Strengthening Gomory Mixed-Integer Cuts: A Computational Study

Franz Wesselmann
Decision Support & Operations Research Lab,
University of Paderborn, Warburger Str. 100,
33098 Paderborn, Germany
wesselmann@dsor.de

October 2009

Gomory mixed-integer cuts are an important ingredient in state-of-the-art software for solving mixed-integer linear programs. In particular, much attention has been paid to the strengthening of these cuts. In this paper, we give an overview of existing approaches for improving the performance of Gomory mixed-integer cuts. More precisely, we consider k-cuts, combined Gomory mixed-integer cuts, reduce-and-split cuts, and lift-and-project cuts. We give a detailed description of the implementation of the separation routines for these cutting planes. Finally, we report on computational results with the different strengthening approaches on a large-scale test set and analyze their performance. We also investigate the characteristics of the generated cutting planes. The results show that, although Gomory mixed-integer cuts are very effective, strengthening these cuts can have a positive impact on the performance of a MIP solver in many cases.

1 Introduction

In the last two decades the performance of software packages for solving mixed-integer programs (MIPs) increased dramatically. The progress becomes especially apparent by means of the standard library MIPLIB 3.0 [14] which contains MIPs from various practical applications. Most of these problems which were hard or even impossible to solve with standard software when MIPLIB 3.0 was assembled are now solvable within a few seconds. This improvement is due to the development of the state-of-the-art in related fields of research. Besides faster computers and improvements in linear programming, enhanced cutting plane techniques [15] make contemporary optimization software a powerful and practical tool which allows for solving large-scale, real-world MIPs.

Gomory mixed-integer (GMI) cuts [22] were proposed in the 1960s. Due to several reasons [18] these cuts were considered to be useless in practice for more than thirty years. In the course of the development of lift-and-project cuts [8, 9] they were reinvestigated and turned out to be very effective [10]. Nowadays GMI cuts are a crucial factor in state-of-the-art software for solving MIPs like Cplex [23] or MOPS [29]. Along with the GMI cuts other families of cutting planes like cover cuts, flow

cover cuts, flow path cuts and mixed integer rounding (MIR) cuts successfully found their way into these systems where they are generally used in conjunction with a branch-and-bound algorithm. In particular, improvements in the performance of GMI cuts are likely to bring about further substantial progress in solving hard MIPs.

In this paper we survey strengthening approaches for GMI cuts. We give a detailed description of their implementation in the MOPS MIP solver. Finally, we present computational results reflecting the effectiveness of our implementations and discuss the impact of certain details on the overall performance.

The MOPS MIP solver is a high performance system for solving large-scale LP and MIP problems. The current version of MOPS features state-of-the-art primal simplex, dual simplex and interior point algorithms to solve LP problems. Moreover, the system uses sophisticated heuristics, branch-and-bound (branch-and-cut) algorithms and cutting plane techniques to tackle MIPs.

Consider a mixed-integer linear program in the form

$$\begin{array}{lrcl} \min & c^T x & & \\ s.t. & Ax & \geq & b \\ & x & \geq & 0 \\ & x_j & \in & \mathbb{Z}, \quad j \in N_I \end{array} \qquad (\text{MIP}^{\geq})$$

where $c,\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^m,\ A \in \mathbb{R}^{m \times n}$ and $N_I \subseteq N = \{1, \ldots, n\}$. For technical reasons we assume that $Ax \geq b$ explicitly contains the simple upper bounds $x_j \leq d_j$ for $j \in N_I$ as constraints. In other words, the last $|N_I|$ inequalities of $Ax \geq b$ are $-x_j \geq -d_j$ for $j \in N_I$. We will discuss a more practical formulation in a subsequent section of this paper.

The linear programming relaxation (LP) of ($\text{MIP}^{\geq}$) is obtained by omitting the integrality conditions on x_j for all $j \in N_I$. Given a basis of (LP), let B index the basic and J index the non-basic variables. Furthermore, let x^* denote an optimal basic solution to (LP). A Gomory mixed-integer cut is generated from a simplex tableau row associated with basic integer-constrained variable which has a fractional value in the solution to (LP). Let such a row be given by

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} x_j. \qquad (1)$$

The GMI cut generated from the simplex tableau row (1) is

$$\begin{aligned} \sum_{j \in J \cap N_I : f_{ij} \leq f_{i0}} f_{ij} x_j + \sum_{j \in J \cap N_I : f_{ij} > f_{i0}} \frac{f_{i0}\,(1 - f_{ij})}{1 - f_{i0}} x_j \\ + \sum_{j \in J \setminus N_I : \bar{a}_{ij} \geq 0} \bar{a}_{ij} x_j + \sum_{j \in J \setminus N_I : \bar{a}_{ij} < 0} \frac{f_{i0}\,(-\bar{a}_{ij})}{1 - f_{i0}} x_j \geq f_{i0}, \end{aligned} \qquad (2)$$

where $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$ and $f_{i0} = \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor > 0$. This valid inequality can equivalently be obtained by applying mixed-integer rounding [24] to the tableau row (1).

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

$$vio\,(\alpha, \beta) = \beta - \alpha^T x^* = \beta \qquad (3)$$

as $\alpha^T x^* = 0$. Thus the violation is equal to the fractional part of the right-hand side of the corresponding simplex tableau row. Due to the fact that using violation

as a quality measure has several drawbacks, we also consider the Euclidean distance between the cut hyperplane $\alpha^T x = \beta$ and the solution x^* which is defined as

$$dis\,(\alpha, \beta) = \frac{\beta - \alpha^T x^*}{\|\alpha\|} = \frac{\beta}{\|\alpha\|}. \tag{4}$$

Observe that to enhance the Euclidean distance, one 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).

2 Strengthening Gomory Mixed-Integer Cuts

In the following section we discuss different strengthening techniques for GMI cuts. We start with the simple idea of modifying an LP tableau row by multiplying it by an integral value. Then we describe two approaches which create combinations of LP tableau rows. Finally, we consider the lift-and-project method which tries to improve the initial optimal LP tableau row by performing a sequence of pivots.

2.1 K-Cuts

Cornuéjols et al. [19] propose to multiply both sides of (1) by an integer $k \neq 1$ and then apply the Gomory mixed-integer cut, instead of generating a GMI cut from an (LP) tableau row directly. The resulting inequality has the form

$$\sum_{j \in J \cap N_I : f_{ij}^k \leq f_{i0}^k} f_{ij}^k x_j + \sum_{j \in J \cap N_I : f_{ij}^k > f_{i0}^k} \frac{f_{i0}^k \left(1 - f_{ij}^k\right)}{1 - f_{i0}^k} x_j + \sum_{j \in J \setminus N_I : k\bar{a}_{ij} \geq 0} k\bar{a}_{ij} x_j + \sum_{j \in J \setminus N_I : k\bar{a}_{ij} < 0} \frac{f_{i0}^k \left(-k\bar{a}_{ij}\right)}{1 - f_{i0}^k} x_j \geq f_{i0}^k, \tag{5}$$

where $f_{ij}^k = k\bar{a}_{ij} - \lfloor k\bar{a}_{ij} \rfloor$ and $f_{i0}^k = k\bar{a}_{i0} - \lfloor k\bar{a}_{i0} \rfloor$. The inequality (5) is called the k-cut generated from (1). For each k a variation of the original GMI cut is obtained as the fractional parts of the right-hand side and the coefficients in the tableau row change. The multiplication by k can produce a larger fractional part of the right-hand side and, therefore, lead to more violated inequalities. The size of the coefficients of the integral variables is not negatively influenced. In [19] it is proven that in the pure integer case k-cuts perform variable-wise better than the GMI cut with exactly fifty percent probability. On the other hand, in the mixed-integer case the coefficients of the continuous variables tend to deteriorate with increasing values of k.

