Arie M.C.A. Koster
Adrian Zymolka
Manuel Kutschka

Algorithms to Separate $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts

ZIB-Report 07–10 (May 2007)

Algorithms to Separate $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts

Arie M.C.A. Koster[*] Adrian Zymolka[§] Manuel Kutschka[¶]

Abstract

Chvátal-Gomory cuts are among the most well-known classes of cutting planes for general integer linear programs (ILPs). In case the constraint multipliers are either 0 or $\frac{1}{2}$, such cuts are known as $\{0, \frac{1}{2}\}$-cuts. It has been proven by Caprara and Fischetti [8] that separation of $\{0, \frac{1}{2}\}$-cuts is $\mathcal{NP}$-hard.

In this paper, we study ways to separate $\{0, \frac{1}{2}\}$-cuts effectively in practice. We propose a range of preprocessing rules to reduce the size of the separation problem. The core of the preprocessing builds a Gaussian elimination-like procedure. To separate the most violated $\{0, \frac{1}{2}\}$-cut, we formulate the (reduced) problem as integer linear program. Some simple heuristic separation routines complete the algorithmic framework.

Computational experiments on benchmark instances show that the combination of preprocessing with exact and/or heuristic separation is a very vital idea to generate strong generic cutting planes for integer linear programs and to reduce the overall computation times of state-of-the-art ILP-solvers.

1 Introduction

Each pure integer linear program (ILP) can be written in its standard minimization form

$$\begin{cases} \min & c^T x \\ s.t. & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{cases} \tag{1}$$

with integer matrix $A \in \mathbb{Z}^{m \times n}$, an integer right hand side $b \in \mathbb{Z}^m$, and arbitrary objective values $c \in \mathbb{R}^n$ (here m is the number of rows and n the number of columns of A). Objectives to be maximized can be rewritten as minimization problem by multiplying the coefficients by -1. Similarly, $\geq$ constraints are multiplied by -1 to obtain a $\leq$ constraint. Upper bound constraints for single variables are included in the coefficient matrix.

We assume without loss of generality that each row in A has relatively prime coefficients, since otherwise the row can be simplified by dividing all coefficients and the right hand side with the greatest common divisor among the coefficients (after division, a fractional right hand side can be rounded down). Associated with the program (1), we define the integer

[*]University of Warwick, Centre for Discrete Mathematics and its Applications (DIMAP), Coventry CV4 7AL, United Kingdom, E-mail: Arie.Koster@wbs.ac.uk

[§]atesio GmbH, Sophie-Taeuber-Arp-Weg 27, D-12205 Berlin, Germany. E-mail: zymolka@atesio.de

[¶]Zuse Institute Berlin (ZIB), Takustraße 7, D-14195 Berlin, Germany. E-mail: kutschka@zib.de

Keywords: $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts, separation algorithms, integer programming

Mathematics Subject Classification (2000): 90C10, 90C57, 65K05

solution set $X = \{x \in \mathbb{Z}^n \mid Ax \leq b, x \geq 0\}$, its convex hull polyhedron $P_{IP} = \operatorname{conv}(X)$ and the linear relaxation polyhedron $P_{LP} = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$.

Given a system $Ax \leq b$, a *Chvátal-Gomory* (CG) cut is defined by

$$\left\lfloor u^T A \right\rfloor x \leq \left\lfloor u^T b \right\rfloor \tag{2}$$

with $u \geq 0$. It is easy to show that *undominated* CG cuts have $u \in [0,1)^m$. By the integrality of $x \in X$, (2) is valid for P_{IP}. Gomory [16, 17] showed that if $P_{IP} \neq P_{LP}$, there exists for every fractional vertex $x^* \in P_{LP}$ a CG cut (2) that is violated, i.e., $\left\lfloor u^T A \right\rfloor x^* > \left\lfloor u^T b \right\rfloor$ [11]. In fact, by iteratively extending the system (1) with all possible CG cuts, we obtain the integral polyhedron after a finite number of steps (see Gomory [16, 17] for polytopes and Schrijver [22] for polyhedrons).

Caprara and Fischetti [8] introduced $\{0, \frac{1}{2}\}$-*cuts* for those CG cuts that are derived by $u \in \{0, \frac{1}{2}\}^m$. For several combinatorial optimization problems it is known that problem-specific classes of facet-defining inequalities are $\{0, \frac{1}{2}\}$-cuts with particular properties, e.g., the blossom inequalities of the matching polytope (describing this polytope completely) [12], (odd-valued) odd hole inequalities of the stable (multi-)set polytope [19, 20, 21], or the Möbius ladder inequalities of the linear ordering polytope [13]. Like Gomory showed for general CG cuts, it has recently been shown by Gentile et al. [15] that iteratively extending the system (1) by all possible $\{0, \frac{1}{2}\}$-cuts yields a complete description of the integer polytopes in a finite number of steps as well.

