table B.3. Performance impact of enabling particular single-row tableau cut separators (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	3	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	3	0	0 / 2	0 / 2	0 / 1	+2	+3	+3	+2	+3	+2	+1	+1	+1
GMI + cGMI	3	0	4 / 2	3 / 2	1 / 5	+1	**+6**	**+11**	+2	**+6**	**+9**	+1	-1	+4
GMI + R&S	2	0	2 / 4	4 / 2	4 / 2	**+8**	**+7**	-3	**+7**	**+7**	-3	-1	-4	-1
GMI + P&R	2	0	8 / 11	10 / 8	7 / 1	-4	**-9**	**-8**	**-7**	**-10**	**-8**	**-15**	**-39**	**-7**
L&P	2	0	9 / 12	9 / 9	6 / 5	**+14**	**+7**	**+13**	**+11**	**+7**	**+5**	**-16**	**-29**	-4

Table B.4. Performance impact of enabling particular single-row tableau cut separators (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	11	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	13	0	11 / 18	15 / 16	2 / 8	+3	+4	+1	+3	+4	0	**+6**	**+6**	0
GMI + cGMI	12	0	13 / 27	17 / 21	4 / 10	+1	-1	+1	+1	0	+1	**+5**	**+6**	+1
GMI + R&S	12	1	19 / 36	28 / 26	12 / 7	-3	**-16**	-3	-2	**-12**	-3	+4	**-11**	-2
GMI + P&R	12	0	50 / 67	67 / 48	34 / 7	-2	**-15**	**-10**	**-5**	**-13**	**-9**	**-5**	**-20**	**-6**
L&P	12	1	55 / 81	77 / 59	48 / 16	+4	**-11**	**-12**	0	**-11**	**-11**	-2	**-12**	**-7**

Table B.5. Performance impact of enabling particular single-row tableau cut separators (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + *k*-cuts	1	0	0 / 1	0 / 1	0 / 0	**+451**	**+574**	0	**+530**	**+618**	0	**+4291**	**+6587**	0
GMI + cGMI	0	0	0 / 0	0 / 0	0 / 0	+1	0	0	+1	0	0	+1	0	0
GMI + R&S	0	0	0 / 0	0 / 0	0 / 0	+1	0	0	+1	0	0	+1	0	0
GMI + P&R	0	0	0 / 2	0 / 2	0 / 0	**+38**	**+52**	0	**+38**	**+52**	0	**+39**	**+52**	0
L&P	0	0	2 / 0	1 / 1	0 / 0	**-12**	**-23**	0	**-13**	**-23**	0	**-15**	**-25**	0

Table B.6. Performance impact of enabling particular single-row tableau cut separators (easy ACC instances). We consider an instance to be easy if it can be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 3 instances in the ACC test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + *k*-cuts	0	0	3 / 5	4 / 4	0 / 2	+3	-3	+3	**+6**	0	+2	**+15**	**+15**	0
GMI + cGMI	0	0	1 / 4	1 / 3	0 / 0	+3	+2	0	+4	+2	0	**+7**	0	0
GMI + R&S	1	0	4 / 8	5 / 3	2 / 1	**+15**	**-16**	-4	**+29**	**-7**	-4	**+556**	**+116**	-2
GMI + P&R	0	0	7 / 18	9 / 13	1 / 1	**+25**	**+5**	**-5**	**+32**	**+6**	**-5**	**+46**	**-14**	-4
L&P	0	0	8 / 21	16 / 12	4 / 2	**+30**	**+18**	-2	**+31**	**+8**	-2	**+50**	**+26**	0

Table B.7. Performance impact of enabling particular single-row tableau cut separators (easy CORAL instances). We consider an instance to be easy if it can be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 43 instances in the CORAL test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	0	0	2 / 6	3 / 5	1 / 2	+2	**+14**	0	+2	**+11**	0	**+30**	**+46**	0
GMI + cGMI	0	0	1 / 7	4 / 4	0 / 3	+2	-3	+2	-3	-3	+2	**-8**	-1	+1
GMI + R&S	0	1	2 / 12	9 / 9	3 / 0	**+18**	+2	**-5**	**+9**	+1	**-5**	+1	**-7**	-4
GMI + P&R	0	0	6 / 16	12 / 9	10 / 2	**+5**	**-32**	**-11**	**-15**	**-27**	**-12**	**-32**	**-48**	**-12**
L&P	0	0	3 / 20	10 / 13	11 / 4	**+37**	**-15**	+2	**+60**	**-8**	-2	**+642**	**+110**	**-6**

Table B.8. Performance impact of enabling particular single-row tableau cut separators (easy MIPLIB instances). We consider an instance to be easy if it can be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 38 instances in the MIPLIB test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	0	0	0 / 1	0 / 1	0 / 1	+1	**+6**	**+14**	+1	**+6**	**+17**	+3	+3	**+21**
GMI + cGMI	0	0	2 / 0	0 / 0	0 / 2	**-12**	+1	**+25**	**-12**	+1	**+29**	**-13**	+4	**+32**
GMI + R&S	0	0	0 / 2	0 / 1	1 / 0	**+52**	**+59**	**-6**	**+79**	**+82**	**-7**	**+187**	**+817**	**-8**
GMI + P&R	0	0	0 / 4	2 / 2	2 / 0	**+90**	**+53**	**-14**	**+113**	**+79**	**-13**	**+197**	**+934**	**-17**
L&P	0	0	0 / 3	0 / 3	1 / 3	**+194**	**+169**	**+46**	**+279**	**+222**	**+54**	**+1209**	**+5445**	**+56**

Table B.9. Performance impact of enabling particular single-row tableau cut separators (easy MILP instances). We consider an instance to be easy if it can be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 5 instances in the MILP test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	1	0	5 / 13	7 / 11	1 / 5	**+8**	**+11**	+2	**+17**	**+12**	+1	**+369**	**+50**	0
GMI + cGMI	0	0	4 / 11	5 / 7	0 / 5	+2	0	+2	+1	-1	+2	+2	-1	+1
GMI + R&S	1	1	6 / 22	14 / 13	6 / 1	**+17**	**-5**	-4	**+24**	0	-4	**+375**	**+44**	-3
GMI + P&R	0	0	13 / 40	23 / 26	13 / 3	**+20**	**-9**	**-8**	**+23**	**-6**	**-8**	**+40**	**-30**	**-7**
L&P	0	0	13 / 44	27 / 29	16 / 9	**+37**	**+6**	+3	**+46**	**+5**	+1	**+239**	**+104**	-2

Table B.10. Performance impact of enabling particular single-row tableau cut separators (easy instances in the entire test set). We consider an instance to be easy if it can be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 89 instances in the entire test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + cGMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + R&S	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + P&R	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
L&P	1	0	1 / 0	1 / 0	0 / 0	**-7**	-3	0	**-7**	-3	0	**-8**	-3	0

Table B.11. Performance impact of enabling particular single-row tableau cut separators (hard ACC instances). We consider an instance to be hard if it can not be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 4 instances in the ACC test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	6	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	7	0	5 / 2	7 / 2	1 / 1	**-5**	-4	-2	-4	-4	-2	+2	-4	-2
GMI + cGMI	6	0	6 / 12	8 / 10	3 / 1	-1	-4	-3	0	-3	-2	+3	+2	-1
GMI + R&S	7	0	8 / 8	7 / 8	2 / 3	**-17**	**-29**	-1	**-11**	**-23**	-2	+1	**-14**	-3
GMI + P&R	8	0	22 / 17	28 / 14	12 / 2	**-10**	**-14**	**-18**	**-9**	**-13**	**-14**	-2	**-7**	**-6**
L&P	8	1	24 / 22	31 / 21	17 / 4	**-11**	**-16**	**-25**	**-10**	**-16**	**-20**	+2	-3	**-9**

Table B.12. Performance impact of enabling particular single-row tableau cut separators (hard CORAL instances). We consider an instance to be hard if it can not be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 67 instances in the CORAL test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + *k*-cuts	1	0	1 / 2	1 / 2	0 / 2	+3	+1	**+6**	+3	+1	**+6**	**+13**	**+34**	**+9**
GMI + cGMI	2	0	1 / 2	1 / 2	0 / 1	+3	+1	+4	+4	+1	**+5**	**+30**	**+23**	**+9**
GMI + R&S	1	0	3 / 4	3 / 4	1 / 1	**-35**	**-31**	0	**-25**	**-31**	0	-4	**-17**	+1
GMI + P&R	1	0	7 / 3	8 / 2	4 / 1	**-39**	**-39**	-2	**-33**	**-39**	-3	**-5**	**-25**	-3
L&P	1	0	8 / 6	9 / 3	10 / 1	**-43**	**-51**	**-89**	**-37**	**-50**	**-68**	**-7**	**-17**	**-42**

Table B.13. Performance impact of enabling particular single-row tableau cut separators (hard MIPLIB instances). We consider an instance to be hard if it can not be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 17 instances in the MIPLIB test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	3	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + *k*-cuts	3	0	0 / 1	0 / 1	0 / 0	+3	+3	0	+3	+3	0	+1	+1	0
GMI + cGMI	3	0	2 / 2	3 / 2	1 / 3	+4	**+7**	**+7**	+4	**+7**	**+6**	+1	-1	+2
GMI + R&S	2	0	2 / 2	4 / 1	3 / 2	0	-3	-2	0	-3	-2	-1	-4	-1
GMI + P&R	2	0	8 / 7	8 / 6	5 / 1	**-19**	**-20**	**-6**	**-19**	**-21**	**-8**	**-16**	**-40**	**-7**
L&P	2	0	9 / 9	9 / 6	5 / 2	**-9**	**-15**	+2	**-9**	**-13**	-4	**-19**	**-31**	**-7**

Table B.14. Performance impact of enabling particular single-row tableau cut separators (hard MILP instances). We consider an instance to be hard if it can not be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 21 instances in the MILP test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
GMI	11	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
GMI + k-cuts	12	0	6 / 5	8 / 5	1 / 3	-2	-2	0	-2	-2	0	+2	**+5**	0
GMI + cGMI	12	0	9 / 16	12 / 14	4 / 5	+1	-1	+1	+1	0	+1	**+5**	**+6**	+1
GMI + R&S	11	0	13 / 14	14 / 13	6 / 6	**-17**	**-24**	-1	**-11**	**-20**	-2	0	**-12**	-2
GMI + P&R	12	0	37 / 27	44 / 22	21 / 4	**-17**	**-20**	**-12**	**-15**	**-19**	**-10**	**-6**	**-20**	**-5**
L&P	12	1	42 / 37	50 / 30	32 / 7	**-17**	**-22**	**-26**	**-14**	**-21**	**-22**	**-5**	**-13**	**-12**

Table B.15. Performance impact of enabling particular single-row tableau cut separators (hard instances in the entire test set). We consider an instance to be hard if it can not be solved to optimality using only the GMI cut separator in 60 seconds or less. There are 109 instances in the entire test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only GMI	1	0	3 / 2	2 / 2	0 / 0	-4	+3	0	-3	+3	0	**+8**	**+15**	0
no k-cuts	1	0	0 / 0	0 / 0	0 / 0	+1	0	0	+1	0	0	0	0	0
no cGMI	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no R&S	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no P&R	1	0	2 / 1	2 / 1	0 / 0	**-10**	**-10**	0	**-8**	**-8**	0	-1	**+11**	0
no L&P	1	0	2 / 2	1 / 3	0 / 0	+3	**+9**	0	+2	**+8**	0	**+8**	**+15**	0

Table B.16. Performance impact of disabling particular single-row tableau cut separators (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	9	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only GMI	6	0	45 / 35	27 / 48	1 / 27	-4	**+18**	**+22**	-2	**+15**	**+18**	**-10**	**+10**	**+9**
no k-cuts	9	0	18 / 8	18 / 10	6 / 5	**-14**	**-20**	-1	**-13**	**-17**	-1	**-9**	**-15**	-1
no cGMI	9	0	21 / 12	21 / 14	4 / 4	**-7**	**-8**	0	**-6**	**-6**	0	**-6**	-3	0
no R&S	7	0	23 / 11	15 / 16	3 / 8	**-11**	**-7**	+2	**-10**	**-7**	+2	**-14**	-3	0
no P&R	9	0	36 / 23	32 / 28	8 / 11	**-10**	**-20**	+1	**-6**	**-15**	+1	**-5**	**-9**	0
no L&P	11	0	37 / 28	39 / 33	3 / 21	**+8**	**+15**	**+14**	**+9**	**+14**	**+11**	**+5**	-1	**+5**

Table B.17. Performance impact of disabling particular single-row tableau cut separators (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only GMI	1	0	26 / 13	8 / 26	4 / 25	**-9**	**+138**	**+26**	**+19**	**+103**	**+25**	**+19**	**+62**	**+20**
no *k*-cuts	1	0	9 / 10	11 / 9	1 / 3	+4	**+11**	+2	**+6**	**+8**	+2	+2	**+11**	0
no cGMI	1	0	8 / 15	10 / 12	3 / 3	**+6**	**+6**	0	**+6**	**+10**	0	+1	**+5**	0
no R&S	1	0	21 / 9	17 / 14	2 / 5	**-18**	+4	+4	**-11**	**-5**	+4	-4	**-12**	+4
no P&R	1	0	12 / 18	6 / 23	4 / 10	**+12**	**+67**	+3	**+14**	**+49**	+4	**+8**	**+19**	**+5**
no L&P	2	0	19 / 10	6 / 27	2 / 13	**+13**	**+52**	**+13**	**+25**	**+47**	**+10**	**+68**	**+116**	**+7**

Table B.18. Performance impact of disabling particular single-row tableau cut separators (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only GMI	3	0	11 / 8	6 / 14	2 / 10	-4	**+16**	**+17**	-1	**+13**	**+18**	**+37**	**+91**	**+14**
no *k*-cuts	2	0	5 / 6	3 / 5	2 / 2	**+9**	**+7**	+3	**+9**	**+6**	**+5**	**+22**	**+8**	+4
no cGMI	1	0	4 / 7	3 / 6	2 / 4	**+6**	+3	+2	+4	+3	+3	**+27**	0	+3
no R&S	1	0	6 / 3	6 / 3	0 / 3	-2	-1	**+5**	-1	-1	**+5**	**+13**	**+13**	+3
no P&R	1	0	5 / 10	3 / 9	3 / 7	**+15**	**+23**	**+8**	**+16**	**+22**	**+9**	**+41**	**+99**	**+6**
no L&P	2	0	11 / 8	7 / 12	1 / 9	-1	**+15**	**+22**	0	**+11**	**+21**	**+14**	**+7**	**+12**

Table B.19. Performance impact of disabling particular single-row tableau cut separators (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	11	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only GMI	11	0	85 / 58	43 / 90	7 / 62	**-5**	**+42**	**+22**	+2	**+33**	**+19**	+1	**+35**	**+12**
no *k*-cuts	13	0	32 / 24	32 / 24	9 / 10	**-6**	**-8**	+1	**-5**	**-8**	+1	-3	**-6**	0
no cGMI	12	0	33 / 34	34 / 32	9 / 11	-2	-3	0	-2	-1	+1	0	-1	+1
no R&S	10	0	50 / 23	38 / 33	5 / 16	**-12**	-3	+3	**-8**	**-5**	+3	**-8**	-2	+2
no P&R	12	0	55 / 52	43 / 61	15 / 28	-2	+4	+3	+1	+4	+3	+4	**+17**	+2
no L&P	16	0	69 / 48	53 / 75	6 / 43	**+8**	**+24**	**+14**	**+10**	**+21**	**+12**	**+12**	**+22**	**+6**