2.2 Combined Gomory Mixed-Integer Cuts

Ceria et al. [16] discuss combining LP tableau rows with the objective of obtaining a strengthened GMI cut. The basic assumption of their approach is that all tableau rows are given in or can be transformed into rational data respectively. Consider two LP tableau rows exclusively written in rational numbers

$$x_i = \frac{e_{i0}}{D} - \sum_{j \in J} \frac{e_{ij}}{D} x_j \tag{6a}$$

$$x_k = \frac{e_{k0}}{D} - \sum_{j \in J} \frac{e_{kj}}{D} x_j \tag{6b}$$

where $e_{i0}, e_{i0}, D \in \mathbb{Z}$ and $e_{ij}, e_{kj} \in \mathbb{Z}$ for all $j \in J$. Multiplying (6a) by $\mu_i \in \mathbb{Z}$ and (6b) by $\mu_k \in \mathbb{Z}$, we obtain the following combined tableau row

$$\mu_i x_i + \mu_k x_k = \mu_i \frac{e_{i0}}{D} + \mu_k \frac{e_{k0}}{D} - \sum_{j \in J} \frac{\mu_i e_{ij} + \mu_k e_{kj}}{D} x_j. \tag{7}$$

We would like to choose the multipliers μ_i and μ_k such that the right-hand side of the GMI cut generated from the combination is maximized and the coefficients on the left-hand side are minimized. We start by optimizing the right-hand side. The right-hand side of the GMI cut generated from the combined row (7) will be

$$f_0 = \mu_i \frac{e_{i0}}{D} + \mu_k \frac{e_{k0}}{D} - \left\lfloor \mu_i \frac{e_{i0}}{D} + \mu_k \frac{e_{k0}}{D} \right\rfloor. \tag{8}$$

Ceria et al. [16] propose a method which allows for computing μ_i and μ_k such that a certain value of f_0 is generated. Suppose we would like to obtain the fractional part $f_0 = \frac{e}{D}$. Let G be the greatest common divisor of e_{i0}, e_{k0}, and D. Furthermore, suppose that G divides e. Then, the fractional part $f_0 = \frac{e}{D}$ can be obtained by setting

$$\mu_i = D - \frac{D-e}{G} p_i \quad \text{and} \quad \mu_k = D - \frac{D-e}{G} p_k \tag{9}$$

where p_i, p_k and q are integers which solve the diophantine equation

$$G = p_i e_{i0} + p_k e_{k0} + qD. \tag{10}$$

The requirement that G divides e must be made to ensure that μ_i and μ_k are integral. After maximizing the right-hand side, the same approach is used to minimize coefficients on the left-hand side while keeping the right-hand side maximal. Of course it is possible to consider more than two tableau rows at a time.

2.3 Reduce-and-Split Cuts

A different approach to strengthen the GMI cut was developed by Andersen et al. [2]. It is based on the observation that the coefficients of non-basic continuous variables in a GMI cut are not bounded and depend on the size of the coefficients in the corresponding simplex tableau row. The size of these coefficients has a direct influence on the cut quality (4). The idea described in [2] is to reduce the size of these coefficients by forming linear combinations of simplex tableau rows. Consider an additional row of the simplex tableau.

$$x_k = \bar{a}_{k0} - \sum_{j \in J} \bar{a}_{kj} x_j \tag{11}$$

To improve the GMI cut generated from (1), the latter row is combined with (11) by adding $\delta \in \mathbb{Z}$ times (11) to (1):

$$x_i + \delta x_k = \bar{a}_{i0} + \delta \bar{a}_{k0} - \sum_{j \in J \cap N_I} \left(\bar{a}_{ij} + \delta \bar{a}_{kj}\right) x_j - \sum_{j \in J \setminus N_I} \left(\bar{a}_{ij} + \delta \bar{a}_{kj}\right) x_j \tag{12}$$

As the procedure aims at reducing the coefficient of the variables $j \in J \setminus N_I$, one chooses δ such that it minimizes the function

$$h(\delta) = \sum_{j \in J \setminus N_I} \left(\bar{a}_{ij} + \delta \bar{a}_{kj}\right)^2. \tag{13}$$

In [2] it is shown that (13) is a quadratic convex function in δ and that its minimum can be found by rounding. If a reduction with $\delta \neq 0$ is identified, the tableau rows are combined and the process is iterated.

2.4 Lift-and-Project Cuts

Reconsider our MIP formulation (MIP$^{\geq}$) and suppose that all integer-constrained variables are 0-1 variables, i.e. we have a (mixed) 0-1 program. The whole constraint system will in the following also be denoted by $\hat{A}x \geq \hat{b}$ and can be written as $\hat{A}x - s = \hat{b}$ by introducing surplus variables. Note that the vector $s \in \mathbb{R}^{m+n}$ consists of m surplus variables from the constraints in A, including $|N_I|$ surplus variables from the upper bound constraints, and n surplus variables from the lower bound constraints. As all bounds on variables are contained in the constraint system $\hat{A}x \geq \hat{b}$, all structural variables are unrestricted and can w.l.o.g. be assumed to be basic. Due to the fact that $s_{m+j} = x_j$ for $j \in N$, it is possible to write the simplex tableau row (1) completely using only surplus variables.

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j. \tag{14}$$

The intersection cut [3] derived from (14) is given by $\alpha s \geq \beta$ where $\beta = \bar{a}_{i0}\,(1 - \bar{a}_{i0})$ and $\alpha_j = \max\{\bar{a}_{ij}\,(1 - \bar{a}_{i0})\,, -\bar{a}_{ij}\bar{a}_{i0}\}$ for $j \in J$. This cut can be strengthened by using the integrality conditions on the variables s_j for $j \in J \cap N_I$. The strengthened intersection cut is given by $\bar{\alpha} s \geq \beta$ where

$$\bar{\alpha}_j = \begin{cases} \min\{f_{ij}\,(1 - \bar{a}_{i0})\,, (1 - f_{ij})\,\bar{a}_{i0}\}\,, & \text{for } j \in J \cap N_I \\ \alpha_j, & \text{otherwise} \end{cases} \tag{15}$$

and $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$. The strengthened intersection cut is the same as the GMI cut.

Lift-and-project cuts [8, 9] are simple disjunctive [4] or split cuts [17] which are derived from a disjunction

$$\begin{pmatrix} \hat{A}x & \geq & \hat{b} \\ -x_i & \geq & 0 \end{pmatrix} \vee \begin{pmatrix} \hat{A}x & \geq & \hat{b} \\ x_i & \geq & 1 \end{pmatrix} \tag{16}$$

on a fractional 0-1 variable. Further developments of the method were documented in [5, 11, 28]. The most-violated (deepest) lift-and-project cut $\alpha x \geq \beta$ from this disjunction is obtained by solving a cut generating linear program (CGLP)

$$\begin{array}{llllllllll} \min & \alpha x^* & -\beta & & & & & & & \\ s.t. & & & & & & & & & \\ & \alpha & & -u\hat{A} & & & + & u_0 e_i & & = 0 \\ & \alpha & & & - & v\hat{A} & & & - \; v_0 e_i & = 0 \\ & & -\beta & +u\hat{b} & & & & & & = 0 \\ & & -\beta & & + & v\hat{b} & & & + \; v_0 & = 0 \end{array} \tag{CGLP$_i$}$$

where $u, v, u_0, v_0 \geq 0$ and e_i is the i-th unit vector. The set of feasible solutions to (CGLP$_i$) is a cone which needs to be truncated by a so-called normalization constraint

$$\sum_{i=1}^{m+n} u_i + \sum_{i=1}^{m+n} v_i + u_0 + v_0 = 1 \tag{17}$$

in order to obtain a bounded set.

Any basic solution $(\alpha, \beta, u, v, u_0, v_0)$ of (CGLP$_i$) corresponds to a lift-and-project cut which is given by $\beta = u\hat{b} + v\hat{b} + v_0$, $\alpha_i = \max\{u\hat{A}_i - u_i - u_0, v\hat{A}_i - v_i + v_0\}$ and $\alpha_j = \max\{u\hat{A}_j - u_j, v\hat{A}_j - v_j\}$ for $j \neq i$ where $\hat{A}_j$ is the j-th column of $\hat{A}$. Like the intersection cut, the lift-and-project cut can be strengthened by considering the

integrality of some variables. The strengthened lift-and-project cut $\bar{\alpha}x \geq \beta$ has the coefficients

$$\bar{\alpha}_j = \begin{cases} \min\{u\hat{A}_j - u_j + u_0 \lceil m_j \rceil, v\hat{A}_j - v_j - v_0 \lfloor m_j \rfloor\}, & \text{for } j \in N_I \setminus \{i\} \\ \alpha_j, & \text{otherwise} \end{cases} \tag{18}$$

where

$$m_j = \frac{v\hat{A}_j - v_j - u\hat{A}_j + u_j}{u_0 + v_0}. \tag{19}$$

A major drawback of the lift-and-project approach is that solving the (CGLP) may be time consuming if the underlying problem is sufficiently large.