For $\{0, \frac{1}{2}\}$-cuts, we consider the following separation problem:

$\{0, \frac{1}{2}\}$-SEP
Given: The program (1) and a fractional solution $x^* \in P_{LP}$.
Find: A weight vector $u \in \{0, \frac{1}{2}\}^m$ such that $\left\lfloor u^T A \right\rfloor x^* > \left\lfloor u^T b \right\rfloor$ or a proof that none exists.

Theorem 1 (Caprara and Fischetti [8]) $\{0, \frac{1}{2}\}$-SEP *is $\mathcal{NP}$-complete.*

Consequently, Caprara and Fischetti [8] concentrate on polynomial-time solvable cases of $\{0, \frac{1}{2}\}$-SEP. In particular, they show that if A is an integer matrix with at most two odd coefficients per row, $\{0, \frac{1}{2}\}$-SEP is polynomial-time solvable. They propose therefore to weaken A to a matrix with the described property. In Andreello et al. [4], a computational study is presented to reveal the strength of this heuristic approach for $\{0, \frac{1}{2}\}$-SEP. They restrict cuts to have $\left\lfloor u^T A \right\rfloor = u^T A$, i.e., a rounding of the left hand side is avoided. Caprara and Fischetti [9] propose a number of reduction rules to limit the size of the separation problem.

Contribution. This paper reports on our study to separate general $\{0, \frac{1}{2}\}$-cuts effectively, despite its $\mathcal{NP}$-completeness. We recall that the 0 and $\frac{1}{2}$ coefficients of the vector u allow to reduce the size of $\{0, \frac{1}{2}\}$-SEP considerably by an extended set of preprocessing steps, ranging from obvious observations to a sophisticated procedure based on Gaussian elimination to eliminate rows and columns. After preprocessing, violated $\{0, \frac{1}{2}\}$-cuts can often be indicated directly as single rows of the reduced problem. Our computational experiments show that this is a very vital idea generating many violated $\{0, \frac{1}{2}\}$-cuts with small effort.

Independently from the preprocessing, an ILP is formulated to find the most violated $\{0, \frac{1}{2}\}$-cut. This auxiliary ILP can be solved either for the original separation problem or the reduced one. In a computational study we show that the exact separation can be sped up by a factor of at least 10 if preprocessing is performed first.

The effect of the separation of $\{0, \frac{1}{2}\}$-cuts on the performance of state-of-the-art ILP solvers is documented in a further computational study. It shows that by exact separation the number of branch&cut nodes is reduced by 20% on average at the cost of increased overall computation times due to the auxiliary ILP that has to be solved. Moreover, it is unclear whether the most violated $\{0, \frac{1}{2}\}$-cut is also the one that strengthens the formulation the most. Therefore, we additionally propose a heuristic routine (after preprocessing) to find violated $\{0, \frac{1}{2}\}$-cuts that are likely to strengthen the formulation. Computational experiments show that in such a way the overall computation times can be sped up by 20% for moderately sized instances.

Recently, Fischetti and Lodi [14] followed independently a similar integer programming approach as to optimize over the first Chvátal closure, i.e., the polytope derived by adding all inequalities (2) with $u \in [0, 1)^m$. In contrast to their approach, we can exploit the addressed preprocessing techniques for $\{0, \frac{1}{2}\}$-SEP as to reduce the problem size. By this, we can optimize more effectively over the first Chvátal closure in case the $\{0, \frac{1}{2}\}$-cuts are the only undominated CG cuts, e.g., for the matching polytope.

Outline. The rest of the paper is organized as follow. This section is completed with some further notation used in this paper. Section 2 is dedicated to preprocessing for $\{0, \frac{1}{2}\}$-SEP, whereas exact and heuristic separation algorithms for $\{0, \frac{1}{2}\}$-SEP are presented in Section 3. In Section 4 we report on the results of the computational studies on the effectiveness of the developed ideas and algorithms. The paper is closed with concluding remarks in Section 5.

Notation. Let e^j denote a unit vector of appropriate size with j-th coefficient equal to one, whereas $\mathbf{1}$ ($\mathbf{0}$) denotes the all one (zero) vector and $\mathbb{1}_I(i)$ the indicator function being 1 if $i \in I$ and 0 otherwise. With modulo applied component-wise, we define $\bar{A} = A \mod 2$ and $\bar{b} = b \mod 2$. Moreover, for a fractional solution $x^* \in P_{LP}$, we set $s = b - Ax^* \geq 0$ as slack vector. The violation of (2) for a vector $u \in \{0, \frac{1}{2}\}^m$ and fractional solution $x^* \in P_{LP}$ is denoted by $z(u, x^*) := \lfloor u^T A \rfloor x^* - \lfloor u^T b \rfloor$.