Table B.20. Performance impact of disabling particular single-row tableau cut separators (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR tableau + strong CG	2	0	2 / 3	3 / 3	0 / 0	**+6**	**-8**	0	**+9**	**-7**	0	**+36**	-2	0
SR tableau + $\{0, \frac{1}{2}\}$	0	0	3 / 4	3 / 3	0 / 0	**-8**	**+8**	0	**-10**	**+11**	0	**-23**	**-9**	0
SR tableau + strong CG + $\{0, \frac{1}{2}\}$	2	0	3 / 2	4 / 2	0 / 0	**+44**	**+38**	0	**+50**	**+42**	0	**+78**	**+73**	0

Table B.21. Performance impact of enabling particular CG cut separators (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	9	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR tableau + strong CG	6	0	14 / 7	14 / 10	3 / 7	**-5**	-4	0	**-7**	**-7**	0	**-15**	**-15**	0
SR tableau + $\{0, \frac{1}{2}\}$	5	0	17 / 9	18 / 6	7 / 1	**-21**	**-26**	**-9**	**-19**	**-25**	**-7**	**-23**	**-15**	-4
SR tableau + strong CG + $\{0, \frac{1}{2}\}$	6	0	17 / 16	20 / 12	9 / 3	**-19**	**-23**	**-11**	**-19**	**-25**	**-9**	**-24**	**-22**	**-5**

Table B.22. Performance impact of enabling particular CG cut separators (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR tableau + strong CG	1	0	9 / 6	10 / 6	4 / 2	+1	-2	**-5**	**+6**	-1	-3	**+6**	-2	-1
SR tableau + $\{0, \frac{1}{2}\}$	1	0	5 / 11	7 / 5	7 / 3	**+11**	0	-4	**+11**	+1	-2	**+7**	0	-1
SR tableau + strong CG + $\{0, \frac{1}{2}\}$	1	0	10 / 10	15 / 8	7 / 1	+2	**-13**	**-7**	**+8**	**-5**	**-5**	**+8**	-2	-2

Table B.23. Performance impact of enabling particular CG cut separators (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR tableau + strong CG	1	0	2 / 6	2 / 4	2 / 1	+4	0	**+8**	+4	0	**+5**	**+19**	-1	+2
SR tableau + $\{0, \frac{1}{2}\}$	0	0	1 / 3	4 / 1	2 / 0	**+11**	**-8**	**-10**	**+10**	-2	**-7**	+1	0	-4
SR tableau + strong CG + $\{0, \frac{1}{2}\}$	1	0	2 / 8	6 / 5	4 / 0	**+23**	0	**-15**	**+20**	+1	**-11**	**+21**	-1	**-5**

Table B.24. Performance impact of enabling particular CG cut separators (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR tableau	11	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR tableau + strong CG	10	0	27 / 22	29 / 23	9 / 10	-2	-3	0	-2	**-5**	0	-4	**-10**	0
SR tableau + $\{0, \frac{1}{2}\}$	6	0	26 / 27	32 / 15	16 / 4	**-9**	**-16**	**-7**	**-9**	**-15**	**-5**	**-16**	**-10**	-3
SR tableau + strong CG + $\{0, \frac{1}{2}\}$	10	0	32 / 36	45 / 27	20 / 4	**-7**	**-16**	**-10**	**-6**	**-15**	**-8**	**-8**	**-14**	-4

Table B.25. Performance impact of enabling particular CG cut separators (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
L&P	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
L&P + disj. mod.	1	0	1 / 0	1 / 0	0 / 0	-2	**-5**	0	-1	**-5**	0	0	0	0
L&P + Eucl. norm.	1	0	1 / 2	2 / 1	0 / 0	**-18**	**-21**	0	**-20**	**-22**	0	**-22**	**-20**	0
L&P + disj. mod. + Eucl. norm	1	0	1 / 3	2 / 2	0 / 0	**+9**	**+10**	0	**+9**	**+9**	0	**+9**	+4	0

Table B.26. Performance impact of enabling disjunctive modularization and Euclidean normalization (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
L&P	8	1	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
L&P + disj. mod.	9	0	11 / 20	14 / 20	6 / 5	+4	+2	+1	+4	+2	+1	+3	**+5**	0
L&P + Eucl. norm.	9	0	29 / 37	29 / 39	4 / 6	+2	+3	-2	+2	+4	-1	+2	+2	0
L&P + disj. mod. + Eucl. norm	10	0	27 / 37	30 / 42	5 / 8	+3	+3	-1	+2	+3	0	**+8**	**+9**	+1

Table B.27. Performance impact of enabling disjunctive modularization and Euclidean normalization (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
L&P	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
L&P + disj. mod.	1	0	7 / 16	13 / 17	4 / 4	**+6**	**+10**	**-5**	+3	**+5**	-2	+4	+1	+1
L&P + Eucl. norm.	1	0	13 / 16	19 / 20	13 / 6	**-7**	**-15**	-2	**-10**	**-13**	-2	**-19**	**-17**	-3
L&P + disj. mod. + Eucl. norm	1	0	13 / 18	22 / 14	12 / 6	-3	**-9**	-2	**-8**	**-9**	-2	**-23**	**-24**	-3

Table B.28. Performance impact of enabling disjunctive modularization and Euclidean normalization (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
L&P	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
L&P + disj. mod.	2	0	7 / 3	5 / 3	0 / 5	**-6**	-4	+4	**-5**	**-5**	+3	-1	+1	+2
L&P + Eucl. norm.	1	0	8 / 8	9 / 9	4 / 3	**-11**	**-17**	**-17**	**-11**	**-13**	**-9**	-1	**+9**	-1
L&P + disj. mod. + Eucl. norm	3	1	8 / 10	10 / 10	4 / 2	+2	**-5**	**-18**	+2	**-6**	**-12**	**+32**	**+49**	-4

Table B.29. Performance impact of enabling disjunctive modularization and Euclidean normalization (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
L&P	12	1	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
L&P + disj. mod.	13	0	26 / 39	33 / 40	10 / 14	+3	+3	0	+2	+2	0	+2	+4	0
L&P + Eucl. norm.	12	0	51 / 63	59 / 69	21 / 15	-3	**-6**	-4	-4	**-5**	-2	-2	-1	-1
L&P + disj. mod. + Eucl. norm	15	1	49 / 68	64 / 68	21 / 16	+1	-1	-3	0	-1	-2	**+10**	**+10**	-1

Table B.30. Performance impact of enabling disjunctive modularization and Euclidean normalization (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	2	0	0 / 2	0 / 2	0 / 0	**+12**	**+24**	0	**+9**	**+22**	0	0	+2	0
SR + $T2_2$	2	0	0 / 2	0 / 2	0 / 0	**+20**	**+25**	0	**+16**	**+23**	0	+1	+3	0
SR + $T3_2$	2	0	0 / 2	0 / 2	0 / 0	**+22**	**+34**	0	**+18**	**+31**	0	+1	**+5**	0
SR + G_2	2	0	0 / 2	0 / 2	0 / 0	**+16**	**+24**	0	**+13**	**+21**	0	0	+2	0
SR + SP_2	2	0	0 / 3	0 / 3	0 / 0	**+28**	**+34**	0	**+22**	**+30**	0	+1	+3	0
SR + $T1L_2$	2	0	0 / 2	0 / 2	0 / 0	**+18**	**+24**	0	**+15**	**+21**	0	0	+2	0
SR + $T2L_2$	2	0	0 / 3	0 / 2	0 / 0	**+20**	**+24**	0	**+16**	**+21**	0	0	+2	0
SR + $T3L_2$	2	0	0 / 3	0 / 2	0 / 0	**+20**	**+24**	0	**+16**	**+21**	0	0	+2	0
SR + GL_2	2	0	0 / 2	0 / 2	0 / 0	**+17**	**+24**	0	**+14**	**+21**	0	0	+2	0
SR + SPL_2	2	0	0 / 1	0 / 1	0 / 0	**+14**	**+20**	0	**+12**	**+18**	0	0	+2	0
SR + $T1_{2-6}$	2	0	0 / 3	0 / 3	0 / 0	**+24**	**+28**	0	**+18**	**+24**	0	0	+2	0
SR + $T2_{2-3}$	2	0	0 / 3	0 / 3	0 / 0	**+27**	**+26**	0	**+20**	**+23**	0	+1	+2	0
SR + $T3_{2-3}$	2	0	0 / 3	0 / 3	0 / 0	**+68**	**+76**	0	**+57**	**+71**	0	+4	**+24**	0
SR + G_{2-6}	2	0	0 / 3	0 / 3	0 / 0	**+26**	**+31**	0	**+19**	**+27**	0	+1	+2	0
SR + SP_{2-6}	2	0	0 / 3	0 / 3	0 / 0	**+23**	**+31**	0	**+18**	**+27**	0	0	+3	0
SR + $T1L_{2-6}$	2	0	0 / 4	0 / 2	0 / 0	**+181**	**+12**	0	**+146**	**+10**	0	**+10**	-3	0
SR + $T2L_{2-3}$	2	0	0 / 3	0 / 3	0 / 0	**+83**	**+19**	0	**+64**	**+16**	0	+3	0	0
SR + $T3L_{2-3}$	2	0	0 / 3	1 / 2	0 / 0	**+76**	**+6**	0	**+59**	**+5**	0	+2	-1	0
SR + GL_{2-6}	2	0	0 / 3	0 / 3	0 / 0	**+67**	**+33**	0	**+52**	**+29**	0	+2	+2	0
SR + SPL_{2-6}	2	0	0 / 2	0 / 2	0 / 0	**+16**	**+24**	0	**+13**	**+21**	0	0	+2	0

Table B.31. Performance impact of enabling particular multi-row tableau cut separators (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	6	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	5	0	19 / 26	20 / 20	6 / 9	+2	**-6**	-1	+1	-3	0	+2	+1	0
SR + $T2_2$	7	1	18 / 25	20 / 21	5 / 11	-3	**-13**	+1	-1	**-12**	+1	**+7**	**+28**	0
SR + $T3_2$	4	0	18 / 24	22 / 19	3 / 9	**-7**	**-16**	+2	**-8**	**-13**	+2	**-14**	**-16**	+1
SR + G_2	6	2	17 / 23	23 / 26	8 / 5	**+5**	+3	-4	**+7**	**+6**	-3	**+10**	**+15**	-1
SR + SP_2	5	0	13 / 11	19 / 9	2 / 3	**-9**	**-22**	-1	**-8**	**-17**	0	**-6**	**-11**	0
SR + $T1L_2$	6	0	20 / 33	25 / 23	4 / 11	**+5**	+3	-3	+4	+1	-1	**+7**	0	+1
SR + $T2L_2$	6	1	17 / 36	21 / 23	4 / 6	+1	**-15**	-4	0	**-14**	-3	**+6**	0	-1
SR + $T3L_2$	5	0	21 / 36	27 / 20	8 / 11	**+5**	**-18**	-2	+2	**-11**	-2	0	**+8**	-1
SR + GL_2	5	1	19 / 29	25 / 21	7 / 4	-4	**-17**	-4	-4	**-13**	-4	+1	**-7**	-2
SR + SPL_2	3	0	13 / 15	20 / 13	2 / 4	**-5**	**-10**	**-6**	**-6**	**-9**	-4	**-7**	**-14**	-2
SR + $T1_{2-6}$	5	0	15 / 39	21 / 24	6 / 8	**+7**	**-17**	-1	+3	**-14**	-1	+1	**-17**	-1
SR + $T2_{2-3}$	4	1	18 / 36	28 / 23	8 / 9	-2	**-28**	**-5**	-2	**-21**	-3	+3	+4	-2
SR + $T3_{2-3}$	4	0	16 / 35	26 / 19	8 / 8	0	**-17**	0	-1	**-14**	0	**-8**	**-9**	-1
SR + G_{2-6}	6	0	18 / 28	23 / 22	8 / 5	**+5**	**-8**	**-5**	**+5**	-3	-3	**+8**	**-9**	-1
SR + SP_{2-6}	3	0	16 / 22	17 / 14	4 / 4	**-8**	**-21**	-2	**-10**	**-16**	-1	**-19**	**-17**	0
SR + $T1L_{2-6}$	7	0	9 / 68	24 / 22	4 / 13	**+141**	**-33**	-2	**+101**	**-27**	-1	**+53**	**-25**	0
SR + $T2L_{2-3}$	8	0	12 / 57	22 / 25	5 / 10	**+76**	**-9**	**-6**	**+56**	**-8**	-3	**+33**	**-18**	-1
SR + $T3L_{2-3}$	5	1	11 / 59	23 / 29	8 / 9	**+86**	**-12**	**-6**	**+57**	**-7**	-4	**+25**	**+8**	-2
SR + GL_{2-6}	6	0	15 / 52	21 / 24	9 / 8	**+56**	+1	-4	**+38**	-1	-3	**+16**	-1	-2
SR + SPL_{2-6}	5	0	19 / 20	21 / 14	4 / 6	**-10**	**-10**	**-6**	**-11**	**-12**	-4	**-12**	**-9**	-2

Table B.32. Performance impact of enabling particular multi-row tableau cut separators (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	1	0	9 / 17	9 / 22	3 / 6	**+13**	**+29**	**+7**	**+11**	**+16**	**+5**	**+7**	**+16**	+4
SR + $T2_2$	1	0	7 / 24	10 / 21	5 / 8	**+24**	**+32**	**+6**	**+21**	**+20**	**+6**	**+16**	**+38**	**+5**
SR + $T3_2$	1	0	7 / 23	12 / 22	4 / 7	**+18**	**+20**	**+5**	**+13**	**+12**	**+5**	+2	**-14**	+4
SR + G_2	1	0	7 / 18	17 / 18	5 / 6	**+12**	**+11**	+2	**+6**	**+6**	+2	**-8**	**-22**	+2
SR + SP_2	1	0	7 / 19	11 / 16	3 / 4	**+13**	**+12**	+1	**+8**	**+13**	+2	**-5**	**-21**	+2
SR + $T1L_2$	1	0	7 / 23	9 / 21	2 / 9	**+22**	**+24**	**+7**	**+17**	**+17**	**+7**	**+7**	**+7**	**+6**
SR + $T2L_2$	1	0	4 / 25	14 / 21	4 / 7	**+32**	**+25**	**+11**	**+24**	**+15**	**+9**	**+19**	**+49**	**+6**
SR + $T3L_2$	1	0	6 / 22	13 / 18	3 / 6	**+11**	**+10**	**+7**	**+8**	+2	**+7**	+1	**-5**	**+5**
SR + GL_2	1	0	10 / 17	9 / 17	3 / 8	**+8**	+4	+3	**+6**	+1	+3	+4	**-7**	+3
SR + SPL_2	1	0	5 / 17	8 / 18	4 / 6	**+6**	**+7**	+1	+3	**+6**	+2	**-5**	**-20**	+2
SR + $T1_{2-6}$	1	0	8 / 24	13 / 15	4 / 11	**+24**	**+6**	**+9**	**+14**	+4	**+7**	**+10**	**+15**	+4
SR + $T2_{2-3}$	1	0	8 / 23	15 / 17	3 / 10	**+21**	**+12**	**+7**	**+15**	**+12**	**+7**	0	**-8**	**+5**
SR + $T3_{2-3}$	1	0	8 / 25	15 / 21	4 / 6	**+15**	**+10**	**+5**	**+10**	-2	+4	+1	**-15**	+3
SR + G_{2-6}	1	1	8 / 23	12 / 21	5 / 10	**+11**	**+13**	**+6**	**+5**	**+9**	**+6**	**-6**	**-24**	**+5**
SR + SP_{2-6}	1	0	7 / 16	14 / 13	3 / 5	+2	**-14**	+3	-2	**-12**	+3	**-6**	**-22**	+2
SR + $T1L_{2-6}$	2	0	2 / 38	19 / 17	5 / 10	**+271**	**-14**	**+7**	**+177**	**-6**	**+7**	**+112**	+3	**+5**
SR + $T2L_{2-3}$	1	1	3 / 35	14 / 18	6 / 7	**+133**	**+9**	**+10**	**+91**	+1	**+8**	**+46**	**+15**	**+5**
SR + $T3L_{2-3}$	1	0	3 / 32	13 / 23	4 / 7	**+121**	**+22**	**+6**	**+84**	**+9**	**+5**	**+34**	**-17**	+4
SR + GL_{2-6}	1	0	3 / 33	13 / 21	5 / 9	**+76**	**+8**	+4	**+52**	**+9**	+4	**+19**	**-25**	+4
SR + SPL_{2-6}	1	0	5 / 19	13 / 13	4 / 4	**+20**	**+5**	+2	**+21**	**+6**	+2	**+17**	**-11**	+3