2.4.1 Solving the CGLP on the LP Tableau

Balas and Perregaard [12] discovered a correspondence between bases of (LP) and (CGLP$_i$) which allows for a more efficient generation of lift-and-project cuts. Suppose that a partition (M_1, M_2) of J such that $j \in M_1$, if $\bar{a}_{ij} < 0$ and $j \in M_2$, if $\bar{a}_{ij} > 0$ is given. The strengthened lift-and-project cut which is defined by the solution of (CGLP$_i$) corresponding to the basis $(\alpha, \beta, u_0, v_0, \{u_k : k \in M_1\}, \{v_k : k \in M_2\})$ is equivalent to the strengthened intersection cut (or GMI cut) derived from (14). The same correspondence can also be established between the unstrengthened versions of these cuts. Based on these insights, Balas and Perregaard developed a very elegant method which mimics the optimization of (CGLP$_i$) by performing a sequence of pivots on the original (LP) tableau.

At each iteration of this procedure a pivot in a row $k \neq i$ of the simplex tableau is performed which produces a linear combination of the reference row i with row k such that the GMI cut from the combined row is more violated than the one solely obtained from row i. 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 row k of the (LP) simplex tableau corresponds to a pair u_k, v_k of non-basic variables of (CGLP$_i$). Pivoting u_k or v_k into the basis of (CGLP$_i$) causes the variable x_k to leave the basis of (LP). The reduced-cost of the non-basic variables u_k and v_k can be calculated from the entries in the (LP) tableau rows i and k and the (LP) solution x^* for each row $k \notin J \cup i$

$$r_{u_k} = -\sigma + \bar{a}_{k0}(1 - x_i^*) - \tau_k \tag{20a}$$
$$r_{v_k} = -\sigma - \bar{a}_{k0}(1 - x_i^*) + s_k^* + \tau_k \tag{20b}$$

where

$$\sigma = \frac{\sum_{j \in M_2} \bar{a}_{ij} s_j^* - \bar{a}_{i0}(1 - x_i^*)}{1 + \sum_{j \in J} |\bar{a}_{ij}|} \tag{21}$$

and

$$\tau_k = \sum_{j \in M_1} \sigma \bar{a}_{kj} + \sum_{j \in M_2} \left(s_j^* - \sigma\right) \bar{a}_{kj}. \tag{22}$$

In a second step, a non-basic variable x_p from row k is selected to enter the basis. The variable x_p which brings about the largest improvement of the cut violation among all other non-basic variables present in row k is pivoted into the basis. Two

evaluation functions are used to measure the effect of pivoting x_p into the basis.

$$g^+(\gamma) = \frac{\sum_{j\in J} \max\{\bar{a}_{ij}, -\gamma\bar{a}_{kj}\}\, s_j^* - \bar{a}_{i0} + (\bar{a}_{i0} + \gamma\bar{a}_{k0})\, x_i^*}{1 + \gamma + \sum_{j\in J} |\bar{a}_{ij} + \gamma\bar{a}_{kj}|} \tag{23a}$$

$$g^-(\gamma) = \frac{\sum_{j\in J} \max\{0, \bar{a}_{ij} + \gamma\bar{a}_{kj}\}\, s_j^* - (\bar{a}_{i0} + \gamma\bar{a}_{k0})\,(1 - x_i^*)}{1 - \gamma + \sum_{j\in J} |\bar{a}_{ij} + \gamma\bar{a}_{kj}|} \tag{23b}$$

2.4.2 Connection to Split Cuts

Multiplying the left-hand side of the disjunction (16) with $u, u_0 \geq 0$ and the right-hand side with $v, v_0 \geq 0$ we obtain

$$\left(u\hat{A}x - u_0 x_i \geq u\hat{b}\right) \vee \left(v\hat{A}x + v_0 x_i \geq v\hat{b} + v_0\right). \tag{24}$$

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 (simple split) disjunction $x_i \leq 0 \vee x_i \geq 1$. 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 some of the variables. This second operation can be seen as a strengthening of the underlying disjunction. Therefore the most-violated strengthened intersection cut (or GMI cut) is the result of a two-stage procedure. Ideally one would like 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 nonlinear program. Solving this mixed-integer nonlinear program has been treated in [13].

2.4.3 The Role of the Normalization

In a recent paper, Fischetti et al. [21] examine the strengths and weaknesses of the standard normalization constraint (17). There are two positive characteristics of this normalization. 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 (17) 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 depended on the scaling of the constraint system. Consider an inequality $a_i x \leq b_i$ and its scaled version $a_k x \leq b_k$ where $a_k = \mu a_i$ and $b_k = \mu b_i$ with $\mu > 1$. Clearly, the multipliers of the second inequality are $u_k = \frac{u_i}{\mu}$ and $v_k = \frac{v_i}{\mu}$. Therefore selecting the second inequality is more favorable as it consumes less 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. [21] propose the Euclidean normalization

$$\sum_{i=1}^{m+n} \|\hat{a}_i\| \, u_i + \sum_{i=1}^{m+n} \|\hat{a}_i\| \, v_i + u_0 + v_0 = 1 \tag{25}$$

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

3 Implementation

In this section we describe several technical details of our implementation. We created a framework which allows for efficient computation, handling, storage and combination of simplex tableau rows.

Unfortunately a mixed-integer program is generally not given in the form (MIP$^{\geq}$). Therefore consider a mixed-integer linear program in a very general form

$$\begin{array}{lrrl} \min & & c^T x & \\ s.t. & L \leq & \tilde{A}x & \leq D \\ & \tilde{l} \leq & x & \leq \tilde{d} \\ & & x_j & \in \mathbb{Z}, \quad j \in \tilde{N}_I \end{array} \tag{MIP}$$

where c, $x \in \mathbb{R}^{\tilde{n}}$, $A \in \mathbb{R}^{m \times \tilde{n}}$, $L, D \in (\mathbb{R} \cup \{-\infty, +\infty\})^m$, $\tilde{l}, \tilde{d} \in (\mathbb{R} \cup \{-\infty, +\infty\})^{\tilde{n}}$, $\tilde{N}_I \subseteq \tilde{N} = \{1, ..., \tilde{n}\}$ and $M = \{1, \ldots, m\}$. It consists of an objective function, constraints with ranges (L, D) and bounds ($\tilde{l}$, $\tilde{d}$) on the variables. All of the latter parts are linear. Moreover, some of the variables are constrained to take integral values.

An (MIP) can be restated in an alternative form by transforming all constraints (rows) of $\tilde{A}$ into equalities. In MIP solvers this is accomplished by a standardized procedure which adds a complete identity matrix to the constraint matrix $\tilde{A}$.

$$\begin{array}{lrrl} \min & & c^T x_S & \\ s.t. & & Ax & = 0 \\ & l \leq & x & \leq d \\ & & x_j & \in \mathbb{Z}, \quad j \in N_I \end{array} \tag{MIP$^{=}$}$$

where $A = \left[\tilde{A} \mid I\right] \in \mathbb{R}^{m \times n}$, $x = \begin{bmatrix} x_S \\ x_L \end{bmatrix} \in \mathbb{R}^n$, $l = \begin{bmatrix} \tilde{l} \\ -D \end{bmatrix} \in \mathbb{R}^n$, $d = \begin{bmatrix} \tilde{d} \\ -L \end{bmatrix} \in \mathbb{R}^n$ and $n = \tilde{n} + m$. The system (MIP$^{=}$) is also known as the internal model representation (IMR). 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*) . The ranges on the constraints are transformed into bounds on the logical variables.