2 Preprocessing $\{0, \frac{1}{2}\}$-SEP

To find a separating $\{0, \frac{1}{2}\}$-cut, we seek for a weight vector u such that $z(u, x^*) > 0$. The next lemma restates this task.

Lemma 2 *Let $x^* \in P_{LP}$ be a fractional solution. There exists a vector $u \in \{0, \frac{1}{2}\}^m$ such that $z(u, x^*) > 0$ if and only if there exists a binary vector $v \in \{0, 1\}^m$ such that $v^T \bar{b}$ is odd and*

$$v^T s + (v^T \bar{A} \mod 2) x^* < 1 \tag{3}$$

holds.

Proof: The violation $z(u, x^*)$ can be rewritten as follows:

$$\begin{aligned} z(u, x^*) &= \lfloor u^T A \rfloor x^* - \lfloor u^T b \rfloor \\ &= \tfrac{1}{2}\left((2u)^T b \mod 2\right) - u^T s - \tfrac{1}{2}\left((2u)^T A \mod 2\right) x^* \\ &\overset{v:=2u}{=} \tfrac{1}{2}\left((v^T \bar{b} \mod 2) - v^T s - (v^T \bar{A} \mod 2) x^*\right) \end{aligned}$$

Since $\bar{A}, \bar{b}, s$, and $v = 2u$ are all non-negative, the only way to obtain a positive violation $z(u, x^*)$ consists in $v^T\bar{b} \mod 2 \equiv 1$ and $v^Ts + (v^T\bar{A} \mod 2)x^* < v^T\bar{b} \mod 2 \equiv 1$. □

Note that both conditions in Lemma 2 are independent of the actual values of coefficients and right hand sides, but take into account only their parities, i.e., whether they are even or odd. The vector v indicates the original inequalities to combine with weight $\frac{1}{2}$ such that the right hand side is in fact rounded down, and this strengthening (by $\frac{1}{2}$) is not compensated by the collected slacks together with the necessary rounding of the fractional left hand side coefficients.

In order to simplify the restated task, the system $(\bar{A}, \bar{b}, s)$ and x^* can be preprocessed by a series of transformations and problem size reductions, see also Caprara and Fischetti [9]. The following observations are helpful in this regard:

Lemma 3 *The reductions below do not influence the set of undominated $\{0, \frac{1}{2}\}$-cuts for the original system (and can be assumed to be carried out before for the follow ups):*

(i) All columns in $\bar{A}$ corresponding to variables $x_i^* = 0$ can be removed.

(ii) Zero rows in $(\bar{A}, \bar{b})$ can be removed.

(iii) Zero columns in $\bar{A}$ can be removed.

(iv) Identical columns in $\bar{A}$ can be replaced by a single representative with associated variable value as sum of the merged variables.

(v) Any unit vector column $\bar{a}^i = e^j$, $1 \le j \le m$, in $\bar{A}$ can be removed provided that x_i^* is added to the slack s_j of row j.

(vi) Any row $1 \le j \le m$ with slack $s_j \ge 1$ can be removed.

(vii) Rows identical in $(\bar{A}, \bar{b})$ can be eliminated except for one with smallest slack value.

Proof:

(i) Zero variable values do not contribute to the left hand side of (3).

(ii) Though originally $\bar{A}$ does not contain zero rows due to the assumption of relatively prime coefficients, after any (combination) of the other preprocessing steps, zero rows can appear in $(\bar{A}, \bar{b})$. Such rows have neither impact on the value $v^T\bar{b}$ nor on the value of $v^T\bar{A}$, only the total slack is increased. Hence, such rows can be left out of further consideration.

(iii) The corresponding variable has only even coefficients, hence left hand side rounding will never occur.

(iv) Either all or none variables of identical columns will have to be rounded on the left hand side.

(v) Whenever v indicates to include inequality j, the i-th variable's coefficient on the left hand side will have to be rounded down (after division by two) since the inequality holds the only odd coefficient for that variable. So, x_i^* adds to the left hand side of (3) whenever the slack s_j does.

(vi) Setting $v_j = 1$ where $s_j \ge 1$ violates directly condition (3).

(vii) Whenever an indicated cut involves one of the eliminated inequalities, the latter can be replaced in the generation by the associated one kept in the system, yielding a cut with at least the same violation value.

□

As a result, we obtain a reduced system which is equivalent for the separation. For notational convenience, we continue to use m and n for the (reduced) numbers of rows and columns, respectively. Moreover, we assume throughout the sequel that for any arising interim system, all of these reductions are applied as well.

So far, any row of the system $(\bar{A}, \bar{b}, s)$ represents a single original inequality. A further reduction in problem size can be obtained by row combinations according to rules specified below. For this, we associate with each row j of $(\bar{A}, \bar{b}, s)$ an index set R_j holding the indices of original inequalities currently combined for this row. These index sets are initialized by $R_j = \{j\}$.