Table B.33. Performance impact of enabling particular multi-row tableau cut separators (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	2	0	9 / 4	10 / 3	0 / 6	**-8**	**-8**	**+14**	**-7**	**-6**	**+12**	**-10**	**-8**	**+6**
SR + $T2_2$	2	0	5 / 7	7 / 8	1 / 5	-2	**-6**	**+7**	-1	-3	**+6**	+3	**+11**	+3
SR + $T3_2$	1	0	9 / 3	9 / 3	0 / 2	**-8**	**-11**	+2	**-8**	**-8**	+2	**-7**	-4	+1
SR + G_2	3	0	6 / 5	7 / 4	0 / 0	**-6**	-4	+2	**-5**	-2	+2	+1	-4	+1
SR + SP_2	2	0	7 / 5	6 / 4	1 / 4	**-5**	**-7**	**+6**	-4	-3	**+6**	+1	0	+3
SR + $T1L_2$	2	0	3 / 9	7 / 7	3 / 3	**+11**	0	+1	**+10**	+2	+1	-1	**+11**	0
SR + $T2L_2$	1	0	7 / 8	9 / 5	2 / 3	-1	**-17**	+2	-2	**-13**	+2	**-14**	**+5**	+2
SR + $T3L_2$	1	1	7 / 6	8 / 5	1 / 4	**-11**	**-21**	+3	**-11**	**-18**	+4	**-22**	**+5**	+4
SR + GL_2	2	0	7 / 9	7 / 6	1 / 1	**+8**	-4	0	**+8**	-1	0	+1	+4	0
SR + SPL_2	2	0	5 / 7	7 / 5	1 / 2	**+7**	0	+1	**+6**	+2	+1	+2	-4	+1
SR + $T1_{2-6}$	1	0	8 / 6	7 / 4	1 / 5	-4	**-7**	**+13**	**-5**	-4	**+12**	-4	**-5**	**+6**
SR + $T2_{2-3}$	1	0	9 / 6	8 / 5	1 / 6	**-6**	**-8**	**+8**	**-6**	**-7**	**+7**	**-8**	**-6**	+3
SR + $T3_{2-3}$	2	0	8 / 5	11 / 2	1 / 4	**-12**	**-16**	**+10**	**-11**	**-14**	**+8**	-2	**-9**	**+5**
SR + G_{2-6}	2	0	10 / 4	9 / 4	1 / 5	**-16**	**-12**	**+8**	**-15**	**-9**	**+6**	**-9**	-4	+3
SR + SP_{2-6}	2	0	7 / 6	7 / 4	1 / 4	+1	-4	+4	+1	-1	+4	**+6**	+3	+3
SR + $T1L_{2-6}$	3	0	6 / 8	9 / 6	2 / 6	**+50**	**-21**	**+8**	**+48**	**-21**	**+8**	**+37**	+4	**+6**
SR + $T2L_{2-3}$	3	0	6 / 10	8 / 7	3 / 5	**+42**	-3	+4	**+41**	-3	+4	**+34**	**+6**	+4
SR + $T3L_{2-3}$	3	0	5 / 9	6 / 6	2 / 3	**+49**	+2	0	**+47**	+3	+1	**+38**	**+6**	+1
SR + GL_{2-6}	1	0	8 / 7	9 / 6	2 / 2	**+16**	**-14**	**-5**	**+15**	**-13**	**-5**	**-6**	**-6**	-2
SR + SPL_{2-6}	1	0	5 / 5	8 / 3	2 / 3	+3	**-5**	-1	+2	-3	-1	**-6**	**-6**	0

Table B.34. Performance impact of enabling particular multi-row tableau cut separators (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	10	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	10	0	37 / 49	39 / 47	9 / 21	+4	+3	+3	+2	+2	+3	0	+2	+1
SR + $T2_2$	12	1	30 / 58	37 / 52	11 / 24	**+5**	0	+3	+4	-2	+3	**+6**	**+26**	+2
SR + $T3_2$	8	0	34 / 52	43 / 46	7 / 18	0	**-5**	+3	-2	**-5**	+3	**-9**	**-13**	+2
SR + G_2	12	2	30 / 48	47 / 50	13 / 11	**+6**	**+5**	-2	**+5**	**+5**	-1	**+5**	+2	0
SR + SP_2	10	0	27 / 38	36 / 32	6 / 11	-1	**-10**	+1	-3	**-6**	+1	-3	**-11**	+1
SR + $T1L_2$	11	0	30 / 67	41 / 53	9 / 23	**+11**	**+9**	+1	**+8**	**+6**	+2	+4	+4	+2
SR + $T2L_2$	10	1	28 / 72	44 / 51	10 / 16	**+9**	-4	+1	**+5**	**-5**	+1	+2	**+12**	+1
SR + $T3L_2$	9	1	34 / 67	48 / 45	12 / 21	**+5**	**-10**	+1	+2	**-8**	+1	**-5**	**+5**	+1
SR + GL_2	10	1	36 / 57	41 / 46	11 / 13	+1	**-9**	-2	+1	**-7**	-1	+1	-4	0
SR + SPL_2	8	0	23 / 40	35 / 37	7 / 12	0	-3	-3	-2	-2	-2	-4	**-13**	0
SR + $T1_{2-6}$	9	0	31 / 72	41 / 46	11 / 24	**+10**	**-8**	+4	**+5**	**-7**	+3	+1	**-7**	+1
SR + $T2_{2-3}$	8	1	35 / 68	51 / 48	12 / 25	+4	**-14**	0	+2	**-10**	+1	0	-1	+1
SR + $T3_{2-3}$	9	0	32 / 68	52 / 45	13 / 18	+4	**-8**	+3	+1	**-8**	+2	-4	**-10**	+1
SR + G_{2-6}	11	1	36 / 58	44 / 50	14 / 20	+4	-2	0	+3	0	+1	+2	**-11**	+1
SR + SP_{2-6}	8	0	30 / 47	38 / 34	8 / 13	-3	**-16**	0	**-5**	**-12**	+1	**-10**	**-13**	+1
SR + $T1L_{2-6}$	14	0	17 / 118	52 / 47	11 / 29	**+156**	**-26**	+2	**+106**	**-20**	+3	**+50**	**-12**	+2
SR + $T2L_{2-3}$	14	1	21 / 105	44 / 53	14 / 22	**+85**	-2	0	**+59**	-4	+1	**+30**	**-5**	+1
SR + $T3L_{2-3}$	11	1	19 / 103	43 / 60	14 / 19	**+89**	-1	-2	**+59**	-1	-1	**+26**	+2	0
SR + GL_{2-6}	10	0	26 / 95	43 / 54	16 / 19	**+56**	+2	-2	**+36**	+1	-1	**+10**	**-7**	0
SR + SPL_{2-6}	9	0	29 / 46	42 / 32	10 / 13	0	-4	-3	-1	**-5**	-2	**-6**	**-8**	0

Table B.35. Performance impact of enabling particular multi-row tableau cut separators (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	0	0	0 / 2	0 / 2	0 / 0	**+30**	**+66**	0	**+33**	**+70**	0	**+47**	**+111**	0
SR + $T2_2$	0	0	0 / 2	0 / 2	0 / 0	**+52**	**+69**	0	**+59**	**+74**	0	**+97**	**+118**	0
SR + $T3_2$	0	0	0 / 2	0 / 2	0 / 0	**+58**	**+99**	0	**+66**	**+108**	0	**+117**	**+214**	0
SR + G_2	0	0	0 / 2	0 / 2	0 / 0	**+41**	**+64**	0	**+46**	**+68**	0	**+74**	**+106**	0
SR + SP_2	0	0	0 / 3	0 / 3	0 / 0	**+79**	**+98**	0	**+81**	**+101**	0	**+97**	**+128**	0
SR + $T1L_2$	0	0	0 / 2	0 / 2	0 / 0	**+47**	**+64**	0	**+52**	**+68**	0	**+79**	**+106**	0
SR + $T2L_2$	0	0	0 / 3	0 / 2	0 / 0	**+54**	**+64**	0	**+58**	**+68**	0	**+84**	**+106**	0
SR + $T3L_2$	0	0	0 / 3	0 / 2	0 / 0	**+54**	**+64**	0	**+58**	**+68**	0	**+84**	**+106**	0
SR + GL_2	0	0	0 / 2	0 / 2	0 / 0	**+44**	**+64**	0	**+49**	**+68**	0	**+76**	**+106**	0
SR + SPL_2	0	0	0 / 1	0 / 1	0 / 0	**+36**	**+53**	0	**+42**	**+58**	0	**+71**	**+99**	0
SR + $T1_{2-6}$	0	0	0 / 3	0 / 3	0 / 0	**+66**	**+77**	0	**+65**	**+78**	0	**+66**	**+84**	0
SR + $T2_{2-3}$	0	0	0 / 3	0 / 3	0 / 0	**+74**	**+72**	0	**+75**	**+74**	0	**+82**	**+92**	0
SR + $T3_{2-3}$	0	0	0 / 3	0 / 3	0 / 0	**+236**	**+274**	0	**+262**	**+299**	0	**+698**	**+1095**	0
SR + G_{2-6}	0	0	0 / 3	0 / 3	0 / 0	**+70**	**+89**	0	**+72**	**+91**	0	**+85**	**+105**	0
SR + SP_{2-6}	0	0	0 / 3	0 / 3	0 / 0	**+62**	**+87**	0	**+66**	**+90**	0	**+89**	**+121**	0
SR + $T1L_{2-6}$	0	0	0 / 3	0 / 2	0 / 0	**+924**	**+36**	0	**+911**	**+36**	0	**+932**	**+36**	0
SR + $T2L_{2-3}$	0	0	0 / 3	0 / 3	0 / 0	**+300**	**+52**	0	**+294**	**+53**	0	**+282**	**+59**	0
SR + $T3L_{2-3}$	0	0	0 / 3	1 / 2	0 / 0	**+269**	**+16**	0	**+263**	**+16**	0	**+250**	**+17**	0
SR + GL_{2-6}	0	0	0 / 3	0 / 3	0 / 0	**+224**	**+95**	0	**+225**	**+100**	0	**+250**	**+144**	0
SR + SPL_{2-6}	0	0	0 / 2	0 / 2	0 / 0	**+43**	**+65**	0	**+48**	**+69**	0	**+75**	**+107**	0

Table B.36. Performance impact of enabling particular multi-row tableau cut separators (easy ACC instances). We consider an instance to be easy if it can be solved to optimality using the single-row cut separators in 60 seconds or less. There are 3 instances in the ACC test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	0	0	7 / 13	7 / 9	0 / 2	**+17**	**+10**	-2	**+13**	**+14**	0	**+11**	**+109**	+1
SR + $T2_2$	0	0	5 / 11	7 / 7	0 / 3	**+5**	**-7**	+1	**+5**	**-8**	+1	**+7**	**-15**	+1
SR + $T3_2$	0	0	4 / 11	6 / 8	0 / 1	**+9**	-2	+2	+3	+4	+2	0	**+34**	+1
SR + G_2	1	1	8 / 7	8 / 11	0 / 2	**+17**	**+36**	+1	**+27**	**+35**	+1	**+581**	**+1698**	+1
SR + SP_2	0	0	3 / 4	5 / 3	0 / 1	-1	**-27**	-3	-1	**-16**	-1	0	**-25**	0
SR + $T1L_2$	1	0	2 / 22	5 / 12	1 / 3	**+40**	**+46**	-1	**+50**	**+42**	0	**+687**	**+1076**	+1
SR + $T2L_2$	0	0	5 / 19	9 / 6	1 / 0	**+21**	**-6**	0	**+17**	**-6**	0	**+27**	**-23**	0
SR + $T3L_2$	0	0	4 / 21	9 / 7	2 / 1	**+28**	**-8**	+1	**+24**	+2	+1	**+28**	**+50**	0
SR + GL_2	0	0	2 / 15	6 / 9	1 / 2	**+13**	+2	+1	**+10**	**+5**	+1	**+8**	**+45**	+1
SR + SPL_2	0	0	3 / 8	7 / 5	0 / 1	**+6**	-2	**-5**	+3	+4	-2	+3	**+30**	0
SR + $T1_{2-6}$	0	0	3 / 25	7 / 10	0 / 1	**+38**	**+5**	-2	**+34**	**+5**	-1	**+56**	**+65**	0
SR + $T2_{2-3}$	0	0	5 / 19	12 / 6	1 / 2	**+26**	**-13**	0	**+20**	-4	0	**+30**	**-16**	0
SR + $T3_{2-3}$	0	0	3 / 23	7 / 8	1 / 1	**+23**	-1	0	**+18**	+4	0	**+23**	**+33**	0
SR + G_{2-6}	1	0	5 / 15	7 / 11	1 / 1	**+36**	**+36**	0	**+42**	**+44**	0	**+593**	**+505**	+1
SR + SP_{2-6}	0	0	3 / 10	4 / 5	0 / 1	**+5**	**-23**	-4	+4	**-12**	-2	**+6**	**-23**	0
SR + $T1L_{2-6}$	1	0	0 / 37	7 / 11	1 / 1	**+618**	-1	-2	**+556**	**+5**	-1	**+2363**	**+48**	0
SR + $T2L_{2-3}$	1	0	2 / 31	8 / 9	1 / 0	**+266**	**+23**	**-5**	**+256**	**+14**	-2	**+1109**	**+286**	0
SR + $T3L_{2-3}$	0	0	1 / 35	8 / 12	2 / 1	**+271**	+2	**-5**	**+240**	**+8**	-2	**+572**	**+73**	0
SR + GL_{2-6}	1	0	0 / 32	5 / 11	3 / 2	**+225**	**+75**	-1	**+216**	**+54**	-1	**+1023**	**+492**	0
SR + SPL_{2-6}	1	0	3 / 13	5 / 7	0 / 3	**+22**	**+36**	-4	**+29**	**+32**	-1	**+584**	**+1481**	+1