State-of-the-art simplex engines do not explicitly work on the simplex tableau. The row of the simplex tableau corresponding to a basic variable x_i is obtained by multiplying the i-th row of the basis inverse with the constraint matrix A. Let the result of this multiplication be

$$x_i + \sum_{j \in J} \bar{a}'_{ij} x_j = 0 \tag{26}$$

where $i \in B$. Suppose that the set J is partitioned 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. By complementing the variables in J^d and shifting the variables in J^l we obtain

$$x_i = \bar{a}_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j, \tag{27}$$

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

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

3.1 K-Cuts

Integrating k-cuts into an existing separation routine for GMI cuts is quite easy. For the resulting cuts to remain valid, the values for k have to be integral. In our implementation, we generate k-cuts for $k = 2^i$, $i = 0, \ldots, 3$.

3.2 Combined Gomory Mixed-Integer Cuts

The combination procedure as proposed by Ceria et al. [16] has some major drawbacks. Suppose that we intend to combine a number of LP tableau rows and have computed multipliers such that the GMI cut generated from the combination has a maximal right-hand side. In the second part of the procedure, we now want to optimize the coefficients on the left-hand side. A correlation which has an important influence on this part of the algorithm is that the number of coefficients on the left-hand side that can be optimized is constrained by the number of involved tableau rows. Note that the LP tableau consists of m rows (basic variables) and $\tilde{n}$ (number of structurals) non-basic variables. In case that each row contains nonzero coefficients for all non-basic variables and $m \geq \tilde{n}$ optimizing all of the non-basic coefficients is possible but potentially computationally expensive whereas it would be generally impossible if $m < \tilde{n}$. In our implementation we use three tableau rows at a time.

The transformation of all entries of a simplex tableau into a rational representation can theoretically always be obtained by choosing a sufficiently large D. Unfortunately the size of the multipliers μ_j for $j = 1, \ldots, p$ is proportional to the size of D. Large multipliers can produce inaccuracies and lead to invalid cuts and will, in addition, bring about large coefficients on variables which are not optimized by the procedure. Therefore a resulting cut might have strong coefficients on a few variables and large, weak coefficients on the majority of the non-basic variables. In particular, remember that the coefficients of continuous non-basic variables in the GMI cut are proportional in size to the entries in the combined tableau row. Therefore in our implementation we set $D = 100$ and round the coefficients of the rows of the simplex tableau accordingly.

3.3 Reduce-and-Split Cuts

The reduce-and-split reduction algorithm is implemented as follows. In a first step, we compute and store tableau rows corresponding to basic integer variables. As these tableau rows are of the form (26), we do not have to consider right-hand sides in any part of the algorithm. Then we compute the quadratic Euclidean norm of all of these tableau rows. The reduce-and-split reduction algorithm iteratively replaces each row of the constructed matrix by a new row which has smaller coefficients on the continuous variables by forming linear combinations of simplex tableau rows.

Observe that our implementation is a slight variation of the reduce-and-split cuts as described in [2]. After the simplex tableau has been reduced by forming the linear combinations, Andersen et al. strengthen the underlying split disjunction and then generate the intersection cut. In our implementation we directly generate the GMI cut from a reduced simplex tableau row.

3.4 Lift-and-Project Cuts

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 a row of an (LP) tableau row can exclusively be written in the space of the slack or surplus variables respectively. Specifically, the theory assumes that all structural variables are unrestricted. However, in state-of-the-art optimization software bounds on variables are stored separately from the constraint matrix. Suppose that the problem is given in the form ($\text{MIP}^=$). Reconsider the transformed row of the simplex tableau

$$x_k = \bar{a}_{k0} - \sum_{j \in J} \bar{a}_{kj} s_j, \tag{28}$$

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 structural variable x_k which is bounded, we can again use the corresponding slack or surplus variables from the bound constraints.

$$s_k^l = (\bar{a}_{k0} - l_k) - \sum_{j \in J} \bar{a}_{kj} s_j \tag{29a}$$

$$s_k^d = (d_k - \bar{a}_{k0}) - \sum_{j \in J} (-\bar{a}_{kj})\, s_j \tag{29b}$$

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 (29a). 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 (29b).

3.4.1 Computing the Reduced Cost

As we have the choice to either pivot x_k to its lower bound l_k or to its upper bound d_k, we have to slightly modify the computation of the reduced cost:

$$r_{u_k}^l = -\sigma + (\bar{a}_{k0} - l_k)\,(1 - x_i^*) - \tau_k \tag{30a}$$

$$r_{v_k}^l = -\sigma - (\bar{a}_{k0} - l_k)\,(1 - x_i^*) + (x_k^* - l_k) + \tau_k \tag{30b}$$

$$r_{u_k}^d = -\sigma + (d_k - \bar{a}_{k0})\,(1 - x_i^*) + \tau_k \tag{30c}$$

$$r_{v_k}^d = -\sigma - (d_k - \bar{a}_{k0})\,(1 - x_i^*) + (d_k - x_k^*) - \tau_k \tag{30d}$$

To efficiently separate lift-and-project cuts in the framework of Balas and Perregaard a specialized simplex code is necessary. A point which is of crucial importance is a fast computation of the reduced cost, particularly the values τ_k. It is possible to obtain τ_k by recomputing the k-th (LP) tableau row and then perform the multiplication based on the partitioning of the entries. As this approach involves computing a large number of tableau rows, it is computationally very expensive.

But, as the multipliers are equal for each tableau row, we can rewrite the expression for τ_k in the form

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

where $A_{B_k}^{-1}$ is the k-th row of the basis inverse, $\left(A_J\right)_j$ is the j-th non-basic column of A, $y_j = \sigma$ for $j \in M_1$ and $y_j = s_j^* - \sigma$ 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

$$A_B \tau = \sum_{j\in J} \left(A_J\right)_j y_j \tag{31}$$

for τ. Fortunately, solving this system is a standard operation in state-of-the-art simplex engines. Specifically, it involves one ftran (forward transformation) operation, which is carried out in highly efficient manor.

3.4.2 Finding the Incoming Variable

As elaborated above, the variable which enters the basis is identified by finding the minimum of the functions $g^+(\gamma)$ and $g^-(\gamma)$. A procedure for finding the minima of these functions is described by Perregaard [27]. An important detail is that it is not necessary to identify the minima of both functions. 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. Using the bounds on x_k, we can rewrite the corresponding row of the simplex tableau in the form (29a). The pivot then has the following effect on (27):

$$x_i = \bar{a}_{i0} + \gamma_p \left(\bar{a}_{k0} - l_k\right) - \left(\sum_{j\in J} \left(\bar{a}_{ij} + \gamma_p \bar{a}_{kj}\right) s_j + \gamma_p s_k^l\right) \tag{32}$$

Now, observe that $\gamma_p = -\frac{\bar{a}_{ip}}{\bar{a}_{kp}}$ is the coefficient of the variable that was just pivoted out of the basis in the new combined simplex tableau row associated with x_i. This is also true in case we pivot x_k to its upper bound. Moreover, if we update the partition (M_1, M_2) of the non-basic indexes, the sign of γ_p will determine whether s_k^l (or x_k respectively) 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 $g^+(\gamma)$ or $g^-(\gamma)$ once the leaving variable has been selected.

3.4.3 Handling General Integer Variables

The procedure for generating lift-and-project cuts as outlined above is designed to exclusively deal with 0-1 variables. Only cuts for fractional 0-1 variables are generated, i.e. x_i with $0 < \bar{a}_{i0} < 1$. An (LP) tableau row which belongs to a basic general-integer variable has the form (14) where $l_i \leq \bar{a}_{i0} \leq d_i$. By rewriting this

row as

$$\begin{aligned} x_i - \lfloor \bar{a}_{i0} \rfloor &= (\bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor) - \sum_{j \in J} \bar{a}_{ij} s_j \\ &= f_{i0} - \sum_{j \in J} \bar{a}_{ij} s_j \end{aligned} \tag{33}$$

we obtain a form that the method can handle as $0 < f_{i0} < 1$. Basically, we use the disjunction

$$(-x_i \geq - \lfloor \bar{a}_{i0} \rfloor) \vee (x_i \geq \lceil \bar{a}_{i0} \rceil) \,. \tag{34}$$

3.4.4 Disjunctive Modularization

In order to approximate the generation of an optimal split cut, Balas and Bonami [6] iteratively strengthen the considered (LP) tableau row after each pivot by disjunctive modularization. More precisely, after each pivot the reference row of the (LP) tableau (14) is replaced by the row

$$y_i = \bar{a}_{i0} - \sum_{j \in J} \bar{\varphi}_{ij} s_j \tag{35}$$

where y_i is a new unrestricted integer-constrained variable and

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

with $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$. It is easy to see that the intersection cut derived from (35) is the strengthened intersection cut (or GMI cut) derived from (14).