We consider a basic operation performed on the rows of $(\bar{A}, \bar{b}, s)$: the addition of one row to another one, where the coefficients of $\bar{A}$ and $\bar{b}$ are added in modulo 2 arithmetic, the coefficients of s in normal arithmetic, and the symmetric difference is taken for the associated index sets. So, adding row i to row j gives a new row j with the following values:

$$\bar{a}_{jk} := \bar{a}_{ik} + \bar{a}_{jk} \mod 2 \;\forall k, \quad \bar{b}_j := \bar{b}_i + \bar{b}_j \mod 2, \quad s_j := s_i + s_j, and \quad R_j := R_i \triangle R_j,$$

where $X \triangle Y = (X \cup Y) \setminus (X \cap Y)$ for sets X, Y.

Using this operation, the system $(\bar{A}, \bar{b}, s)$ can be further transformed and might then allow for additional application of reduction rules from Lemma 3. Except for this, we are particularly interested in rows with zero coefficients and non-zero right hand side.

Lemma 4 *Let j be the index of a zero row in $\bar{A}$ with $\bar{b}_j = 1$. If $s_j < 1$, then the weight vector u defined by $u_i = \frac{1}{2}$ for all $i \in R_j$ and 0 otherwise, defines a violated $\{0, \frac{1}{2}\}$-cut on the original system (A, b, s)*

Proof: Let $v = e^j$. Then $v^T \bar{b} = 1$ and the left hand side of (3) equals $s_j < 1$, and thus by Lemma 2 a violated $\{0, \frac{1}{2}\}$-cut inequality is found. By construction, the index set R_j defines exactly the original inequalities to be combined. □

Notice that rows with slack zero have a special property: Adding such a row i twice to any other row j results in the original row j. Rows with slack zero play a key role in the next reduction rule.

Proposition 5 *Let i be the index of a row and k the index of a column of $\bar{A}$ such that $\bar{a}_{ik} = 1$ and $s_i = 0$. Then column k can be removed from $\bar{A}$ provided that row i is added to all other rows j with $\bar{a}_{jk} = 1$ and the slack of row i is set to $s_i := x^*_k$.*

Proof: Let i be the index of a row and k the index of a column of $\bar{A}$ such that $\bar{a}_{ik} = 1$ and $s_i = 0$. Further let $J := \{\, j : \bar{a}_{jk} = 1\} \setminus \{i\}$. Consider a cut which is generated by combining all rows with indices $j \in I \subseteq \{1, ...m\}$ of the system $(\bar{A}, \bar{b})$ such that w.l.o.g. $|I \cap J| = q$. W.l.o.g. we assume $R_j = \{j\}$ for all $j \in I$. This gives $R = I$ for the symmetric difference R

of all index sets R_j and for the violation $z(R,x^*)$ holds

$$\begin{aligned} z(R,x^*) &= \sum_{j\in I} s_j + \sum_{\ell} [(\sum_{j\in I} \bar{a}_{j\ell}) \bmod 2]\ x^*_\ell \\ &= \sum_{j\in I\setminus\{i\}} s_j + \sum_{\ell\neq k} [(\sum_{j\in I} \bar{a}_{j\ell}) \bmod 2]\ x^*_\ell + [(\sum_{j\in I\cap J} \bar{a}_{jk} + \mathbb{1}_I(i)\bar{a}_{ik}) \bmod 2]\ x^*_k \\ &= \sum_{j\in I\setminus\{i\}} s_j + \sum_{\ell\neq k} [(\sum_{j\in I} \bar{a}_{j\ell}) \bmod 2]\ x^*_\ell + [(q + \mathbb{1}_I(i)) \bmod 2]\ x^*_k \end{aligned}$$

Now consider the reduced system $(\bar{A}', \bar{b}')$ which results by adding the row i to all rows $j \in J$. Thus $\bar{a}'_{j\ell} = \bar{a}_{j\ell}$ if $j \in I \setminus J$ and $\bar{a}'_{j\ell} = \bar{a}_{j\ell} + \bar{a}_{i\ell}$ for $i \in I \cap J$. Let R'_j denote the updated index sets, s'_j the updated slack values and R' the associated symmetric difference for row set I'. The violation is $z(R',x^*) = \sum_{j\in I'\setminus(J\cup\{i\})} s'_j + \sum_{\ell\neq k} [(\sum_{j\in I'} \bar{a}'_{j\ell}) \bmod 2]\ x^*_\ell + \mathbb{1}_{I'}(i)\ s'_i$. In addition it follows that

$$\begin{aligned} \sum_{j\in I'} \bar{a}'_{j\ell} &= \sum_{j\in I'\setminus(J\cup\{i\})} \bar{a}'_{j\ell} + \sum_{j\in J\cap I'} \bar{a}'_{j\ell} + \mathbb{1}_{I'}(i)\bar{a}'_{i\ell} && (4) \\ &= \sum_{j\in I'\setminus(J\cup\{i\})} \bar{a}_{j\ell} + \sum_{j\in J\cap I'} \bar{a}_{j\ell} + \bar{a}_{i\ell} + \mathbb{1}_{I'}(i)\bar{a}_{i\ell} && (5) \\ &= \sum_{j\in I'\setminus\{i\}} \bar{a}_{j\ell} + (q + \mathbb{1}_{I'}(i))\bar{a}_{i\ell}. && (6) \end{aligned}$$