Table B.37. Performance impact of enabling particular multi-row tableau cut separators (easy CORAL instances). We consider an instance to be easy if it can be solved to optimality using the single-row cut separators in 60 seconds or less. There are 44 instances in the CORAL test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	0	0	7 / 13	7 / 19	2 / 4	**+12**	**+31**	+3	**+18**	**+15**	+3	**+23**	**+7**	+3
SR + $T2_2$	0	0	5 / 18	8 / 17	3 / 6	**+22**	**+33**	**+7**	**+30**	**+18**	**+6**	**+36**	**+10**	+4
SR + $T3_2$	0	0	4 / 17	8 / 18	3 / 3	**+23**	**+27**	+4	**+32**	**+16**	+4	**+46**	**+21**	+3
SR + G_2	0	0	4 / 15	13 / 14	3 / 3	**+25**	**+20**	0	**+34**	**+15**	0	**+41**	**+25**	0
SR + SP_2	0	0	5 / 14	8 / 12	2 / 2	**+16**	**+14**	+1	**+18**	**+17**	+2	**+23**	**+6**	+2
SR + $T1L_2$	0	0	5 / 16	6 / 16	1 / 6	**+20**	**+24**	**+5**	**+25**	**+16**	**+5**	**+29**	**-6**	**+5**
SR + $T2L_2$	0	0	2 / 18	11 / 15	3 / 4	**+38**	**+26**	**+6**	**+52**	**+13**	**+6**	**+68**	**+54**	+4
SR + $T3L_2$	0	0	4 / 18	10 / 15	2 / 3	**+19**	**+18**	**+5**	**+33**	**+7**	+4	**+44**	+4	+3
SR + GL_2	0	0	7 / 12	7 / 14	2 / 5	**+11**	+4	+2	**+16**	0	+2	**+19**	+3	+2
SR + SPL_2	0	0	3 / 11	5 / 13	3 / 4	**+8**	**+10**	+2	**+7**	**+9**	+2	**+10**	+3	+2
SR + $T1_{2-6}$	0	0	4 / 19	9 / 13	3 / 7	**+36**	**+10**	**+6**	**+46**	**+7**	**+5**	**+59**	**-7**	+3
SR + $T2_{2-3}$	0	0	5 / 19	11 / 15	1 / 7	**+32**	**+19**	**+7**	**+46**	**+20**	**+7**	**+59**	**+14**	+4
SR + $T3_{2-3}$	0	0	4 / 20	10 / 17	2 / 3	**+33**	**+25**	+3	**+46**	**+8**	+3	**+58**	**+5**	+2
SR + G_{2-6}	0	1	5 / 18	8 / 17	3 / 8	**+24**	**+28**	**+6**	**+32**	**+25**	**+6**	**+39**	**+15**	+4
SR + SP_{2-6}	0	0	3 / 14	10 / 10	2 / 3	**+18**	**-6**	+3	**+17**	-2	+3	**+18**	**+5**	+1
SR + $T1L_{2-6}$	0	0	1 / 30	13 / 13	2 / 7	**+417**	-2	**+8**	**+494**	+2	**+7**	**+1542**	**-10**	+4
SR + $T2L_{2-3}$	0	1	0 / 27	10 / 15	3 / 4	**+210**	**+21**	**+5**	**+274**	**+11**	**+5**	**+687**	**+9**	+3
SR + $T3L_{2-3}$	0	0	0 / 25	9 / 20	3 / 5	**+184**	**+37**	**+5**	**+248**	**+19**	+4	**+596**	**+9**	+3
SR + GL_{2-6}	0	0	0 / 25	10 / 18	2 / 6	**+116**	**+17**	**+5**	**+151**	**+21**	**+5**	**+304**	**+6**	+4
SR + SPL_{2-6}	0	0	2 / 14	9 / 10	3 / 3	**+33**	**+16**	+4	**+62**	**+19**	+4	**+421**	**+87**	+3

Table B.38. Performance impact of enabling particular multi-row tableau cut separators (easy MIPLIB instances). We consider an instance to be easy if it can be solved to optimality using the single-row cut separators in 60 seconds or less. There are 40 instances in the MIPLIB test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	0	0	1 / 0	1 / 0	0 / 1	**-27**	**-7**	**+42**	**-39**	**-11**	**+59**	**-46**	**-14**	**+67**
SR + $T2_2$	0	0	1 / 0	1 / 0	0 / 1	**-25**	**-7**	**+42**	**-36**	**-11**	**+59**	**-43**	**-13**	**+67**
SR + $T3_2$	0	0	1 / 0	1 / 0	0 / 0	**-12**	**-11**	0	**-18**	**-17**	0	**-23**	**-21**	0
SR + G_2	0	0	1 / 0	1 / 0	0 / 0	**-14**	**-12**	0	**-21**	**-19**	0	**-25**	**-23**	0
SR + SP_2	0	0	1 / 0	1 / 0	0 / 1	**-27**	**-10**	**+42**	**-39**	**-15**	**+59**	**-46**	**-18**	**+67**
SR + $T1L_2$	0	0	0 / 1	1 / 0	1 / 0	**+88**	**-11**	**-29**	**+145**	**-18**	**-45**	**+251**	**-21**	**-67**
SR + $T2L_2$	0	0	0 / 1	1 / 0	0 / 0	**+83**	**-6**	0	**+136**	**-9**	0	**+231**	**-11**	0
SR + $T3L_2$	0	0	0 / 1	0 / 0	0 / 0	**+29**	**-5**	0	**+46**	**-8**	0	**+66**	**-9**	0
SR + GL_2	0	0	0 / 1	0 / 0	0 / 0	**+65**	+2	0	**+105**	+3	0	**+170**	+4	0
SR + SPL_2	0	0	0 / 1	0 / 1	0 / 0	**+40**	**+14**	0	**+63**	**+22**	0	**+94**	**+29**	0
SR + $T1_{2-6}$	0	0	0 / 1	0 / 0	0 / 1	**+12**	-4	**+42**	**+19**	**-6**	**+59**	**+25**	**-8**	**+67**
SR + $T2_{2-3}$	0	0	1 / 0	0 / 0	0 / 1	**-14**	**-5**	**+42**	**-21**	**-8**	**+59**	**-25**	**-9**	**+67**
SR + $T3_{2-3}$	0	0	1 / 0	1 / 0	0 / 1	**-15**	**-12**	**+42**	**-23**	**-18**	**+59**	**-28**	**-22**	**+67**
SR + G_{2-6}	0	0	1 / 0	0 / 0	0 / 1	**-22**	-2	**+42**	**-32**	-4	**+59**	**-38**	-4	**+67**
SR + SP_{2-6}	0	0	1 / 0	1 / 0	0 / 0	**-8**	**-5**	0	**-12**	**-8**	0	**-15**	**-10**	0
SR + $T1L_{2-6}$	0	0	0 / 1	1 / 0	1 / 0	**+953**	**-5**	**-29**	**+1704**	**-8**	**-45**	**+10868**	**-10**	**-67**
SR + $T2L_{2-3}$	0	0	0 / 1	0 / 1	0 / 0	**+365**	**+9**	0	**+637**	**+15**	0	**+2038**	**+19**	0
SR + $T3L_{2-3}$	0	0	0 / 1	1 / 0	0 / 0	**+281**	**-10**	0	**+487**	**-16**	0	**+1340**	**-19**	0
SR + GL_{2-6}	0	0	0 / 1	0 / 1	1 / 0	**+176**	**+8**	**-73**	**+299**	**+13**	**-117**	**+655**	**+17**	**-200**
SR + SPL_{2-6}	0	0	0 / 1	0 / 0	1 / 0	**+12**	-3	**-53**	**+19**	-4	**-84**	**+25**	**-5**	**-133**

Table B.39. Performance impact of enabling particular multi-row tableau cut separators (easy MILP instances). We consider an instance to be easy if it can be solved to optimality using the single-row cut separators in 60 seconds or less. There are 2 instances in the MILP test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	0	0	15 / 28	15 / 30	2 / 7	**+14**	**+20**	+2	**+14**	**+16**	+2	**+15**	**+42**	+2
SR + $T2_2$	0	0	11 / 31	16 / 26	3 / 10	**+13**	**+11**	**+5**	**+14**	**+5**	+4	**+19**	+2	+2
SR + $T3_2$	0	0	9 / 30	15 / 28	3 / 4	**+16**	**+12**	+3	**+14**	**+11**	+3	**+19**	**+27**	+2
SR + G_2	1	1	13 / 24	22 / 27	3 / 5	**+21**	**+28**	0	**+28**	**+26**	0	**+401**	**+594**	+1
SR + SP_2	0	0	9 / 21	14 / 18	2 / 4	**+8**	**-8**	0	**+7**	+1	+1	**+11**	-4	+1
SR + $T1L_2$	1	0	7 / 41	12 / 30	3 / 9	**+32**	**+35**	+1	**+42**	**+30**	+2	**+476**	**+362**	+3
SR + $T2L_2$	0	0	7 / 41	21 / 23	4 / 4	**+31**	**+9**	+3	**+32**	+4	+2	**+46**	**+28**	+2
SR + $T3L_2$	0	0	8 / 43	19 / 24	4 / 4	**+25**	**+5**	+2	**+28**	**+6**	+2	**+37**	**+20**	+1
SR + GL_2	0	0	9 / 30	13 / 25	3 / 7	**+14**	**+5**	+1	**+15**	+4	+1	**+20**	**+18**	+1
SR + SPL_2	0	0	6 / 21	12 / 20	3 / 5	**+9**	**+5**	-2	**+8**	**+8**	0	**+12**	**+13**	+1
SR + $T1_{2-6}$	0	0	7 / 48	16 / 26	3 / 9	**+38**	**+9**	+3	**+38**	**+8**	+3	**+56**	**+18**	+2
SR + $T2_{2-3}$	0	0	11 / 41	23 / 24	2 / 10	**+29**	+2	+4	**+29**	**+8**	+4	**+39**	+4	+2
SR + $T3_{2-3}$	0	0	8 / 46	18 / 28	3 / 5	**+31**	**+15**	+3	**+33**	**+11**	+2	**+80**	**+19**	+1
SR + G_{2-6}	1	1	11 / 36	15 / 31	4 / 10	**+30**	**+33**	+4	**+36**	**+35**	+4	**+409**	**+182**	+2
SR + SP_{2-6}	0	0	7 / 27	15 / 18	2 / 4	**+12**	**-13**	0	**+11**	**-5**	0	**+15**	-4	+1
SR + $T1L_{2-6}$	1	0	1 / 71	21 / 26	4 / 8	**+535**	0	+2	**+530**	+4	+2	**+2278**	**+10**	+2
SR + $T2L_{2-3}$	1	1	2 / 62	18 / 28	4 / 4	**+243**	**+23**	0	**+259**	**+13**	+1	**+972**	**+103**	+1
SR + $T3L_{2-3}$	0	0	1 / 64	19 / 34	5 / 6	**+230**	**+16**	0	**+238**	**+13**	+1	**+573**	**+31**	+1
SR + GL_{2-6}	1	0	0 / 61	15 / 33	6 / 8	**+170**	**+46**	0	**+185**	**+39**	+1	**+787**	**+171**	+1
SR + SPL_{2-6}	1	0	5 / 30	14 / 19	4 / 6	**+27**	**+27**	-1	**+39**	**+26**	0	**+493**	**+561**	+2

Table B.40. Performance impact of enabling particular multi-row tableau cut separators (easy instances in the entire test set). We consider an instance to be easy if it can be solved to optimality using the single-row cut separators in 60 seconds or less. There are 89 instances in the entire test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T2_2$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T3_2$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + G_2	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + SP_2	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1L_2$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T2L_2$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T3L_2$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + GL_2	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + SPL_2	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_{2-6}$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T2_{2-3}$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T3_{2-3}$	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + G_{2-6}	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + SP_{2-6}	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1L_{2-6}$	2	0	0 / 1	0 / 0	0 / 0	**+6**	-4	0	**+6**	-4	0	**+5**	-4	0
SR + $T2L_{2-3}$	2	0	0 / 0	0 / 0	0 / 0	+1	-1	0	+1	-1	0	+1	-1	0
SR + $T3L_{2-3}$	2	0	0 / 0	0 / 0	0 / 0	+1	-1	0	+1	-1	0	+1	-1	0
SR + GL_{2-6}	2	0	0 / 0	0 / 0	0 / 0	+1	-1	0	+1	-1	0	+1	-1	0
SR + SPL_{2-6}	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0

Table B.41. Performance impact of enabling particular multi-row tableau cut separators (hard ACC instances). We consider an instance to be hard if it can not be solved to optimality using the single-row cut separators in 60 seconds or less. There are 4 instances in the ACC test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	6	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	5	0	12 / 13	13 / 11	6 / 7	**-7**	**-16**	0	-3	**-13**	-1	+2	0	-1
SR + $T2_2$	7	1	13 / 14	13 / 14	5 / 8	**-8**	**-16**	+1	-3	**-15**	+1	**+7**	**+28**	0
SR + $T3_2$	4	0	14 / 13	16 / 11	3 / 8	**-16**	**-24**	+2	**-13**	**-23**	+2	**-14**	**-16**	+1
SR + G_2	5	1	9 / 16	15 / 15	8 / 3	-2	**-15**	**-8**	+2	**-8**	**-6**	+3	+2	-3
SR + SP_2	5	0	10 / 7	14 / 6	2 / 2	**-13**	**-18**	+1	**-12**	**-18**	+1	**-6**	**-11**	0
SR + $T1L_2$	5	0	18 / 11	20 / 11	3 / 8	**-14**	**-19**	-4	**-10**	**-18**	-2	-2	**-8**	+1
SR + $T2L_2$	6	1	12 / 17	12 / 17	3 / 6	**-11**	**-20**	**-6**	**-6**	**-19**	**-5**	**+6**	0	-3
SR + $T3L_2$	5	0	17 / 15	18 / 13	6 / 10	**-8**	**-23**	**-5**	**-5**	**-18**	-3	0	**+8**	-2
SR + GL_2	5	1	17 / 14	19 / 12	6 / 2	**-14**	**-28**	**-8**	**-9**	**-23**	**-7**	+1	**-7**	-4
SR + SPL_2	3	0	10 / 7	13 / 8	2 / 3	**-12**	**-16**	**-6**	**-10**	**-16**	**-5**	**-7**	**-15**	-3
SR + $T1_{2-6}$	5	0	12 / 14	14 / 14	6 / 7	**-10**	**-29**	0	**-6**	**-24**	0	0	**-17**	-2
SR + $T2_{2-3}$	4	1	13 / 17	16 / 17	7 / 7	**-17**	**-36**	**-8**	**-9**	**-30**	**-6**	+2	+4	-4
SR + $T3_{2-3}$	4	0	13 / 12	19 / 11	7 / 7	**-13**	**-27**	0	**-8**	**-23**	0	**-8**	**-9**	-2
SR + G_{2-6}	5	0	13 / 13	16 / 11	7 / 4	**-12**	**-29**	**-8**	**-5**	**-23**	**-6**	+1	**-13**	-3
SR + SP_{2-6}	3	0	13 / 12	13 / 9	4 / 3	**-16**	**-19**	0	**-16**	**-19**	0	**-19**	**-17**	-1
SR + $T1L_{2-6}$	6	0	9 / 31	17 / 11	3 / 12	**+15**	**-49**	-2	**+21**	**-42**	-1	**+24**	**-25**	+1
SR + $T2L_{2-3}$	7	0	10 / 26	14 / 16	4 / 10	**+8**	**-25**	**-6**	**+14**	**-20**	-4	**+19**	**-20**	-2
SR + $T3L_{2-3}$	5	1	10 / 24	15 / 17	6 / 8	**+17**	**-21**	**-7**	**+18**	**-15**	**-6**	**+18**	**+7**	-4
SR + GL_{2-6}	5	0	15 / 20	16 / 13	6 / 6	-4	**-30**	**-6**	-1	**-24**	**-5**	+3	-4	-4
SR + SPL_{2-6}	4	0	16 / 7	16 / 7	4 / 3	**-27**	**-32**	**-8**	**-24**	**-31**	**-7**	**-20**	**-20**	**-5**