3.4.5 Scaling

Fischetti et al. [21] show that the coefficient matrix of an MIP can be ill-conditioned in the sense that the scaling of certain constraints prevents the lift-and-project algorithm from producing cuts that are as strong as possible. As a solution they propose the Euclidean normalization (25) which is equivalent to a scaling of the coefficient matrix.

Balas and Bonami [7] take up the ideas of Fischetti et al. and study the normalization constraint

$$\sum_{i=1}^{m+n} \lambda_i \, (u_i + v_i) + u_0 + v_0 = \lambda_0, \tag{36}$$

where $\lambda_i \geq 0$ for $i = 1, \ldots, m+n$ and λ_0 is a positive integer. Introducing the new normalization (36) into (CGLP_i) is easy. Moreover, the correspondence between bases of (LP) and (CGLP_i) is not affected by this 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 (20) and the evaluation functions (23) is based on the assumption that the standard normalization $ve + ue + v_0 + u_0 = 1$ is used. The adapted formulae for the new normalization can be found in [7]. In our implementation, we set the values λ_i as shown in (25).

3.4.6 Additional Considerations

For lift-and-project cuts to be competitive with plain Gomory mixed-integer 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 is performed for each basic integral variable that is fractional. After pivoting based on the reduced cost of (CGLP), one has to restore the initial optimal (LP) basis. Of course, this can be done by running the dual simplex algorithm of MOPS. Although only a few pivots of the dual simplex may be necessary to return to (LP) optimality, it has to be executed m (number of rows) times in the worst case. 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 need to refactorize the basis. We found that this approach is considerably faster, especially when lift-and-project cuts are generated for a large number of basic variables. As previously discussed the calculation of the reduced cost of (CGLP) is a critical part of the lift-and-project algorithm. Moreover, once the reduced cost 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 indexes of the (LP) tableau rows corresponding to basic variables which have negative reduced costs.

4 Computational Experiments

In this section we report on the computational experience we made with the previously discussed approaches to strengthen GMI cuts. The computations were carried out on the Mittelmann MIP collection [26] and MIPLIB 3.0 [14]. We removed six instances from the test set due to the fact that they caused numerical issues[1] or we could not obtain the IP optimum.[2] Furthermore, we do not consider instances where none of the tested configurations was able to generate violated cutting planes.[3] In these cases, cuts were rejected due to density or other numerical reasons. The remaining test set consists of 102 instances. All of the test runs were conducted on an Intel Core 2 Duo PC with 2.40 GHz and 8 GB of main memory.

We add ten rounds of cuts at the root node (cut-and-branch). Cutting planes are generated for the fifty most fractional integer-constrained variables. For all variants of the lift-and-project algorithm we test the limit on the number of pivots is ten. The generated cuts are stored in a cut pool which selects the best cuts. Cut quality is mainly measured by the Euclidean distance between the cut hyperplane and the (LP) solution. We use the Markowitz criterion [25] to reject cuts that are very dense as these cuts are likely to slow down the LU factorization. To reduce the variability between test runs and improve the comparability of the results, we provide the optimal value of the objective function as a cut-off value. For each instance in the test set we report the percentage of integrality gap closed at the root node, the number of branching nodes, the computation time (in seconds) and the number of cutting planes added to the LP relaxation. We impose a time limit of one hour per instance. Note that we only use the cutting planes discussed in this paper. All of the other cutting planes (e.g. MIR cuts) are deactivated. Detailed results obtained with the different separation algorithms are presented in the tables A.1, A.2, A.3, A.4 A.5, A.6 and A.7 in [1].

We frequently use performance profiles [20] for a more comprehensive view on the performance of algorithms. Suppose algorithms $t \in T$, test instances $s \in S$ and a performance measure $q_{t,s}$ which quantifies how algorithm t performs on instance s (e.g. running times) are given. Performance profiles evaluate algorithms

[1] `neos4, qap10, neos823206`

[2] `neos15, neos16, neos19`

[3] `bc1, dano3_3, dano3_4, dano3_5, m20-75-1, m20-75-2, m20-75-3, m20-75-4, m20-75-5, mas74, mas76, misc07, neos13, neos17, neos5, neos671048, pk1, prod2`

by computing ratios $q_{t,s}/q_{t_s^*,s}$ for all $t \in T$ and $s \in S$ where t_s^* is the algorithm that performs best on instance s.

4.1 Solution Times

In the following we examine the performance of the discussed approaches with regard to solution times. For this analysis, we exclude all easy instances from the test set. These instances can be solved in five seconds or less by any of the tested configurations. Similarly, we do not consider instances that can not be solved by any of the tested configurations within one hour of computation time.

In a first experiment, we compare the standard GMI cuts with k-cuts, combined GMI cuts, reduce-and-split (R&S) cuts, and lift-and-project (L&P) cuts. Table 1

separator	geometric mean of solution times on instances solved with GMI cuts in t seconds				
	$t \leq 10$	$10 < t \leq 600$	$600 < t \leq 1800$	$1800 < t \leq 3600$	$3600 < t$
GMI cuts	3.50	110.62	1213.12	2593.62	3600.00
K-cuts	3.77	148.82	1567.21	2155.31	3600.00
R&S cuts	2.97	156.46	1568.37	1815.20	3600.00
Comb. GMI cuts	6.10	122.80	1456.33	1933.14	3600.00
L&P cuts	3.77	128.96	1205.45	1204.20	3298.75
# instances	11	32	5	5	4

Table 1: Comparison with GMI cuts

shows geometric means of the solution times needed by the different variants. A first observation that can be made is that GMI cuts are hard to beat in general. On the other hand, the strengthening approaches can help to reduce the solution times on instances that are difficult to solve using only GMI cuts. Concerning instances which can be solved with GMI cuts in between half an hour and an hour of computation time, table 1 reveals that separating L&P cuts can significantly speed up the solution process. Nevertheless, GMI cuts are superior on the easier instances.

Figure 1 shows performance profiles of GMI cuts and the four strengthening approaches on the test set based on solution times. On about 35% of the instances GMI cuts are faster than the other four algorithms. Furthermore, GMI cuts are able to solve about 90% of the instances in the reduced test set to optimality within the time limit. Combined GMI cuts outperform the other algorithms on 40% of the instances. As expected, these instances are mainly pure integer programs (IPs). R&S cuts and k-cuts are only successful on a few instances. In addition figure 1 again shows that L&P cuts outperform GMI cuts on some instances. Considering the interval in that the time factor takes values between about 5 and 40, the performance profile of the L&P cuts lies above all other profiles. This means that the L&P cuts are solving the most instances in this interval. On the other hand, GMI cuts are often faster than L&P cuts on relatively easy instances where the additional computational work performed by the L&P algorithm does not pay off. To support the latter statement also consider table A.5 in [1] which shows the performance of different strengthening techniques on instances that can not be solved by GMI cuts within 300 seconds of computation time. On these instances L&P cuts are really competitive to GMI cuts, i.e. the geometric mean of the solution times with L&P cuts is 877.49 seconds while it is 1197.55 seconds with GMI cuts.