We now consider three cases: First, q is even. We set $I' = I$. Then the following holds:

$$z(R',x^*) = \sum_{j\in I\setminus\{i\}} s_j + \sum_{\ell\neq k} [(\sum_{j\in I} \bar{a}_{j\ell}) \bmod 2]\ x^*_\ell + \mathbb{1}_I(i)\ x^*_k = z(R,x^*)$$

and hence the same violated inequalities can be found.

Second, q is odd, $i \in I$. Then we set $I' = I \setminus \{i\}$ and consider the violation:

$$\begin{aligned} z(R',x^*) &= \sum_{j\in I\setminus\{i\}} s_j + \sum_{\ell\neq k} [(\sum_{j\in I'} \bar{a}'_{j\ell}) \bmod 2]\ x^*_\ell \\ &= \sum_{j\in I\setminus\{i\}} s_j + \sum_{\ell\neq k} [(\sum_{j\in I\setminus\{i\}} \bar{a}_{j\ell} + q\bar{a}_{il}) \bmod 2]\ x^*_\ell = z(R,x^*) \end{aligned}$$

Third, q is odd, $i \notin I$. Then we set $I' = I \cup \{i\}$. Then the following holds:

$$\begin{aligned} z(R',x^*) &= \sum_{j\in I} s_j + \sum_{\ell\neq k} [(\sum_{j\in I} \bar{a}'_{j\ell} + \bar{a}'_{il}) \bmod 2]\ x^*_\ell + s'_i \\ &= \sum_{j\in I\setminus\{i\}} s_j + \sum_{\ell\neq k} [(\sum_{j\in I} \bar{a}_{j\ell} + (q+1)\,\bar{a}_{il}) \bmod 2]\ x^*_\ell + x^*_k = z(R,x^*) \end{aligned}$$

For all cases we have shown that given an index set I of the system $(\bar{A}, \bar{b})$, we can select an index set I' of the reduced system $(\bar{A}', \bar{b}')$ that generates a cut with exactly the same violation, and vice-versa. □

In case a zero row in $\bar{A}$ is constructed by (repeated) application of Proposition 5, we either have a row with $\bar{b}_j = 0$ and Lemma 3 (ii) can be applied to remove the row as well, or $\bar{b}_j = 1$, and, by Lemma 4, the row describes a $\{0, \frac{1}{2}\}$-cut with violation $1 - s_j$. If in addition $s_j = 0$, the violation is maximal. By Lemma 2 on the other hand, a $\{0, \frac{1}{2}\}$-cut with maximal violation can be only combined from rows with slack zero and parity sum zero (modulo 2) for all columns k with $x^*_k > 0$. Since the above procedure can be applied as long as there are rows with slack zero, it provides a polynomial-time exact algorithm for maximally violated $\{0, \frac{1}{2}\}$-cuts. Caprara et al. [10] observed the same in the more general context of mod-k-cuts.

Further, a zero row j in $\bar{A}$ with $\bar{b}_j = 1$ and $s_j = 0$ can be very helpful in generating further violated $\{0, \frac{1}{2}\}$-cuts: Any other row i with $\bar{b}_i = 0$ and $s_i < 1$ can be turned into a violated $\{0, \frac{1}{2}\}$-cut by adding row j. This way, a zero row with right hand side 1 is generated, whereas the slack remains the same. Only in case $R_i \cap R_j = \emptyset$, the $\{0, \frac{1}{2}\}$-cut is dominated by the one identified by row j. Therefore if such a row j exists, the condition $u^T b = 1$ can be neglected in the search for further violated cuts.

Corollary 6 *Let i be the index of a row and k the index of a column of $\bar{A}$ such that $\bar{a}_{ik} = 1$, $s_i = 0$, and $x^*_k \geq 1$. Then both row i and column k can be removed from $\bar{A}$ provided that row i is added to all other rows j with $\bar{a}_{jk} = 1$.*

Proof: After application of Proposition 5, $s_i = x^*_k > 1$ and thus Lemma 3 (vi) can be applied to remove row i. □

The above holds in particular for tight upper bound constraints from the original system. If $\bar{b}_i = 0$, only the index sets R_j have to be updated. If $\bar{b}_i = 1$, $\bar{b}_j$ have to be adapted additionally.