Table B.42. Performance impact of enabling particular multi-row tableau cut separators (hard CORAL instances). We consider an instance to be hard if it can not be solved to optimality using the single-row cut separators in 60 seconds or less. There are 66 instances in the CORAL test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	1	0	2 / 4	2 / 3	1 / 2	**+16**	**+24**	**+16**	**+15**	**+24**	**+11**	**+6**	**+17**	**+8**
SR + $T2_2$	1	0	2 / 6	2 / 4	2 / 2	**+31**	**+29**	+3	**+30**	**+29**	+4	**+15**	**+39**	**+7**
SR + $T3_2$	1	0	3 / 6	4 / 4	1 / 4	**+6**	**+5**	**+7**	**+7**	**+5**	**+8**	+1	**-15**	**+10**
SR + G_2	1	0	3 / 3	4 / 4	2 / 3	**-16**	**-11**	**+6**	**-11**	**-11**	**+8**	**-9**	**-24**	**+11**
SR + SP_2	1	0	2 / 5	3 / 4	1 / 2	**+6**	**+6**	+1	**+5**	**+6**	+2	**-5**	**-22**	**+5**
SR + $T1L_2$	1	0	2 / 7	3 / 5	1 / 3	**+26**	**+23**	**+10**	**+24**	**+23**	**+11**	**+6**	**+8**	**+12**
SR + $T2L_2$	1	0	2 / 7	3 / 6	1 / 3	**+19**	**+22**	**+22**	**+19**	**+22**	**+17**	**+18**	**+49**	**+14**
SR + $T3L_2$	1	0	2 / 4	3 / 3	1 / 3	**-6**	**-6**	**+12**	-4	**-6**	**+12**	0	**-5**	**+13**
SR + GL_2	1	0	3 / 5	2 / 3	1 / 3	0	+2	**+5**	+4	+2	**+6**	+3	**-7**	**+8**
SR + SPL_2	1	0	2 / 6	3 / 5	1 / 2	+3	+2	0	+2	+2	+1	**-5**	**-21**	+4
SR + $T1_{2-6}$	1	0	4 / 5	4 / 2	1 / 4	-2	-2	**+17**	0	-2	**+12**	**+8**	**+16**	**+8**
SR + $T2_{2-3}$	1	0	3 / 4	4 / 2	2 / 3	-4	-4	**+5**	+2	-4	**+6**	-2	**-9**	**+11**
SR + $T3_{2-3}$	1	0	4 / 5	5 / 4	2 / 3	**-20**	**-21**	**+8**	**-10**	**-21**	**+9**	0	**-16**	**+9**
SR + G_{2-6}	1	0	3 / 5	4 / 4	2 / 2	**-15**	**-17**	+4	**-10**	**-17**	**+6**	**-7**	**-26**	**+9**
SR + SP_{2-6}	1	0	4 / 2	4 / 3	1 / 2	**-28**	**-33**	+1	**-17**	**-33**	+2	**-7**	**-23**	+4
SR + $T1L_{2-6}$	2	0	1 / 8	6 / 4	3 / 3	**+60**	**-40**	**+5**	**+71**	**-22**	**+6**	**+75**	+3	**+10**
SR + $T2L_{2-3}$	1	0	3 / 8	4 / 3	3 / 3	**+13**	**-17**	**+22**	**+24**	**-17**	**+17**	**+29**	**+16**	**+15**
SR + $T3L_{2-3}$	1	0	3 / 7	4 / 3	1 / 2	**+17**	**-9**	**+8**	**+24**	**-9**	**+9**	**+19**	**-18**	**+11**
SR + GL_{2-6}	1	0	3 / 8	3 / 3	3 / 3	**+6**	**-13**	0	**+16**	**-13**	+2	**+11**	**-26**	**+6**
SR + SPL_{2-6}	1	0	3 / 5	4 / 3	1 / 1	**-8**	**-17**	-2	**+5**	**-17**	-1	**+6**	**-14**	+3

Table B.43. Performance impact of enabling particular multi-row tableau cut separators (hard MIPLIB instances). We consider an instance to be hard if it can not be solved to optimality using the single-row cut separators in 60 seconds or less. There are 15 instances in the MIPLIB test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	2	0	8 / 4	9 / 3	0 / 5	**-6**	**-8**	**+11**	**-6**	**-6**	**+10**	**-10**	**-8**	**+6**
SR + $T2_2$	2	0	4 / 7	6 / 8	1 / 4	0	**-6**	+3	0	-3	+3	+3	**+11**	+2
SR + $T3_2$	1	0	8 / 3	8 / 3	0 / 2	**-8**	**-11**	+2	**-8**	**-8**	+2	**-6**	-4	+1
SR + G_2	3	0	5 / 5	6 / 4	0 / 0	**-5**	-3	+2	**-5**	-1	+2	+1	-4	+1
SR + SP_2	2	0	6 / 5	5 / 4	1 / 3	-3	**-7**	+2	-3	-3	+3	+1	0	+2
SR + $T1L_2$	2	0	3 / 8	6 / 7	2 / 3	**+6**	+1	+4	**+6**	+3	+3	-1	**+11**	+1
SR + $T2L_2$	1	0	7 / 7	8 / 5	2 / 3	**-6**	**-18**	+2	**-7**	**-14**	+2	**-14**	**+5**	+2
SR + $T3L_2$	1	1	7 / 5	8 / 5	1 / 4	**-14**	**-23**	+4	**-14**	**-19**	+4	**-23**	**+5**	+4
SR + GL_2	2	0	7 / 8	7 / 6	1 / 1	+4	-4	0	+4	-1	0	0	+4	0
SR + SPL_2	2	0	5 / 6	7 / 4	1 / 2	+4	-2	+1	+4	+1	+1	+2	-4	+1
SR + $T1_{2-6}$	1	0	8 / 5	7 / 4	1 / 4	**-6**	**-8**	**+10**	**-6**	**-5**	**+9**	-4	**-5**	**+5**
SR + $T2_{2-3}$	1	0	8 / 6	8 / 5	1 / 5	**-6**	**-8**	+4	**-6**	**-7**	+4	**-8**	**-6**	+2
SR + $T3_{2-3}$	2	0	7 / 5	10 / 2	1 / 3	**-11**	**-17**	**+6**	**-11**	**-15**	**+6**	-2	**-9**	+4
SR + G_{2-6}	2	0	9 / 4	9 / 4	1 / 4	**-15**	**-13**	+4	**-15**	**-10**	+4	**-9**	-4	+2
SR + SP_{2-6}	2	0	6 / 6	6 / 4	1 / 4	+2	-4	+4	+2	-1	+4	**+6**	+3	+3
SR + $T1L_{2-6}$	3	0	6 / 7	8 / 6	1 / 6	**+26**	**-23**	**+11**	**+26**	**-22**	**+11**	**+23**	+4	**+7**
SR + $T2L_{2-3}$	3	0	6 / 9	8 / 6	3 / 5	**+28**	-4	+4	**+28**	-4	+4	**+32**	**+6**	+4
SR + $T3L_{2-3}$	3	0	5 / 8	5 / 6	2 / 3	**+37**	+3	0	**+37**	+4	+1	**+37**	**+6**	+1
SR + GL_{2-6}	1	0	8 / 6	9 / 5	1 / 2	**+8**	**-16**	-1	**+7**	**-14**	0	**-7**	**-6**	0
SR + SPL_{2-6}	1	0	5 / 4	8 / 3	1 / 3	+2	**-5**	+3	+1	-3	+3	**-6**	**-6**	+2

Table B.44. Performance impact of enabling particular multi-row tableau cut separators (hard MILP instances). We consider an instance to be hard if it can not be solved to optimality using the single-row cut separators in 60 seconds or less. There are 24 instances in the MILP test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR	10	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
SR + $T1_2$	10	0	22 / 21	24 / 17	7 / 14	-3	**-9**	**+5**	-1	**-6**	+4	0	+2	+1
SR + $T2_2$	12	1	19 / 27	21 / 26	8 / 14	-1	**-8**	+2	+2	**-6**	+2	**+6**	**+26**	+1
SR + $T3_2$	8	0	25 / 22	28 / 18	4 / 14	**-11**	**-16**	+3	**-9**	**-15**	+3	**-9**	**-13**	+2
SR + G_2	11	1	17 / 24	25 / 23	10 / 6	**-5**	**-11**	-3	-2	**-7**	-2	+1	**-5**	0
SR + SP_2	10	0	18 / 17	22 / 14	4 / 7	**-8**	**-12**	+1	**-8**	**-11**	+1	-3	**-11**	+1
SR + $T1L_2$	10	0	23 / 26	29 / 23	6 / 14	-4	**-9**	0	-2	**-8**	+1	-1	0	+2
SR + $T2L_2$	10	1	21 / 31	23 / 28	6 / 12	**-6**	**-14**	0	-3	**-12**	0	+2	**+12**	0
SR + $T3L_2$	9	1	26 / 24	29 / 21	8 / 17	**-9**	**-20**	0	**-7**	**-16**	+1	**-5**	+4	+1
SR + GL_2	10	1	27 / 27	28 / 21	8 / 6	**-8**	**-19**	-4	-4	**-15**	-3	+1	**-5**	-2
SR + SPL_2	8	0	17 / 19	23 / 17	4 / 7	**-6**	**-10**	-4	**-5**	**-9**	-3	-4	**-14**	-2
SR + $T1_{2-6}$	9	0	24 / 24	25 / 20	8 / 15	**-8**	**-20**	**+5**	**-5**	**-16**	+4	0	**-7**	+1
SR + $T2_{2-3}$	8	1	24 / 27	28 / 24	10 / 15	**-12**	**-25**	-3	**-7**	**-21**	-2	-1	-1	-1
SR + $T3_{2-3}$	9	0	24 / 22	34 / 17	10 / 13	**-13**	**-23**	+3	**-9**	**-20**	+2	**-5**	**-11**	+1
SR + G_{2-6}	10	0	25 / 22	29 / 19	10 / 10	**-13**	**-23**	-3	**-8**	**-19**	-2	-2	**-14**	-1
SR + SP_{2-6}	8	0	23 / 20	23 / 16	6 / 9	**-14**	**-18**	+1	**-12**	**-17**	+1	**-10**	**-13**	+1
SR + $T1L_{2-6}$	13	0	16 / 47	31 / 21	7 / 21	**+23**	**-42**	+2	**+28**	**-34**	+3	**+26**	**-12**	+3
SR + $T2L_{2-3}$	13	0	19 / 43	26 / 25	10 / 18	**+13**	**-19**	+1	**+18**	**-15**	+1	**+20**	**-6**	+1
SR + $T3L_{2-3}$	11	1	18 / 39	24 / 26	9 / 13	**+20**	**-14**	-3	**+22**	**-10**	-2	**+20**	+2	-1
SR + GL_{2-6}	9	0	26 / 34	28 / 21	10 / 11	0	**-24**	-4	+3	**-20**	-3	+1	**-9**	-2
SR + SPL_{2-6}	8	0	24 / 16	28 / 13	6 / 7	**-18**	**-24**	-4	**-14**	**-23**	-4	**-12**	**-15**	-2

Table B.45. Performance impact of enabling particular multi-row tableau cut separators (hard instances in the entire test set). We consider an instance to be hard if it can not be solved to optimality using the single-row cut separators in 60 seconds or less. There are 109 instances in the entire test set falling into this category.

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR + two-row cuts	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only SR cuts	2	0	2 / 0	3 / 0	0 / 0	**-20**	**-20**	0	**-16**	**-18**	0	0	-2	0
no $T1_2$	2	0	1 / 0	1 / 0	0 / 0	**-11**	**-14**	0	**-10**	**-13**	0	0	-1	0
no $T2_2$	2	0	0 / 0	0 / 1	0 / 0	+1	+1	0	+1	+1	0	0	0	0
no $T3_2$	2	0	1 / 0	1 / 0	0 / 0	**-11**	**-14**	0	**-10**	**-13**	0	0	-1	0
no G_2	2	0	1 / 0	1 / 1	0 / 0	**-11**	**-13**	0	**-10**	**-12**	0	0	-2	0
no SP_2	2	0	1 / 1	2 / 0	0 / 0	**-6**	**-16**	0	**-6**	**-14**	0	0	-2	0
no $T1L_2$	2	0	1 / 0	1 / 1	0 / 0	**-12**	**-13**	0	**-10**	**-12**	0	0	-2	0
no $T2L_2$	2	0	1 / 0	1 / 0	0 / 0	**-13**	**-14**	0	**-11**	**-13**	0	0	-2	0
no $T3L_2$	2	0	1 / 0	1 / 1	0 / 0	**-13**	**-13**	0	**-11**	**-12**	0	0	-2	0
no GL_2	2	0	1 / 0	1 / 1	0 / 0	**-12**	**-13**	0	**-10**	**-12**	0	0	-1	0
no SPL_2	2	0	1 / 0	1 / 1	0 / 0	**-12**	**-13**	0	**-10**	**-12**	0	0	-2	0

Table B.46. Performance impact of disabling particular two-row tableau cut separators (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR + two-row cuts	4	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only SR cuts	6	0	45 / 18	27 / 28	7 / 8	**-10**	**+26**	**+7**	**-5**	**+20**	**+5**	+3	0	+2
no $T1_2$	7	0	8 / 13	7 / 13	2 / 2	+4	+4	0	**+6**	**+6**	0	**+19**	**+38**	0
no $T2_2$	4	0	12 / 10	13 / 9	2 / 3	**-6**	**-10**	0	-4	**-6**	0	-1	**+5**	0
no $T3_2$	5	0	10 / 12	12 / 13	3 / 1	+3	0	-1	+4	+3	-1	**+8**	+4	0
no G_2	3	1	5 / 13	6 / 12	1 / 3	+3	**+10**	+1	+2	**+5**	+1	+3	**+6**	+1
no SP_2	4	1	6 / 14	8 / 11	1 / 2	**+8**	**+8**	0	**+8**	**+9**	0	**+12**	**+9**	0
no $T1L_2$	4	1	19 / 11	12 / 13	2 / 6	**-5**	**-6**	+1	-2	-3	+1	+3	0	0
no $T2L_2$	4	0	27 / 8	20 / 14	4 / 5	**-13**	**-17**	0	**-9**	**-11**	0	-2	+1	0
no $T3L_2$	3	0	25 / 9	15 / 12	2 / 3	**-6**	-2	0	-4	-1	0	**-7**	-2	0
no GL_2	3	1	9 / 15	9 / 12	4 / 4	+2	+2	0	+3	+4	0	**+8**	**+8**	0
no SPL_2	4	1	19 / 11	12 / 13	2 / 6	**-5**	**-6**	+1	-2	-3	+1	+3	0	0