Table 2 is an excerpt of the tables A.1 and A.2 (cf. [1]) and shows the best cases where one of the strengthening methods is more than 10% faster than the GMI cuts. The boldfaced numbers highlight the minimal value per instance. For example, observe that combined GMI cuts are superior on the integer programs

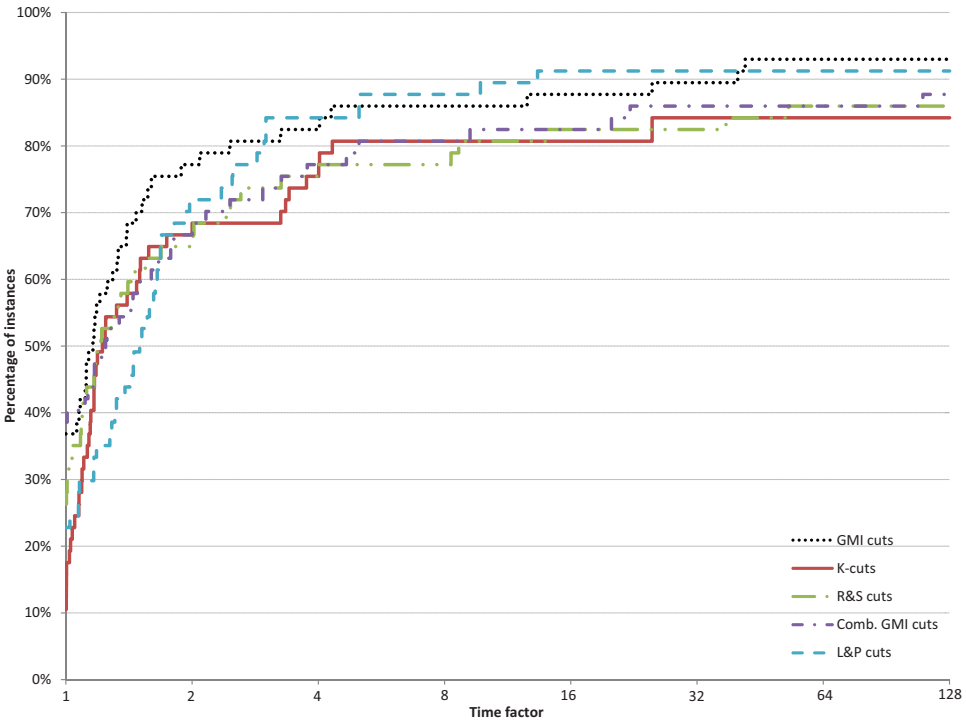

Figure 1: Performance profiles of solution times

30_05_100, harp2, and neos897005 as well as on the mixed 0-1 programs ran13x13 and swath2. Also note that the instance gesa2_o is only solved with L&P cuts within the time limit. Moreover R&S cuts are clearly superior on neos2, neos3 and ran12x21.

In the following we experiment with different enhancements of the lift-and-project algorithm. We test three additional variants: the L&P algorithm with disjunctive modularization (L&P DM), the L&P algorithm with the Euclidean normalization (L&P SC) and a variant that uses both techniques (L&P DM-SC). Table A.2 in [1] provides detailed results obtained with these variants. Note that we only consider the standard L&P algorithm and the three variants here. Concerning these variants, L&P DM and L&P SC are the fastest on about 35% and 28% of the instances in the test set respectively whereas the standard L&P cuts are fastest on 22%. The combined algorithm L&P DM-SC wins on about 37% of the instances. L&P SC solves the most instances to optimality of all L&P variants. Table 3 again shows the best cases where one variant outperforms the standard L&P cuts by 10% or more of computation time. Again the boldfaced numbers highlight the minimal running time per instance. The variant L&P DM can improve upon the results on the standard L&P cuts. For example, the instances neos648910 and ran12x21 solve faster using disjunctive modularization. The variant L&P SC is even faster than L&P DM on the instances gesa2_o, neos20 and ran10x26 among others. Furthermore, it is worth mentioning that the instance acc5 is only solved by the variants which use the Euclidean normalization (25).

As elaborated above, there is no clear winner among the enhanced L&P versions, i.e. no particular version is best on all instances. Table A.2 in [1] shows that the average running time over the whole test set of all L&P versions is quite comparable. The average running time obtained with the standard L&P cuts is 1081.55 seconds.

instance	solution times (in seconds)				
	GMI cuts	K-cuts	R&S cuts	Comb. GMI cuts	L&P cuts
10teams	4.8	5.4	4.8	79.8	**3.6**
30_05_100	468.6	471.6	904.8	**109.2**	1455.6
30_95_98	77.4	75.0	77.4	**65.4**	129.0
air04	56.4	61.8	57.0	**46.8**	61.2
bell3a	4.2	4.2	4.2	**3.6**	4.2
bell5	375.0	> 3600.0	126.0	82.8	**9.0**
bienst1	97.8	98.4	**83.4**	120.6	136.2
gesa2	582.0	586.8	**501.6**	583.2	584.4
gesa2_o	> 3600.0	> 3600.0	> 3600.0	> 3600.0	**2538.0**
harp2	453.6	453.6	455.4	**324.0**	628.8
modglob	2265.6	> 3600.0	> 3600.0	> 3600.0	**1086.6**
neos2	151.8	508.2	**135.6**	245.4	225.6
neos20	157.8	**103.8**	151.8	306.0	258.6
neos22	281.4	225.0	175.8	**149.4**	252.6
neos3	2737.8	1874.4	**1746.0**	2584.8	2065.8
neos648910	841.2	842.4	> 3600.0	2424.6	**742.2**
neos897005	162.6	160.8	157.2	**129.6**	188.4
p0548	9.6	6.0	9.0	4.8	**0.2**
p2756	77.4	71.4	**69.6**	115.8	117.0
pp08a	91.8	91.8	1205.4	85.8	**22.8**
prod1	3367.2	3369.6	3375.6	**1035.6**	1878.6
ran10x26	136.2	**121.8**	126.6	148.2	184.8
ran12x21	1786.2	2229.6	**1281.6**	1605.6	1648.8
ran13x13	334.8	249.0	423.0	**209.4**	225.6
rout	433.2	434.4	378.0	635.4	**294.6**
seymour1	1675.8	2275.8	1681.2	1677.0	**681.0**
swath1	2099.4	568.2	258.0	778.2	**166.8**
swath2	1701.6	> 3600.0	1780.8	**1315.8**	> 3600.0
vpm1	6.0	6.0	**0.2**	1.2	1.2
vpm2	217.8	658.8	330.6	262.8	**164.4**

Table 2: Solution times - best cases

Only the variant using the Euclidean normalization is faster on average with 1060.49 seconds. On the hard instances the situation is the same (cf. table A.5 in [1]).

Finally, we performed experiments with combinations of the strengthening approaches (cf. tables A.3 and A.4 in [1]). With respect to solution times, the combinations of the different techniques do not lead to an improvement. The geometric mean of the solution times obtained with GMI cuts is 57.63 seconds. This is the smallest value among all tested versions.

4.2 Dual Bounds and Enumeration

In this subsection we study how the discussed cutting planes strengthen the formulation of the instances in the test set. To this end, we examine the amount of integrality gap closed at the root node and the number of branch-and-bound nodes. Whenever dual bounds (or integrality gaps) are considered, we do not include instances where there is no gap between the objective function value of the optimal solution to the LP and the IP (or MIP).

Similar to the previous subsection, we start by analyzing the amount of integrality gap closed by generating GMI cuts, combined GMI cuts, R&S cuts and L&P cuts. In contrast to the previous study on solution times, figure 2 reveals that there is a clear best performer with respect to the amount of integrality gap closed. Specifically, L&P cuts close the largest amount of integrality gap on about 58% of the instances. Furthermore, L&P cuts are actually able to close any integrality gap on about 86%. In comparison, GMI cuts close the largest amount on 26% of the test set and succeed in moving the dual bound at all on 84%. The average gap closed with GMI cuts is 37.80% while it is 42.72% with the standard L&P cuts.