Corollary 6 indicates that it is benefical to perform Proposition 5 on columns with large x^*_k. Altogether, the combination of Lemma 3 and Proposition 5 provides an algorithmic framework for preprocessing the system $(\bar{A}, \bar{b}, s)$ and generation of maximally violated $\{0, \frac{1}{2}\}$-cuts.

3 Separation algorithms

With or without preprocessing, the separation problem $\{0, \frac{1}{2}\}$-SEP can be described by a system $(\bar{A}, \bar{b}, s)$ and a fractional solution $x^* \in P_{LP}$. To find the most violated $\{0, \frac{1}{2}\}$-cut, we first formulate the problem as integer linear program. This ILP allows for the exact separation of $\{0, \frac{1}{2}\}$-cuts. Next, we present some ways to advance the solution of such an auxiliary ILP as well as an heuristic routine to find violated cuts fast.

3.1 Exact separation

The exact separation problem can be modeled by an auxiliary integer linear program which maximizes the violation. By Lemma 2, $\{0, \frac{1}{2}\}$-SEP can be restated as the search for a binary weight vector $v \in \{0, 1\}^m$ such that $v^T \bar{b} \mod 2 = 1$ and (3) is satisfied. These weights are used as binary variables v_i in the formulation.

Condition (3) requires to determine $u^T \bar{A} \mod 2 \in \{0, 1\}^n$. To this end, the variables $y_i \in \{0, 1\}$ for all $i = 1, \ldots, n$ are introduced to express whether the i-th variable's coefficient becomes odd in the indicated inequality sum or not. To model the modulo 2 computations, we further need auxiliary integer variables $r = (r_i)_{i=1,\ldots,n} \in \mathbb{Z}^n_+$ for all columns $i = 1, \ldots, n$, as well as an additional $q \in \mathbb{Z}_+$ for the right hand side. The separation problem then reads:

$$\begin{cases} \hat{z} = & \min \quad s^T v + (x^*)^T y \\ & s.t. \quad \bar{b}^T v - 2q = 1 \\ & \qquad \bar{A}^T v - 2r - y = \mathbf{0} \\ & \qquad v \in \{0,1\}^m \\ & \qquad y \in \{0,1\}^n \\ & \qquad r \in \mathbb{Z}_+^n \\ & \qquad q \in \mathbb{Z}_+ \end{cases} \tag{7}$$

The optimum value $\hat{z}$ of (7) indicates whether a violated cut has been found or not. If $\hat{z} \geq 1$, this is not the case. If $0 \leq \hat{z} < 1$, $1 - \hat{z}$ equals twice the violation $z(u, x^*)$ of the cut generated by combining the original inequalities which are obtained from symmetric difference of those sets R_j with $v_j = 1$ in the optimum solution.

To guide the search to highly violated $\{0, \frac{1}{2}\}$-cuts, we can add an inequality

$$s^T v + (x^*)^T y \leq 1 - \varepsilon \tag{8}$$

to (7) where $\varepsilon \in (0, 1]$. In this way only cuts with a violation of at least $\frac{1}{2}\varepsilon$ are found.

3.2 Heuristic search

After the above described reductions, the auxiliary ILP might nevertheless stay large for larger ILPs (1). Hence the search for violated $\{0, \frac{1}{2}\}$-cuts might still be time-consuming. As a faster alternative, we study combinatorial search heuristics which examine the reduced system $(\bar{A}, \bar{b})$ for violated cuts.

A simple approach is to enumerate all possible combinations of k or less rows of $(\bar{A}, \bar{b})$ that yield a violated $\{0, \frac{1}{2}\}$-cut with $0 < k \leq m$: First, we check if any single row of $(\bar{A}, \bar{b})$ results in a violated $\{0, \frac{1}{2}\}$-cut. If none of them is violated, we test all combinations of two rows for violation. This process is continued to combinations of k rows, if all combinations up to $k - 1$ rows are not violated or the number of detected violated cuts does not exceed a given limit.

4 Computational results

Framework. We implemented our preprocessing and separation algorithms as additional separator within the branch&cut framework SCIP v0.90 [1, 2] using CPLEX 10.01 [18] as underlying LP solver. All of SCIP's standard modules (e.g. separators, heuristics) are kept if not stated differently. SCIP's parameters are set to their default values except for a global time limit of 1 hour per instance and avoidance of restarts during solving.

If not stated differently, our separator is called only in the root node like SCIP's standard separators. To investigate the added value of $\{0, \frac{1}{2}\}$-cut separation more accurately our separator is called before SCIP's separators (Gomory, Strong Chvátal-Gomory, Complemented MIR [2]) and cut generating constraint handlers (knapsack, linear). At default, SCIP's separators and constraint handlers are called to separate cuts if and only if our separator does not find a violated cut anymore.