Table B.47. Performance impact of disabling particular two-row tableau cut separators (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR + two-row cuts	1	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only SR cuts	1	0	29 / 8	26 / 10	8 / 4	**-35**	**-35**	**-6**	**-30**	**-22**	**-6**	**-10**	+4	**-5**
no $T1_2$	1	0	7 / 4	9 / 5	0 / 2	-4	**-9**	+1	-4	**-7**	+1	**+13**	**+34**	0
no $T2_2$	1	0	13 / 8	13 / 12	1 / 2	**-9**	-4	+1	**-7**	-3	+1	**+5**	**+8**	0
no $T3_2$	1	0	9 / 9	10 / 13	0 / 3	-4	+1	+1	-2	+3	+1	**+9**	**+22**	+1
no G_2	1	0	6 / 8	9 / 6	1 / 1	+2	**-9**	0	+2	-2	0	**+13**	**+21**	0
no SP_2	1	0	9 / 9	10 / 11	0 / 4	-2	-3	+3	-2	-2	+2	0	+7	+1
no $T1L_2$	1	0	16 / 9	21 / 10	4 / 3	**-7**	**-18**	**+5**	**-6**	**-15**	+3	**+12**	**+43**	-1
no $T2L_2$	1	0	17 / 8	16 / 11	2 / 7	**-8**	**-10**	+3	**-7**	**-5**	+2	-4	-3	+1
no $T3L_2$	1	0	20 / 7	18 / 7	2 / 6	**-11**	**-13**	+2	**-9**	**-9**	+2	**+21**	**+55**	+1
no GL_2	1	0	11 / 5	11 / 7	3 / 3	**-5**	**-6**	+2	-3	**-5**	+1	**+23**	**+45**	0
no SPL_2	1	0	16 / 9	21 / 10	4 / 3	**-7**	**-18**	**+5**	**-6**	**-15**	+3	**+12**	**+43**	-1

Table B.48. Performance impact of disabling particular two-row tableau cut separators (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR + two-row cuts	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only SR cuts	1	0	9 / 4	6 / 8	4 / 1	**+6**	**+20**	-4	**+7**	**+20**	-3	**+10**	+4	-2
no $T1_2$	2	0	3 / 2	5 / 1	0 / 3	+1	**-8**	+3	+1	-4	+2	-1	-1	0
no $T2_2$	3	0	2 / 3	5 / 1	0 / 2	0	**-13**	**+5**	0	**-9**	**+5**	**+7**	-1	+2
no $T3_2$	2	0	2 / 4	3 / 2	2 / 2	+3	**-8**	+1	+3	-3	+1	+2	+1	0
no G_2	2	0	2 / 2	3 / 1	1 / 1	**+5**	**-6**	0	**+5**	-3	0	+3	0	-1
no SP_2	2	0	0 / 1	2 / 1	1 / 1	+3	-3	0	+3	-1	0	+2	+3	-1
no $T1L_2$	3	0	4 / 5	5 / 4	1 / 3	**+5**	+1	**+7**	**+5**	+2	**+6**	**+8**	**+8**	+3
no $T2L_2$	3	0	7 / 4	6 / 3	3 / 2	+1	-4	**+5**	+1	-4	+4	+4	-3	+2
no $T3L_2$	2	0	4 / 5	3 / 5	0 / 1	**+22**	**+20**	+1	**+22**	**+23**	+1	**+16**	+4	0
no GL_2	2	0	3 / 1	4 / 1	0 / 1	-3	**-6**	+1	-3	-3	+1	-4	+1	+2
no SPL_2	3	0	4 / 5	5 / 4	1 / 3	**+5**	+1	**+7**	**+5**	+2	**+6**	**+8**	**+8**	+3

Table B.49. Performance impact of disabling particular two-row tableau cut separators (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
SR + two-row cuts	9	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
only SR cuts	10	0	85 / 30	62 / 46	19 / 13	**-17**	+2	+2	**-10**	**+5**	+1	+3	+2	-1
no $T1_2$	12	0	19 / 19	22 / 19	2 / 7	+1	-2	0	+2	0	0	**+11**	**+28**	0
no $T2_2$	10	0	27 / 21	31 / 23	3 / 7	**-6**	**-8**	+1	-4	**-6**	+1	+2	+4	0
no $T3_2$	10	0	22 / 25	26 / 28	5 / 6	0	-1	0	+2	+2	0	**+6**	**+7**	0
no G_2	8	1	14 / 23	19 / 20	3 / 5	+2	+1	0	+2	+1	0	+4	**+8**	0
no SP_2	9	1	16 / 25	22 / 23	2 / 7	+4	+3	+1	+4	+4	+1	**+7**	**+7**	0
no $T1L_2$	10	1	40 / 25	39 / 28	7 / 12	-4	**-9**	+3	-3	**-6**	+2	+4	**+11**	0
no $T2L_2$	10	0	52 / 20	43 / 28	9 / 14	**-10**	**-13**	+1	**-7**	**-9**	+1	-1	-1	0
no $T3L_2$	8	0	50 / 21	37 / 25	4 / 10	-4	-3	+1	-2	-1	+1	+2	**+11**	0
no GL_2	8	1	24 / 21	25 / 21	7 / 8	-1	-1	+1	0	0	0	**+6**	**+14**	0
no SPL_2	10	1	40 / 25	39 / 28	7 / 12	-4	**-9**	+3	-3	**-6**	+2	+4	**+11**	0

Table B.50. Performance impact of disabling particular two-row tableau cut separators (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
no cut selection	1	2	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
violation	2	0	3 / 0	1 / 2	0 / 0	**-42**	**+40**	0	**-39**	**+41**	0	**-6**	**+126**	0
rel. violation	1	0	4 / 1	1 / 3	0 / 0	**-37**	**+57**	0	**-34**	**+58**	0	**-42**	**+59**	0
distance	2	0	3 / 2	3 / 2	0 / 0	**-11**	**+83**	0	-3	**+87**	0	**+26**	**+153**	0
obj. parallelism	2	0	4 / 0	1 / 4	0 / 0	**-39**	**+48**	0	**-38**	**+48**	0	**-32**	**+63**	0
exp. improvement	1	1	4 / 0	2 / 3	0 / 0	**-48**	**+39**	0	**-44**	**+39**	0	**-33**	**+61**	0
support	0	0	5 / 0	3 / 1	0 / 0	**-66**	**-19**	0	**-64**	**-19**	0	**-78**	**-42**	0
int. support	1	0	3 / 0	0 / 4	0 / 0	**-30**	**+72**	0	**-27**	**+72**	0	-2	**+127**	0
distance var. I	2	0	3 / 1	1 / 2	0 / 0	**-48**	**+13**	0	**-44**	**+23**	0	**-6**	**+117**	0
distance var. II	2	0	3 / 0	1 / 4	0 / 0	**-29**	**+72**	0	**-26**	**+73**	0	-2	**+105**	0

Table B.51. Performance impact of using particular cut quality measures (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
no cut selection	6	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
violation	5	1	59 / 34	34 / 52	16 / 9	**-32**	**+13**	-2	**-30**	**+22**	-2	**-31**	**+18**	-2
rel. violation	8	0	52 / 34	34 / 59	18 / 10	**-29**	**+15**	0	**-25**	**+22**	0	**-22**	**+21**	+1
distance	5	1	60 / 29	41 / 44	20 / 5	**-39**	**-11**	**-7**	**-36**	-2	**-7**	**-41**	**+24**	**-5**
obj. parallelism	11	1	60 / 32	29 / 60	11 / 27	**-22**	**+66**	**+22**	**-20**	**+61**	**+18**	**-11**	**+53**	**+9**
exp. improvement	7	1	66 / 27	39 / 48	16 / 17	**-33**	**+18**	**+9**	**-31**	**+21**	**+6**	**-27**	**+76**	+4
support	1	0	66 / 26	32 / 47	16 / 9	**-49**	**-25**	-4	**-45**	**-13**	**-5**	**-48**	0	-4
int. support	5	0	67 / 25	39 / 48	12 / 15	**-41**	0	+3	**-38**	+1	0	**-35**	**+35**	-1
distance var. I	7	0	64 / 28	40 / 50	21 / 5	**-47**	**-24**	**-7**	**-42**	**-13**	**-6**	**-37**	-3	-4
distance var. II	8	0	56 / 36	33 / 55	16 / 16	**-34**	+2	+4	**-30**	**+16**	+2	**-25**	**+30**	0

Table B.52. Performance impact of using particular cut quality measures (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
no cut selection	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
violation	1	1	21 / 19	12 / 30	11 / 13	**-17**	**+57**	+1	**-17**	**+26**	0	**-35**	**+116**	-1
rel. violation	0	2	21 / 16	14 / 27	14 / 9	**-14**	**+27**	**-6**	**-17**	**+25**	**-8**	**-62**	**+59**	**-6**
distance	0	1	25 / 11	17 / 23	13 / 9	**-36**	**-6**	**+7**	**-38**	**-14**	0	**-72**	**+29**	**-5**
obj. parallelism	2	0	20 / 19	7 / 34	5 / 25	+3	**+331**	**+30**	**+8**	**+197**	**+25**	-2	**+234**	**+17**
exp. improvement	0	2	27 / 11	17 / 22	11 / 10	**-35**	-2	**+5**	**-34**	**-12**	0	**-68**	**+36**	**-5**
support	0	1	25 / 12	20 / 18	13 / 8	**-35**	**-18**	+1	**-33**	**-19**	-3	**-63**	**+36**	**-6**
int. support	0	1	19 / 21	11 / 26	11 / 10	**-25**	+4	**-7**	**-26**	-3	**-10**	**-51**	**+90**	**-8**
distance var. I	0	1	25 / 14	14 / 23	15 / 8	**-23**	**+17**	**-8**	**-24**	**+8**	**-8**	**-61**	**+34**	**-6**
distance var. II	2	1	22 / 16	8 / 32	9 / 14	**-15**	**+115**	**+11**	**-14**	**+66**	**+7**	**-41**	**+94**	+2

Table B.53. Performance impact of using particular cut quality measures (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
no cut selection	5	1	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
violation	1	0	13 / 6	8 / 10	4 / 5	**-31**	**+21**	**+5**	**-31**	**+23**	+3	**-29**	-1	-1
rel. violation	2	1	14 / 7	14 / 8	9 / 2	**-37**	**-6**	**-6**	**-37**	**-6**	**-8**	**-34**	**+13**	**-8**
distance	0	0	17 / 3	13 / 8	7 / 4	**-53**	**-16**	**+11**	**-52**	**-15**	**+5**	**-51**	**-11**	-2
obj. parallelism	4	0	14 / 3	7 / 11	4 / 5	**-46**	**+18**	**+16**	**-44**	**+18**	**+12**	**-21**	**+37**	+3
exp. improvement	2	0	16 / 4	11 / 8	7 / 4	**-47**	**+5**	+3	**-47**	-2	+1	**-34**	-3	-2
support	3	0	13 / 5	7 / 12	6 / 2	**-31**	**+12**	**-7**	**-31**	**+16**	**-7**	**-21**	**-7**	**-6**
int. support	2	0	13 / 5	11 / 7	9 / 2	**-43**	**-6**	**-9**	**-43**	-2	**-8**	**-28**	**+15**	**-7**
distance var. I	0	0	17 / 5	14 / 8	5 / 2	**-52**	**-19**	-3	**-52**	**-18**	**-5**	**-57**	**-7**	**-5**
distance var. II	0	0	15 / 6	11 / 9	6 / 5	**-46**	+4	**+13**	**-46**	**+10**	**+7**	**-39**	**+33**	-1

Table B.54. Performance impact of using particular cut quality measures (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
no cut selection	14	3	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
violation	9	2	96 / 59	55 / 94	31 / 27	**-28**	**+26**	0	**-27**	**+23**	-1	**-30**	**+24**	-2
rel. violation	11	3	91 / 58	63 / 97	41 / 21	**-27**	**+16**	-2	**-25**	**+20**	-3	**-30**	**+23**	-2
distance	7	2	105 / 45	74 / 77	40 / 18	**-39**	**-9**	-1	**-37**	**-5**	-3	**-44**	**+15**	**-5**
obj. parallelism	19	1	98 / 54	44 / 109	20 / 57	**-20**	**+106**	**+23**	**-18**	**+82**	**+19**	**-13**	**+69**	**+11**
exp. improvement	10	4	113 / 42	69 / 81	34 / 31	**-36**	**+11**	**+7**	**-34**	**+9**	+4	**-33**	**+50**	+1
support	4	1	109 / 43	62 / 78	35 / 19	**-44**	**-19**	-3	**-40**	**-11**	-4	**-45**	+2	-4
int. support	8	1	102 / 51	61 / 85	32 / 27	**-37**	+2	-1	**-35**	+1	-3	**-34**	**+36**	-4
distance var. I	9	1	109 / 48	69 / 83	41 / 15	**-42**	**-13**	**-6**	**-39**	**-8**	**-6**	**-43**	0	-4
distance var. II	12	1	96 / 58	53 / 100	31 / 35	**-31**	**+28**	**+7**	**-28**	**+28**	+4	**-29**	**+38**	+1

Table B.55. Performance impact of using particular cut quality measures (entire test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
default	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no cut selection	1	2	3 / 2	3 / 2	0 / 0	**-17**	**-66**	0	**-21**	**-66**	0	**-22**	**-64**	0
no int. support	2	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no obj. parallelism	2	0	1 / 0	1 / 0	0 / 0	**-26**	**-37**	0	**-24**	**-36**	0	-2	**-10**	0
only distance	2	0	1 / 0	1 / 0	0 / 0	**-26**	**-37**	0	**-24**	**-36**	0	-2	**-10**	0

Table B.56. Performance impact of changing the default cut quality measure (ACC test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
default	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no cut selection	6	0	27 / 70	38 / 54	2 / 23	**+98**	**+61**	**+9**	**+90**	**+39**	**+7**	**+145**	**+22**	**+5**
no int. support	6	0	26 / 36	22 / 41	4 / 12	**+29**	**+63**	+3	**+28**	**+49**	+2	**+70**	**+135**	+1
no obj. parallelism	2	0	24 / 19	25 / 21	6 / 6	**+6**	**+23**	0	**+8**	**+19**	0	**+24**	**+6**	0
only distance	5	1	29 / 40	30 / 42	6 / 11	**+31**	**+60**	+2	**+29**	**+48**	+1	**+57**	**+54**	0

Table B.57. Performance impact of changing the default cut quality measure (CORAL test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
default	0	1	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no cut selection	2	0	16 / 27	24 / 16	4 / 16	**+36**	**+8**	**+17**	**+40**	**+18**	**+15**	**+165**	-4	**+9**
no int. support	0	1	16 / 10	16 / 16	2 / 6	**-9**	0	+2	**-8**	-2	+1	**-11**	**+16**	0
no obj. parallelism	0	1	15 / 13	20 / 13	7 / 4	**-7**	**-16**	+2	-4	**-9**	0	-1	0	-2
only distance	0	1	24 / 8	19 / 18	2 / 8	**-15**	0	**+22**	**-15**	+1	**+15**	**-26**	**+24**	+4

Table B.58. Performance impact of changing the default cut quality measure (MIPLIB test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
default	0	0	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no cut selection	5	1	4 / 19	7 / 13	3 / 7	**+125**	**+27**	+1	**+125**	**+25**	+3	**+188**	**+20**	**+5**
no int. support	1	0	6 / 8	6 / 7	2 / 2	**-5**	+1	+3	-2	+2	+2	**+33**	-3	+1
no obj. parallelism	1	0	7 / 5	7 / 6	2 / 3	**+14**	**+17**	+1	**+15**	**+17**	+1	**+41**	**+9**	+1
only distance	0	0	5 / 10	4 / 10	1 / 3	**+5**	**+10**	**+12**	**+6**	**+9**	**+8**	**+36**	**+10**	+3

Table B.59. Performance impact of changing the default cut quality measure (MILP test set)

			wins / losses			geom. mean			shifted geom. mean			total		
setting	T	F	time	nodes	gap	time	nodes	gap	time	nodes	gap	time	nodes	gap
default	2	1	0 / 0	0 / 0	0 / 0	0	0	0	0	0	0	0	0	0
no cut selection	14	3	50 / 118	72 / 85	9 / 46	**+76**	**+32**	**+10**	**+73**	**+25**	**+9**	**+140**	**+18**	**+6**
no int. support	9	1	48 / 54	44 / 64	8 / 20	**+11**	**+30**	+3	**+13**	**+24**	+2	**+48**	**+82**	0
no obj. parallelism	5	1	47 / 37	53 / 40	15 / 13	+2	**+6**	+1	**+5**	**+7**	0	**+23**	**+6**	-1
only distance	7	2	59 / 58	54 / 70	9 / 22	**+11**	**+29**	**+10**	**+12**	**+24**	**+6**	**+39**	**+38**	+1

Table B.60. Performance impact of changing the default cut quality measure (entire test set)

Appendix C.