instance	solution times (in seconds)			
	L&P	L&P DM	L&P SC	L&P DM-SC
30_05_100	1455.6	918.0	450.6	**292.8**
acc1	12.0	**9.6**	15.0	878.4
acc5	> 3600.0	> 3600.0	**408.6**	**408.6**
air05	133.2	**102.6**	114.6	105.6
bell3a	4.2	4.8	4.2	**3.6**
gesa2	584.4	581.4	360.6	**360.0**
gesa2_o	2538.0	2530.2	**1787.4**	1788.6
harp2	628.8	557.4	688.8	**487.8**
mod010	6.0	8.4	**4.2**	**4.2**
neos20	258.6	347.4	205.8	**72.0**
neos21	105.0	**73.2**	105.0	81.6
neos6	> 3600.0	**1383.0**	> 3600.0	> 3600.0
neos648910	742.2	**359.4**	2195.4	738.0
neos808444	60.6	> 3600.0	> 3600.0	**39.6**
neos897005	188.4	**112.2**	189.0	199.8
p2756	117.0	89.4	67.2	**48.6**
pp08acuts	27.6	**13.2**	21.6	16.2
prod1	1878.6	1872.6	**981.0**	> 3600.0
ran10x26	184.8	183.6	147.6	**109.2**
ran12x21	1648.8	**1062.6**	1340.4	2891.4
swath1	166.8	**90.6**	268.2	160.8
swath2	> 3600.0	> 3600.0	**1177.2**	> 3600.0
swath3	> 3600.0	**3046.8**	> 3600.0	> 3600.0
vpm1	1.2	1.2	**0.6**	1.2
vpm2	164.4	**71.4**	231.0	415.2

Table 3: Solution times with L&P variants - best cases

The increase in the dual bounds mostly results in a reduction of the number of branch-and-bound nodes. Figure 3 shows that the number of nodes obtained by separating L&P cuts is the smallest on 34% of the test set. The remaining strengthening approaches lead to the minimal number of nodes in between 16% (GMI cuts) and 19% (combined GMI cuts) of the instances in the test set. The geometric mean of the number of nodes using L&P cuts is 10092.33 nodes which is clearly smaller than the geometric mean yielded by GMI cuts (11980.15 nodes).

Concerning the enhancements to the L&P cuts, L&P DM closes the largest amount of integrality gap on 34%, L&P SC on 44% and L&P DM-SC on 41% of the instances in the test set. The standard L&P cuts are best on 32% of the instances. Note that the overall sum of the percentages is larger than hundred percent due to the fact that several algorithms close the same amount of integrality gap on a number of instances (e.g. `10teams`). The average integrality gap closed with the enhancements is consistently larger than the one closed without them (cf. table A.2 in [1]). This shows that the performance of L&P cuts can be improved by these techniques.

However, nothing is perfect. The L&P algorithm in its current version is heavily dependent on the formulation of the IP or MIP. To show this, we ran the L&P cut separator without applying the LP preprocessing of MOPS (cf. table A.4 in [1]). Consider the instances `gesa2` and `gesa2_o`. Without LP preprocessing, L&P cuts close 82.35% and 87.81% of the integrality gap for these instances respectively. When LP preprocessing is applied, L&P cuts only close 29.49% and 8.40% of the integrality gap respectively. This shows how changes in the configuration which seem to be irrelevant for other parts of the solver can lead to a drastic change in performance. Therefore there is still room for improvements concerning the robustness of the L&P cuts.

Combining GMI cuts with R&S cuts or combined GMI cuts increases the running times as additional time is spent in the separation routines. The geometric mean

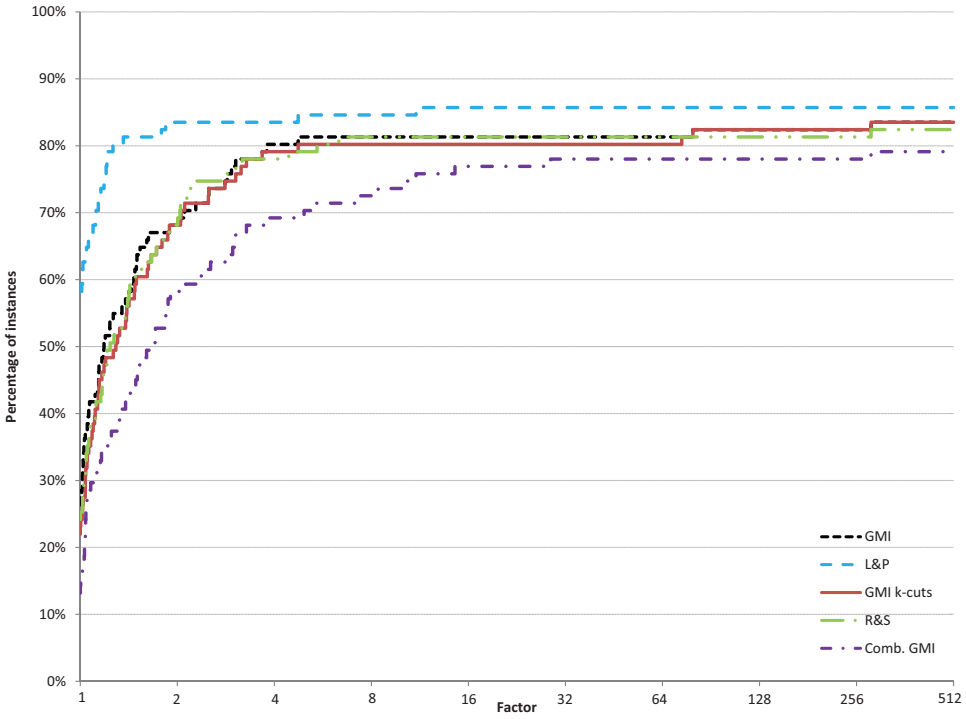

Figure 2: Performance profiles of integrality gaps closed

of the integrality gaps closed by the combination of GMI and combined GMI cuts is 36.81%. This is slightly smaller than the geometric mean of the gaps closed by GMI cuts alone, namely 37.80%. The number of branch-and-bound nodes processed by the combination is also larger. Separating GMI cuts together with R&S cuts does increase the average amount of integrality gap closed (cf. table A.3 in [1]). Furthermore, the number of nodes is also decreasing. The combination of all three cut separators does not improve upon the results of GMI cuts together with R&S cuts. The geometric means of the integrality gaps closed are equal. The number of cutting planes added to the LP relaxation is more or less the same in all these configurations.

The combination of L&P cuts with combined GMI cuts and R&S cuts consistently increases the amount of integrality gap closed. The geometric means are 42.72% with L&P cuts, 43.15% with L&P cuts and combined GMI cuts, 43.37% with L&P cuts and R&S cuts and 44.17% with all of the three aforementioned cut separators. The number of nodes is marginally increasing along with the amount of integrality gap closed. Although these improvements are not dramatic, two conclusions can be drawn. First, the other cut separators are obviously generating cutting planes that are not generated by the L&P algorithm. Secondly, although the L&P algorithm is the only exact approach (i.e. optimizing the (CGLP)), the formulation of an IP or MIP can be successfully strengthened by combining L&P cuts with heuristic approaches like R&S cuts. While the dual bounds are clearly better, the solution times are unfortunately increasing. Part of the increase is due to the additional computational work necessary for the separation of combined GMI cuts and R&S cuts. The number of added cutting planes is again quite comparable.