Instead of adding violated $\{0, \frac{1}{2}\}$-cuts directly to the LP, they are first stored in a pool from which only the best cuts are selected and added to the LP. We tested several methods to rate the cuts in the pool but we restrict to two methods in the following. The first one is to rate cuts by their violation (i.e., cuts with large violation are better than those with small violation). The second one is similar: Cuts are rated by non-increasing efficacy which is defined as its violation divided by the Euclidian norm of its coefficients (i.e., cuts are better the higher the "average" violation is). The best up to p cuts are transferred to SCIP (with p given as input parameter) which uses further criteria like the parallelism to the objective and other cuts to select the best among all violated inequalities found.

All computations are done on a computer with 3.6 Ghz CPU, 3.7 GB RAM and Linux as operating system. Our computational study includes all pure integer (i.e., non-mixed) instances from MIPLIB 3.0 [6] and MIPLIB 2003 [3] as well as the 2-matching-relaxations of TSP instances from the TSPLIB [5] that also have been studied in [14].

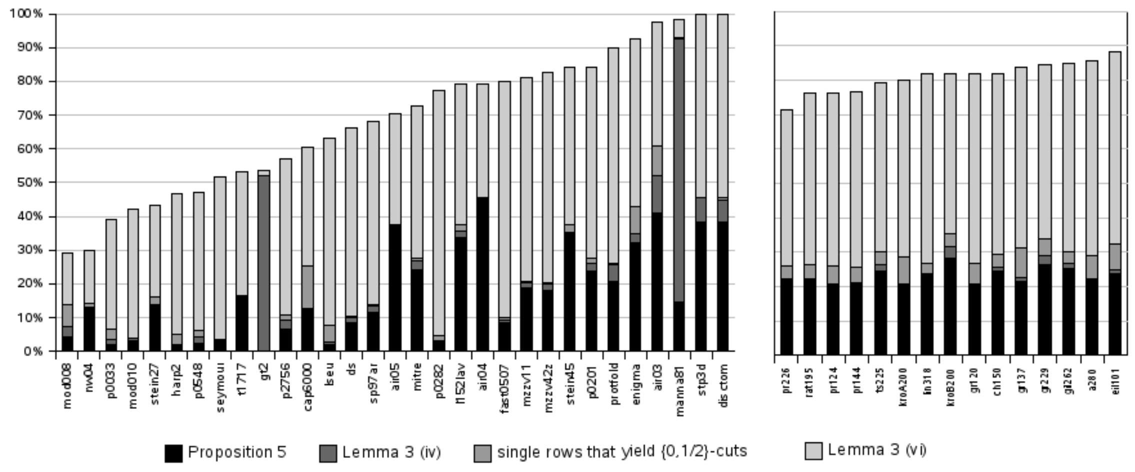
Speed-up by preprocessing. We implemented the preprocessing methods suggested in Section 2 in the following order: Removing columns whose corresponding variables (a) are zero in the current LP solution (Lemma 3 (i)) or (b) have a tight variable bound constraint, (c) removing rows with slack at least 1 (Lemma 3 (vi)), (d) removing columns by repeatedly applying Proposition 5, and (e) removing unit vector columns (Lemma 3 (v)). Next we check for empty rows of the preprocessed matrix with a nonzero right hand side (i.e., $\bar{b}_j = 1$). Such a row directly yields a $\{0, \frac{1}{2}\}$-cut. It is (f) deleted from the matrix and if the corresponding cut is violated with violation at least ε, it is added to the pool because every further combination of rows containing such a row cannot yield a stronger cut. Finally, (g) we erase identical rows except for one with the lowest slack value (Lemma 3 (vii)).

Whenever a zero column, a zero row, or a row with slack at least 1 results from a preprocessing step, it is removed from $(\bar{A}, \bar{b})$ immediately (e.g., zero rows that result from Proposition 5 are removed from $(\bar{A}, \bar{b})$ and yield a reduction in the number of rows).

To test the effect of preprocessing, we ran all instances with separation of $\{0, \frac{1}{2}\}$-cuts at all nodes of the branch&cut tree without a minimum violation (i.e., $\varepsilon = 0$). The steps (a)-(c) reduce the size of $\bar{A}$ significantly, e.g., considering the MIPLIB instances, on average 83.2% in number of rows (ranging from 46.96% (stein27) to 99.9% (nw04)) and can be applied without greater effort. Considering the 2-matching relaxations of the TSPLIB instances this reduction is even more effective, namely 99.5% of the size of $\bar{A}$ is eliminated (on average). Hence, these steps should certainly be applied and we focus on the further reductions. Figure 1 shows the effect of the steps (d), (e), (f) and (g), using the number of rows as a measure for the problem size. All reduction values are given relative to the number of rows of $\bar{A}$ after applying steps (a)-(c) and are averaged over all times they are applied within the branch&cut. Hence a value of 0% means that no further reduction beside the steps (a)-(c) can be achieved and a value of 100% corresponds to a reduction resulting in an empty pair $(\bar{A}, \bar{b})$. The instances are sorted according to non-decreasing total reduction. Figure 1 shows that on average a reduction of about 14.6% of the remaining size (after applying (a)-(c)) is achieved by applying Proposition 5 (step (d)), 1.26% by removing unit vector columns (step (e)), 1.5% by removing empty rows that yield a violated cut (see above) and finally 40.7% can be achieved by removing identical rows which arise from applying the previous preprocessing methods (in particular within step (d) and (e)).