Notation

In this chapter of the appendix we describe the notation we use in this thesis.

Basic sets	
$\emptyset$	the empty set
$\mathbb{R}$	the set of the real numbers
$\mathbb{R}^n$	the set of the n-dimensional real numbers
$\mathbb{Q}$	the set of the rational numbers
$\mathbb{Q}^n$	the set of the n-dimensional rational numbers
$\mathbb{Z}$	the set of the integer numbers
$\mathbb{Z}^n$	the set of the n-dimensional integer numbers
$\mathbb{R}^L$	the set of vectors indexed by the set L, i.e. $x \in \mathbb{R}^L \Leftrightarrow x = (x_j)_{j \in L}$ and $x_j \in \mathbb{R}$
Basic set operations	
$\lvert A \rvert$	the cardinality of the set A
$A \subseteq B$	A is a subset of B or $A = B$
$A = (B, C)$	(B, C) is a partition of the set A, i.e. $A = B \cup C$ and $B \cap C = \emptyset$
$\text{int}(A)$	the interior of the set A
$\text{boundary}(A)$	the boundary of the set A
$\text{conv}(A)$	the convex hull of the set A

Table C.1. Notation

MIP notation	
N	the set of variables
N_I	the set of integer variables, i.e. $N_I \subseteq N$
A	the coefficient matrix
b	the right-hand side vector
x	the vector of variables
c	the objective vector
Bases and the simplex tableau	
B	the set of basic variables, i.e. a basis of the LP relaxation of an MIP
B_I	the set of basic integer-constrained variables, i.e. $B_I = B \cap N_I$
J	the set of non-basic variables, i.e. $J = N \setminus B$
J_I	the set of non-basic integer-constrained variables, i.e. $J_I = J \cap N_I$
Sets	
X_{MIP}	the set of feasible solutions of an MIP
P	a polyhedron; the set of feasible solutions of an LP; the set of feasible solutions of the LP relaxation of an MIP
P_I	the integer hull; the convex hull of integer solutions in P
X_{DP}	a disjunctive set; the set of feasible solutions of disjunctive program; the set of feasible solutions of the disjunctive relaxation of an MIP
P^i	a polyhedron; the set of feasible solutions to a disjunct involved in the description of the disjunctive set X_{DP}
Solutions	
x^*	the optimal solution of an LP; the optimal solution of the LP relaxation of an MIP
c^*	the optimal objective value of an LP; the optimal objective value of the LP relaxation of an MIP
$\bar{x}$	the optimal solution of an MIP
$\bar{c}$	the optimal objective value of an MIP
$\hat{x}$	the current incumbent solution (branch-and-bound) of an MIP; a feasible solution of an MIP
$\hat{c}$	the objective value of the incumbent solution of an MIP

Table C.2. Notation (continued)

	Disjunctions
$D(\pi, \pi_0)$	the split disjunction $\pi x \leq \pi_0$ or $\pi x \geq \pi_0 + 1$
$\epsilon(\pi, \pi_0)$	the amount by which the first term of the split disjunction $D(\pi, \pi_0)$ is violated by an LP solution x^*, i.e. $\pi x^* - \pi_0$
	Miscellaneous
$\lvert a \rvert$	the absolute value of a
$\lVert a \rVert$	the (Euclidean) norm of the vector a
$\lfloor a \rfloor$	the largest integer less or equal to a
$\lceil a \rceil$	the smallest integer greater or equal to a
$(a)^+$	the maximum of a and 0
$\wedge$	the logical "and"
$\vee$	the logical "or"
$r = a \bmod b$	the remainder r of the division $\frac{a}{b}$ where a and b are integers, i.e. $a = \lfloor \frac{a}{b} \rfloor \cdot b + r$
I	the identity matrix, i.e. the matrix with ones on the main diagonal and zeros elsewhere
e_i	the unit vector with a one in the i^{th} position and zeros elsewhere
e	the vector of all ones
e^n	the vector of all ones of dimension n
(A, B)	the matrix (or vector) obtained by placing the matrix (or vector) A next to the matrix (or vector) B

Table C.3. Notation (continued)

List of Abbreviations

CG	Chvátal-Gomory
CGLP	cut generating linear program
cGMI	combined Gomory mixed-integer
EMR	external model representation
GMI	Gomory mixed-integer
IMR	internal model representation
IP	integer program
IPM	interior point method
L&P	lift-and-project
LP	
	linear programming
	linear program
MIP	
	mixed-integer programming
	mixed-integer program
MIR	mixed-integer rounding
P&R	pivot-and-reduce
R&S	reduce-and-split

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] COIN-OR – Computational Infrastructure for Operations Research. `http://www.coin-or.org`.

[2] SCIP – Solving Constraint Integer Programs. `http://scip.zib.de`.

[3] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[4] Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[5] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

[6] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. `http://miplib.zib.de`.

[7] Kent Andersen. *Intersection Cuts, Split Cuts and Mixed Integer Gomory Cuts*. PhD thesis, Graduate School of Industrial Administration, Carnegie Mellon University, 2003.

[8] Kent Andersen, Gérard Cornuéjols, and Yanjun Li. Reduce-and-split cuts: Improving the performance of mixed-integer Gomory cuts. *Management Science*, 51(11):1720–1732, 2005.

[9] Kent Andersen, Gérard Cornuéjols, and Yanjun Li. Split closure and intersection cuts. *Mathematical Programming*, 102(3):457–493, 2005.

[10] Kent Andersen, Quentin Louveaux, and Robert Weismantel. Mixed-integer sets from two rows of two adjacent simplex bases. *Mathematical Programming*, 124(1–2):455–480, 2010.

[11] Kent Andersen, Quentin Louveaux, Robert Weismantel, and Laurence A. Wolsey. Inequalities from Two Rows of a Simplex Tableau. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, volume 4513 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[12] Kent Andersen, Christian Wagner, and Robert Weismantel. On an Analysis of the Strength of Mixed-Integer Cutting Planes from Multiple Simplex Tableau Rows. *SIAM Journal on Optimization*, 20(2):967–982, 2009.

[13] Giuseppe Andreello, Alberto Caprara, and Matteo Fischetti. Embedding $\{0, \frac{1}{2}\}$-Cuts in a Branch-and-Cut Framework: A Computational Study. *INFORMS Journal on Computing*, 19(2):229–238, 2007.

[14] David L. Applegate, William J. Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35:693–699, 2007.

[15] Alper Atamtürk. Cover and Pack Inequalities for (Mixed) Integer Programming. *Annals of Operations Research*, 139(1):21–38, 2005.

[16] Alper Atamtürk and Oktay Günlük. Mingling: mixed-integer rounding with bounds. *Mathematical Programming*, 123(2):315–338, 2010.

[17] Egon Balas. Intersection Cuts - A New Type of Cutting Planes for Integer Programming. *Operations Research*, 19(1):19–39, 1971.

[18] Egon Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164, 1975.

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

[20] Egon Balas. A modified lift-and-project procedure. *Mathematical Programming*, 79(1–3):19–31, 1997.

[21] Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44, 1998.

[22] Egon Balas and Pierre Bonami. New variants of lift-and-project cut generation from the LP tableau: Open source implementation and testing. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Opimization*, volume 4513 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2007.

[23] Egon Balas and Pierre Bonami. Generating lift-and-project cuts from the LP simplex tableau: open source implementation and testing of new variants. *Mathematical Programming Computation*, 1(2–3):165–199, 2009.

[24] 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.

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

[26] Egon Balas, Sebastián Ceria, Gérard Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[27] Egon Balas and Robert G. Jeroslow. Strengthening cuts for mixed integer programs. *European Journal of Operational Research*, 4(4):224–234, 1980.

[28] Egon Balas and François Margot. Generalized Intersection Cuts. Technical report, Carnegie Mellon University, 2010.

[29] Egon Balas and Michael Perregaard. Lift-and-project for mixed 0-1 programming: recent progress. *Discrete Applied Mathematics*, 123(1):129–154, 2002.

[30] 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(2–3):221–245, 2003.

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

[32] Amitabh Basu, Pierre Bonami, Gérard Cornuéjols, and François Margot. Experiments with two-row cuts from degenerate tableaux. Technical report, 2009.

[33] Amitabh Basu, Pierre Bonami, Gérard Cornuéjols, and François Margot. On the relative strength of split, triangle and quadrilateral cuts. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 1220–1229, 2009.

[34] Amitabh Basu, Manoel Campelo, Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. On Lifting Integer Variables in Minimal Inequalities. In Friedrich Eisenbrand and F. Bruce Shepherd, editors, *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2010.

[35] Amitabh Basu, Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Minimal Inequalities for an Infinite Relaxation of Integer Programs. *SIAM Journal on Discrete Mathematics*, 24(1):158–168, 2010.

[36] Amitabh Basu, Gérard Cornuéjols, and François Margot. Intersection Cuts with Infinite Split Rank. Technical report, 2010.

[37] 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, 1998. `http://www.caam.rice.edu/~bixby/miplib/miplib.html`.

[38] 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(1):37–41, 2007.

[39] Pierre Bonami and Gérard Cornuéjols. A note on the MIR closure. *Operations Research Letters*, 36(1):4–6, 2008.

[40] 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.

[41] Valentin Borozan and Gérard Cornuéjols. Minimal Valid Inequalities for Integer Constraints. *Mathematics of Operations Research*, 34(3):538–546, 2009.

[42] Alberto Caprara and Matteo Fischetti. $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts. *Mathematical Programming*, 74(3):221–235, 1996.

[43] Alberto Caprara, Matteo Fischetti, and Adam N. Letchford. On the separation of maximally violated mod- k cuts. *Mathematical Programming*, 87(1):37–56, 2000.

[44] Alberto Caprara and Adam N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming*, 94(2-3):279–294, 2003.

[45] Sebastián Ceria, Cécile Cordier, Hugues Marchand, and Laurence A. Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81(2):201–214, 1998.

[46] Sebastian Ceria, Gérard Cornuéjols, and Milind Dawande. Combining and strengthening Gomory cuts. In Egon Balas and Jens Clausen, editors, *Integer Programming and Combinatorial Optimization*, volume 920 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 1995.

[47] Sebastián Ceria and Gábor Pataki. Solving integer and disjunctive programs by lift and project. In R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 271–283. Springer, 1998.

[48] Sebastián Ceria and João Soares. Disjunctive cut generation for mixed 0-1 programs: duality and lifting. Technical report, Columbia University, 1997.

[49] Der-San Chen and Stanley Zionts. Comparison of some algorithms for solving the group theoretic integer programming problem. *Operations Research*, 24(6):1120–1128, 1976.

[50] Thomas Christof and Gerhard Reinelt. Algorithmic aspects of using small instance relaxations in parallel branch-and-cut. *Algorithmica*, 30(4):597–629, 2001.

[51] Philipp M. Christophel. *Separation algorithms for cutting planes based on mixed integer row relaxations.* PhD thesis, Universität Paderborn, 2009.

[52] Vašek Chvátal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.

[53] Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 306(10-11):886–904, 2006.

[54] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. A Geometric Perspective on Lifting. Technical report, Tepper School Of Business, Carnegie Mellon University, 2009. To appear in Operations Research.

[55] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Corner Polyhedron and Intersection Cuts. Technical report, 2010.

[56] William J. Cook, Ricardo Fukasawa, and Marcos Goycoolea. Effectiveness of different cut selection rules. presented at the workshop on Mixed Integer Programming, Miami, Florida, 2006.

[57] William J. Cook, Ravi Kannan, and Alexander Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47(1–3):155–174, 1990.

[58] Gérard Cornuéjols. Revival of the Gomory cuts in the 1990's. *Annals of Operations Research*, 149(1):63–66, 2007.

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

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

[61] Gérard Cornuéjols, Yanjun Li, and Dieter Vandenbussche. K-cuts: A variation of Gomory mixed integer cuts from the LP tableau. *INFORMS Journal on Computing*, 15(4):385–396, 2003.

[62] Gérard Cornuéjols, Leo Liberti, and Giacomo Nannicini. Improved strategies for branching on general disjunctions. Technical report, Tepper School Of

Business, Carnegie Mellon University, 2008. To appear in Mathematical Programming.

[63] Gérard Cornuéjols and François Margot. On the facets of mixed integer programs with two integer variables and two constraints. *Mathematical Programming*, 120(2):429–456, 2009.

[64] Gérard Cornuéjols and Giacomo Nannicini. Reduce-and-Split Revisited: Efficient Generation of Split Cuts for Mixed-Integer Linear Programs. Technical report, 2010.

[65] Harlan Crowder, Ellis L. Johnson, and Manfred W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[66] George Bernard Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley & Sons, 1951.

[67] George Bernard Dantzig. Discrete-Variable Extremum Problems. *Operations Research*, 5(2):266–277, 1957.

[68] George Bernard Dantzig. On the Significance of Solving Linear Programming Problems with Some Integer Variables. *Econometrica*, 28(1):30–44, 1960.

[69] Sanjeeb Dash and Marcos Goycoolea. A Heuristic to Generate Rank-1 GMI Cuts. Technical Report RC24874, IBM Research, 2009.

[70] Sanjeeb Dash, Marcos Goycoolea, and Oktay Günlük. Two-Step MIR Inequalities for Mixed Integer Programs. Technical Report RC23791, IBM Research, 2006. To appear in INFORMS Journal on Computing.

[71] Sanjeeb Dash and Oktay Günlük. Valid inequalities based on simple mixed-integer sets. *Mathematical Programming*, 105(1):29–53, 2006.

[72] Sanjeeb Dash and Oktay Günlük. On the strength of Gomory mixed-integer cuts as group cuts. *Mathematical Programming*, 115(2):387–407, 2008.