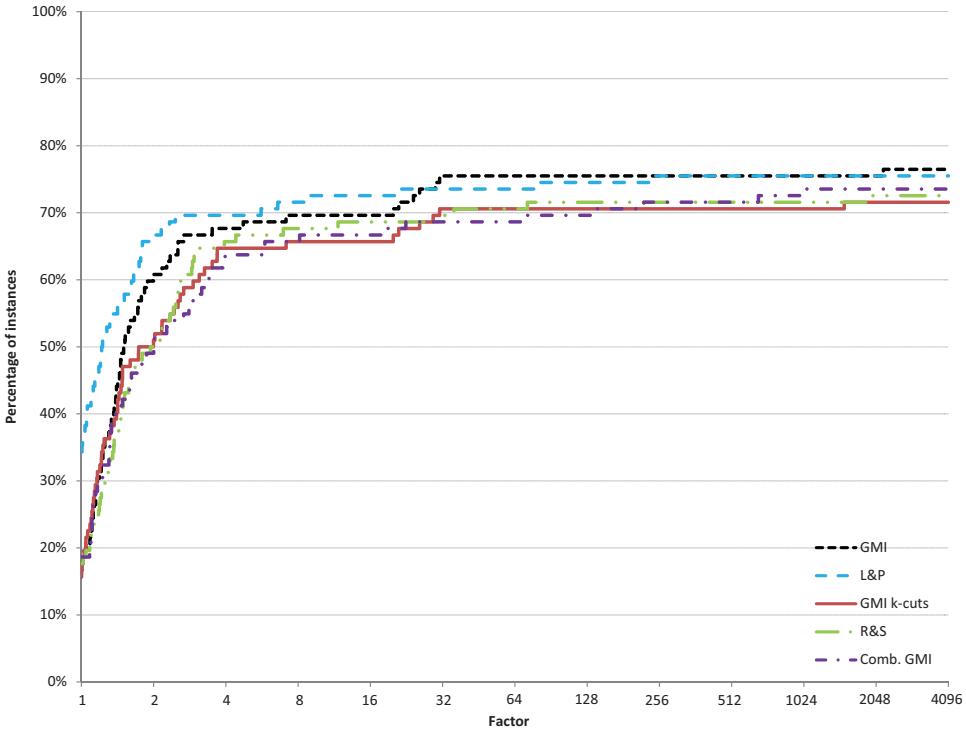

Figure 3: Performance profiles of number of nodes

4.3 Cut Characteristics

In this subsection, we investigate the cutting planes that are generated by the different separators. We proceed as follows. For each instance we compute the violation, the Euclidean distance, the number of non-zeros, the average coefficient on a continuous variable, and the average coefficient on an integer-constrained variable of the cuts generated from the initial optimal basis of the LP relaxation. Furthermore, we save the number of generated cutting planes. We do this for all separators. We then use these values to compute averages over all cutting planes generated by a specific separator.

Table A.6 (cf. [1]) shows the ratios between the averages calculated for GMI cuts and the strengthening techniques, i.e. k-cuts, R&S cuts, combined GMI cuts, and L&P cuts. K-cuts have very similar characteristics to GMI cuts. The geometric mean of the Euclidean distances is 3% larger than in the GMI cuts. Nevertheless, the separated k-cuts are in some instances significantly more violated than the GMI cuts. For example, consider the set-partitioning instance `air03` where the average k-cut is 28% more violated than the GMI cut.

As elaborated above, R&S cuts aim at increasing the Euclidean distance of a cut by reducing the Euclidean norm of the continuous coefficients. Table A.6 (cf. [1]) shows that R&S cuts often succeed in reaching this goal. In case of the instance `binkar10_1` the average size of the continuous variables in the R&S cuts is only about 22% of the average size of the continuous variables in the GMI cuts and the average Euclidean distance is about 91% larger than the one obtained using GMI cuts. Also consider the instance `ran13x13`. In general, the geometric mean of the Euclidean distances over all instances is 15% larger than in the GMI cuts. In addition R&S cuts are are not substantially denser than the GMI cuts. This is remarkable since the R&S reduction algorithm combines a lot of relatively dense

tableau rows and only takes care of the size of the continuous variables.

As expected and discussed before, the size of the coefficients on the continuous and integer-constrained variables in the combined GMI cuts is clearly larger than in the GMI cuts. For example, consider the instance `bienst2` where the average coefficient on a continuous variable is about 540 times larger than in the GMI cuts. Concerning all instances, the geometric mean of the average size of the continuous and integral variables is 3.39 and 5.45 times larger respectively. Moreover the combined GMI cuts contain 21% more non-zero elements than the GMI cuts on average. On the other hand, combined GMI cuts are successfully increasing the violation on a number of instances (e.g. `ran13x13`).

Concerning the improvement of cut quality, L&P cuts are very successful. Compared to GMI cuts, the average Euclidean distance of the L&P cuts is about 50% percent larger while the average violation is 3% larger. Observe that we compute the average violation of the final cuts instead of the final values of the objective function of (CGLP). Therefore the L&P cuts are not necessarily more violated than the GMI cuts. Although one could expect the pivot operations to make the cuts denser, this is not the case. The coefficients in the L&P cuts are also slightly smaller than in the GMI cuts. Finally, note that the L&P algorithm adds 14% more cutting planes on average. This is due to the increased quality of the generated cuts.

The results show that the L&P algorithm is the most successful with respect to the improvement of the cut quality. The resulting enhanced cuts allow for closing a larger portion of the integrality gap. Concerning the other separators, table A.6 in [1] again reveals that these only produce better cutting planes on a limited number of instances. Nevertheless, they are not useless.

In the following we conduct a similar analysis for the enhancements to the L&P algorithm. Table A.7 in [1] shows the ratios between the averages calculated for L&P cuts and the enhancements, i.e. L&P algorithm with disjunctive modularization, L&P algorithm with the Euclidean normalization and a variant that uses both ideas.

Disjunctive modularization produces cutting planes with almost the same characteristics as the standard L&P algorithm. Nevertheless, our computational experiments showed that using disjunctive modularization can reduce the running times significantly. On some instances the cut quality is also increased (e.g. `ran12x21`).

The variant that applies the Euclidean normlization produces cuts which are about 2% more violated and where the Euclidean distance is about 5% larger on average. For example, on the instance `prod1` the average Euclidean distance is about 29% larger than the one obtained separating standard L&P cuts. On the other hand, the cuts tend to be a bit denser and the size of the coefficients increases a little on average.

Using both techniques in combination yields cuts which are on average 2% more violated and where we measure 3% of additional Euclidean distance. In case of the instance `harp2` the cuts are on average 38% more violated than the standard L&P cuts. Also the Euclidean distance is larger by a factor of about two. Note that the combination of both techniques also outperforms the other L&P variants with respect to running times on this instance. In general, an additional observation is that the cuts are on average 50% denser than the standard L&P cuts. The geometric mean is 2% of additional density.

The previously discussed results show that disjunctive modularization and the Euclidean normlization can enhance the computational power of the L&P cuts. In many cases the cut quality measured by the violation or the Euclidean distance is increased. This, in turn, results in reduced integrality gaps or running times.

5 Conclusion

In this paper, we surveyed approaches to strengthen Gomory mixed-integer cuts. The implementation of all of these algorithms was described in detail. Then we conducted detailed computational experiments in the cut-and-branch framework of the MIP solver MOPS. We also analyzed the properties of the cutting planes that are created by the different methods. The results revealed that it is difficult to compete with GMI cuts based on solution times. On the other hand, we showed that the strengthened cuts are superior when integrality gaps are considered. We were also able to show that the combined Gomory cuts are successful on some instances. Moreover we showed that combinations of strengthening approaches can be effective.

References

[1] Detailed results, available at `http://dsor.uni-paderborn.de/fileadmin/materials/publications/results_gomory_cuts.pdf`.

[2] 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.

[3] Egon Balas. Intersection cuts - a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.

[4] Egon Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[5] Egon Balas. A modified lift-and-project procedure. *Mathematical Programming*, 79(1–3):19–31, 1997.

[6] 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.

[7] 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.

[8] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 01 programs. *Mathematical Programming*, 58(1–3):295–324, 1993.

[9] 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.

[10] Egon Balas, Sebastian Ceria, Gérard Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[11] Egon Balas and Michael Perregaard. Lift-and-project for mixed 0-1 programming: recent progress. *Discrete Applied Mathematics*, 123(1):129–154, 2002.

[12] Egon Balas and Michael Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. *Mathematical Programming Series B*, 94(2–3):221–245, 2003.

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

[14] 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.

[15] 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.

[16] 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.

[17] W. Cook, R. Kannan, and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47(1–3):155–174, 1990.

[18] Gérard Cornuéjols. Revival of the Gomory cuts in the 1990's. *Annals of Operations Research*, 149(1):63–66, 2007.

[19] 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.

[20] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming Series A*, 91::201–213, 2002.

[21] Matteo Fischetti, Andrea Lodi, and Andrea Tramontani. On the separation of disjunctive cuts. Technical report, University of Padova, 2008.

[22] Ralph E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation, 1960.

[23] ILOG Inc., An IBM Company. *CPLEX 10.2 Reference Manual*. ILOG, Gentilly Cedex, France, 2007.

[24] Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001.

[25] H.M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.

[26] Hans Mittelmann. Decision tree for optimization software: Benchmarks for optimization software, 2008.

[27] Michael Perregaard. *Generating Disjunctive Cuts for Mixed Integer Programs*. PhD thesis, Graduate School of Industrial Administration, Carnegie Mellon University, 2003.

[28] 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.

[29] Uwe H. Suhl. MOPS - Mathematical OPtimization System. *European Journal of Operational Research*, 72:312–322, 1994.