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

[74] Dash Optimization Inc. XPRESS - MP. `http://www.dashoptimization.com`.

[75] Santanu S. Dey, Andrea Lodi, Andrea Tramontani, and Laurence A. Wolsey. Experiments with Two Row Tableau Cuts. In Friedrich Eisenbrand and F. Shepherd, editors, *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 2010.

[76] Santanu S. Dey and Quentin Louveaux. Split rank of triangle and quadrilateral inequalities. Technical Report 2009/55, CORE, Université Catholique de Louvain, Belgium, 2009.

[77] Santanu S. Dey and Andrea Tramontani. Recent Developments in Multiple Row Cuts. *Optima – Mathematical Programming Society Newsletter*, 80:2–8, 2009.

[78] Santanu S. Dey and Laurence A. Wolsey. Lifting Integer Variables in Minimal Inequalities Corresponding to Lattice-Free Triangles. In Andrea Lodi, A. Panconesi, and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, volume 5035 of *Lecture Notes in Computer Science*, pages 463–475. Springer, 2008.

[79] Santanu S. Dey and Laurence A. Wolsey. Two row mixed integer cuts via lifting. Technical Report 2008/30, CORE, Université Catholique de Louvain, Belgium, 2008.

[80] Santanu S. Dey and Laurence A. Wolsey. Constrained Infinite Group Relaxations of MIPs. Technical Report 2009/60, CORE, Université Catholique de Louvain, Belgium, 2009.

[81] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91::201–213, 2002.

[82] Friedrich Eisenbrand. On the Membership Problem for the Elementary Closure of a Polyhedron. *Combinatorica*, 19:297–300, 1999.

[83] Daniel G. Espinoza. Computing with Multi-row Gomory Cuts. In Andrea Lodi, A. Panconesi, and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, volume 5035 of *Lecture Notes in Computer Science*, pages 214–224. Springer, 2008.

[84] Daniel G. Espinoza. Computing with multi-row Gomory cuts. *Operations Research Letters*, 38(2):115–120, 2010.

[85] Michael C. Ferris, Gábor Pataki, and Stefan Schmieta. Solving the seymour problem. *Optima*, 66:2–6, 2001.

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

[87] Matteo Fischetti, Andrea Lodi, and Andrea Tramontani. On the separation of disjunctive cuts. Technical report, University of Padova, 2008.

[88] Matteo Fischetti and Domenico Salvagnin. A Relax-and-Cut Framework for Gomory's Mixed-Integer Cuts. Technical report, University of Padova, 2010.

[89] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury, 2nd edition, 2002.

[90] Swantje Friedrich. *Algorithmische Verbesserungen für die Lösung diskreter Optimierungsmodelle.* PhD thesis, Freie Universität Berlin, 2007.

[91] Ricardo Fukasawa and Marcos Goycoolea. On the exact separation of mixed integer knapsack cuts. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, volume 4513 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2007.

[92] Ricardo Fukasawa and Oktay Günlük. Strengthening lattice-free cuts using non-negativity. Technical Report RC24798, IBM Research, 2010. To appear in Discrete Optimization.

[93] Bernd Gärtner. Exact arithmetic at low cost - A case study in linear programming. *Computational Geometry*, 13(2):121–139, 1999.

[94] Claudio Gentile, Paolo Ventura, and Robert Weismantel. Mod-2 Cuts Generation Yields the Convex Hull of Bounded Integer Feasible Sets. *SIAM Journal on Discrete Mathematics*, 20(4):913–919, 2006.

[95] Fred Glover. Integer programming over a finite additive group. *SIAM Journal on Control*, 7(2):213–231, 1969.

[96] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[97] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[98] Ralph E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation, 1960.

[99] Ralph E. Gomory. An algorithm for integer solutions to linear programming. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[100] Ralph E. Gomory. On the relation between integer and noninteger solutions to linear programs. *Proceedings of the National Academy of Sciences of the United States of America*, 53(2):260–265, 1965.

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

[102] Ralph E. Gomory and Ellis L. Johnson. Some continuous functions related to corner polyhedra. *Mathematical Programming*, 3(1):23–85, 1972.

[103] Ralph E. Gomory and Ellis L. Johnson. Some continuous functions related to corner polyhedra II. *Mathematical Programming*, 3(1):359–389, 1972.

[104] Ralph E. Gomory and Ellis L. Johnson. T-space and cutting planes. *Mathematical Programming*, 96(2):341–375, 2003.

[105] G. Anthony Gorry, William D. Northup, and Jeremy F. Shapiro. Computational experience with a group theoretic integer programming algorithm. *Mathematical Programming*, 4(1):171–192, 1973.

[106] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1993.

[107] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted Cover Inequalities for 0-1 Integer Programs: Computation. *INFORMS Journal on Computing*, 10(4):427–437, 1998.

[108] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted Cover Inequalities for 0-1 Integer Programs: Complexity. *INFORMS Journal on Computing*, 11(1):117–123, 1999.

[109] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85(3):439–467, 1999.

[110] Christelle Guéret, Christian Prins, Marc Sevaux, and Susanne Heipcke. *Applications of Optimization with XpressMP*. Dash Optimization Ltd., 2002.

[111] Peter L. Hammer, Ellis L. Johnson, and U. N. Peled. Facet of regular 0-1 polytopes. *Mathematical Programming*, 8(1):179–206, 1975.

[112] Qie He, Shabbir Ahmed, and George L. Nemhauser. A probabilistic comparison of split and type 1 triangle cuts for two row mixed-integer programs. Technical report, Georgia Institute of Technology, 2010.

[113] Karla L. Hoffman and Manfred W. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

[114] Te C. Hu. On the asymptotic integer algorithm. *Linear Algebra and its Applications*, 3(3):279–294, 1970.

[115] ILOG Inc. Cplex, 2010. `http://www.ibm.com/software/integration/optimization/cplex/`.

[116] Ellis L. Johnson. On the group problem for mixed integer programming. *Mathematical Programming Study*, 2:137–179, 1974.

[117] Miroslav Karamanov and Gérard Cornuéjols. Branching on general disjunctions. Technical report, Tepper School of Business, Carnegie Mellon University, 2005. To appear in Mathematical Programming.

[118] Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1096, 1979.

[119] Kiavash Kianfar and Yahya Fathi. Generalized mixed integer rounding inequalities: facets for infinite group polyhedra. *Mathematical Programming*, 120(2):313–346, 2009.

[120] Diego Klabjan, George L. Nemhauser, and C. Tovey. The complexity of cover inequality separation. *Operations Research Letters*, 23(1-2):35 – 40, 1998.

[121] Natalia Kliewer, Taïeb Mellouli, and Leena Suhl. A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627, 2006.

[122] Donald E. Knuth. *The Art of Computer Programming, Volume 4*. Addison-Wesley, 2009.

[123] Achim Koberstein. *The Dual Simplex Method, Techniques for a fast and stable implementation*. PhD thesis, Universität Paderborn, 2005.

[124] Achim Koberstein. Progress in the dual simplex method for solving large scale LP problems: techniques for a fast and stable implementation. *Computational Optimization and Applications*, 41(2):185–204, 2008.

[125] Achim Koberstein and Uwe H. Suhl. Progress in the dual simplex method for large scale LP problems: practical dual phase 1 algorithms. *Computational Optimization and Applications*, 37(1):49–65, 2007.

[126] Arie M. C. A. Koster, Adrian Zymolka, and Manuel Kutschka. Algorithms to Separate $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts. *Algorithmica*, 55(2):375–391, 2009.

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

[128] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[129] Adam N. Letchford and Andrea Lodi. Strengthening Chvátal-Gomory cuts and Gomory fractional cuts. *Operations Research Letters*, 30(2):74–82, 2002.

[130] Jeffrey T. Linderoth. *Topics in Parallel Integer Optimization*. PhD thesis, Georgia Institute of Technology, 1998.

[131] Jeffrey T. Linderoth and Theodore K. Ralphs. Noncommercial Software for Mixed-Integer Linear Programming. In John K. Karlof, editor, *Integer Programming: Theory and Practice*, Operations Research Series, pages 253–303. CRC Press, 2005.

[132] Jeffrey T. Linderoth and Martin W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[133] László Lovász. Geometry of numbers and integer programming. In Iri Masao and Kunio Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 177–210. Springer, 1989.

[134] Ashutosh Mahajan and Theodore K. Ralphs. Experiments with Branching using General Disjunctions. In John W. Chinneck, Bjarni Kristjansson, and Matthew J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, volume 47 of *Operations Research/Computer Science Interfaces Series*, pages 101–118. Springer, 2009.

[135] Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001.

[136] François Margot. Testing cut generators for mixed-integer linear programming. *Mathematical Programming Computation*, 1(1):69–95, 2009.

[137] Csaba Mészáros and Uwe H. Suhl. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum*, 25(4):575–595, 2003.

[138] Hans Mittelmann. Decision tree for optimization software: Benchmarks for optimization software, 2010. `http://plato.asu.edu/bench.html`.

[139] MOPS Optimierungssysteme GmbH & Co. KG. MOPS–Mathematical Optimization System. `http://www.mops-optimizer.com`.

[140] Bruce A. Murtagh. *Advanced Linear Programming: Computation and Practice*. McGraw-Hill, 1981.

[141] George L. Nemhauser and Michael A. Trick. Scheduling A Major College Basketball Conference. *Operations Research*, 46(1):1–8, 1998.

[142] George L. Nemhauser and Laurence A. Wolsey. A recursive procedure to generate all cuts for 0–1 mixed integer programs. *Mathematical Programming*, 46(1–3):379–390, 1990.

[143] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series In Discrete Mathematics And Optimization. John Wiley & Sons Inc., 2nd edition, 1999.

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

[145] Manfred W. Padberg. A Note on Zero-One Programming. *Operations Research*, 23(4):833–837, 1975.

[146] Manfred W. Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[147] Manfred W. Padberg, Tony J. van Roy, and Laurence A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.

[148] Michael Perregaard. *Generating Disjunctive Cuts for Mixed Integer Programs*. PhD thesis, Graduate School of Industrial Administration, Carnegie Mellon University, 2003.

[149] Michael Perregaard and Egon Balas. Generating cuts from multiple-term disjunctions. In K. Aardal and B. Gerards, editors, *Integer Programming and Combinatorial Optimization*, volume 2081 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2001.

[150] Jean-Philippe P. Richard and Santanu S. Dey. The Group-Theoretic Approach in Mixed Integer Programming. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, chapter 19, pages 727–801. Springer, 2010.

[151] Barkley Rosser. A note on the linear diophantine equation. *The American Mathematical Monthly*, 48(10):662–666, 1941.

[152] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1986.

[153] Jeremy F. Shapiro. Dynamic programming algorithms for the integer programming problem - I: The integer programming problem viewed as a knapsack type problem. *Operations Research*, 16(1):103–121, 1968.

[154] Jeremy F. Shapiro. Group theoretic algorithms for the integer programming problem - II: Extension to a general algorithm. *Operations Research*, 16(5):928–947, 1968.

[155] Ingmar Steinzen, Achim Koberstein, and Uwe H. Suhl. Ein Entscheidungsunterstützungssystem zur Zuschnittoptimierung von Rollenstahl. In Leena Suhl and Stefan Voß, editors, *Quantitative Methoden in ERP und SCM, DSOR Beiträge zur Wirtschaftsinformatik*, pages 126–143. BoD, 2004.

[156] Leena M. Suhl and Uwe H. Suhl. A fast LU update for linear programming. *Annals of Operations Research*, 43(1):33–47, 1993.

[157] Uwe H. Suhl. MOPS - Mathematical OPtimization System. *European Journal of Operational Research*, 72:312–322, 1994.

[158] Uwe H. Suhl. MOPS - Mathematical OPtimization System. *OR News*, 8:11–16, 2000.

[159] Uwe H. Suhl. IT-gestützte, operative Sortimentsplanung. In B. Jahnke and F. Wall, editors, *IT-gestützte betriebliche Entscheidungsprozesse*, pages 175–194. Gabler, 2001.

[160] 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.

[161] Uwe H. Suhl and Leena M. Suhl. Solving Airline Fleet Scheduling Problems with Mixed-Integer Programming. In T. Ciriani, S. Gliozzi, Ellis L. Johnson, and R. Tadei, editors, *Operational Research in Industry*, pages 135–156. MacMillan Press Ltd., 1999.

[162] Uwe H. Suhl and Ralf Szymanski. Supernode processing of mixed-integer models. *Computational Optimization and Applications*, 3(4):317–331, 1994.

[163] Uwe H. Suhl and Veronika Waue. Fortschritte bei der Lösung gemischt-ganzzahliger Optimierungsmodelle. In Leena Suhl and Stefan Voß, editors, *Quantitative Methoden in ERP und SCM, DSOR Beiträge zur Wirtschaftsinformatik*, pages 35–53. BoD, 2004.

[164] John A. Tomlin and James S. Welch. Finding duplicate rows in a linear programming model. *Operations Research Letters*, 5(1):7–11, 1986.

[165] Tony J. van Roy and Laurence A. Wolsey. Valid inequalities for mixed 0-1 programs. *Discrete Applied Mathematics*, 14:199–213, 1986.

[166] Tony J. van Roy and Laurence A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.

[167] Veronika Waue. *Entwicklung von Software zur Lösung von gemischt-ganzzahligen Optimierungsmodellen mit einem Branch-and-Cut-Ansatz.* PhD thesis, Freie Universität Berlin, 2007.

[168] Robert Weismantel. On the 0/1 knapsack polytope. *Mathematical Programming*, 77(3):49–68, 1997.

[169] Franz Wesselmann. Strengthening Gomory Mixed-Integer Cuts: A Computational Study. Technical report, Universität Paderborn, 2009.

[170] Franz Wesselmann, Achim Koberstein, and Uwe H. Suhl. Pivot-and-Reduce Cuts: An Approach for Improving Gomory Mixed-Integer Cuts. Technical report, Universität Paderborn, 2008.

[171] Franz Wesselmann, Achim Koberstein, and Uwe H. Suhl. Strengthening Gomory Mixed-Integer Cuts. In Bernhard Fleischmann, Karl-Heinz Borgwardt, Robert Klein, and Axel Tuma, editors, *Annual International Conference of the German Operations Research Society (GOR)*, pages 487–492. Springer, 2008.

[172] Laurence A. Wolsey. Faces for a linear inequality in 0-1 variables. *Mathematical Programming*, 8(1):165–178, 1975.

[173] Laurence A. Wolsey. Facets and Strong Valid Inequalities for Integer Programs. *Operations Research*, 24(2):367–372, 1976.

[174] Laurence A. Wolsey. Valid inequalities for 0-1 knapsacks and MIPs with generalized upper bound constraints. *Discrete Applied Mathematics*, 29(2–3):251–162, 1990.

[175] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience Series In Discrete Mathematics And Optimization. John Wiley & Sons Inc., 1998.

[176] Giacomo Zambelli. On degenerate multi-row Gomory cuts. *Operations Research Letters*, 37(1):21–22, 2009.

[177] Arrigo Zanette, Matteo Fischetti, and Egon Balas. Can Pure Cutting Plane Algorithms Work? In Andrea Lodi, A. Panconesi, and G. Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, volume 5035 of *Lecture Notes in Computer Science*, pages 416–434. Springer, 2008.