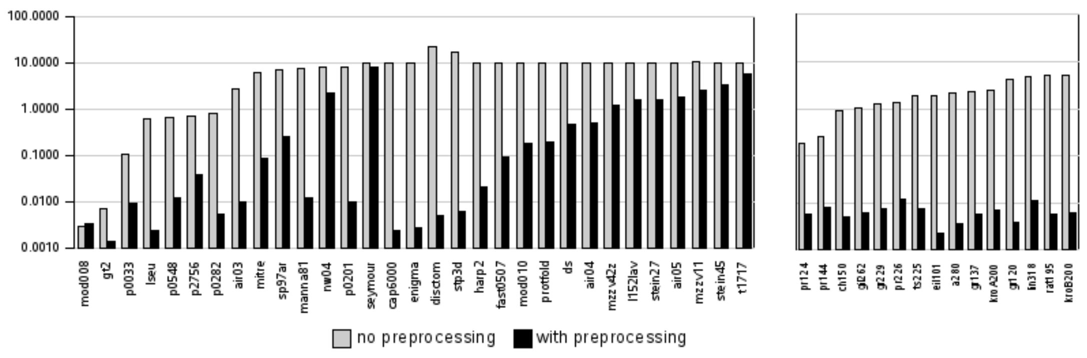
Moreover the total reduction in number of rows of $(\bar{A}, \bar{b})$ (including the steps (a)-(c)) is increased to 95.5% on average (ranging from 70.0% (stein27) to 99.9% (air03)) for the MIPLIB instances, respectively to 99.9% on average for the 2-matching relaxation of the TSPLIB instances.

This reduction of almost 100% of the size of $\bar{A}$ yields an enormous speed-up in the



(a) MIPLIB instances (b) TSPLIB instances

Figure 1: Efficiency of preprocessing: reduction percentage in number of rows averaged over all applications of the separator



(a) MIPLIB instances (b) TSPLIB instances

Figure 2: Efficiency of preprocessing: speed-up in solving the auxiliary ILP

solving time of the auxiliary ILP (7) as shown in Figure 2. Here the CPU times needed to solve (7) with and without preprocessing (i.e., steps (d) to (g)) are displayed, averaged over all auxiliary ILPs within the branch&cut with a time limit of 10s (i.e., no further nodes of the auxiliary branch&cut are solved as soon as the time limit is exceeded). Note that solving times are given in seconds and we use a logarithmic scale to display them. The instances are sorted according non-decreasing average solving time of the auxiliary ILP without preprocessing.

Figure 2 shows that applying the preprocessing steps (d) to (g) reduces the solving time of the auxiliary ILP significantly. Assuming a solving time of 10s for those instances that reach the time limit, for 33% of the MIPLIB instances solving the auxiliary ILP can be sped up by a factor of at least 100. On average over all MIPLIB instances this factor exceeds 38. Considering the 2-matching relaxations, the solving time is sped up by a factor of at least 30 for all instances and exceeds 270 on average. Note that increasing the time limit would yield even higher speed-up factors.

	Fischetti&Lodi		SCIP default		$\{0, \frac{1}{2}\}$-cuts separator	
name	#nodes	#cuts	#nodes	#cuts	#nodes	#cuts
a280	1	104	273	31	1	99
ch150	1	141	13	36	1	58
eil101	1	43	1	60	1	20
gil262	1	266	513	72	1	331
gr120	1	45	1	37	1	25
gr137	1	31	1	61	1	39
gr229	1	224	242	66	1	83
kroA200	1	84	1	99	1	86
kroB200	1	558	616	63	1	135
lin318	1	768	>25010	>44	1	209
pr124	1	320	1	84	1	76
pr144	1	78	1	85	1	41
pr226	1	901	1	110	1	145
rat195	1	237	194	46	1	127
ts225	1	857	2687	138	1	231

Table 1: Efficiency of $\{0, \frac{1}{2}\}$-cuts separation on 2-matching problems

Effect of Separation. To identify the effect of $\{0, \frac{1}{2}\}$-cut separation, two natural keys values are available for comparison: the number of nodes of the branch&cut tree and the overall CPU time. Since the 2-matching polytope is completely described by the model inequalities and all $\{0, \frac{1}{2}\}$-cuts (in fact only the blossom inequalities suffice [12]), no branching is needed for these instances if the $\{0, \frac{1}{2}\}$-cuts are separated exactly. Hence, for the 2-matching relaxations of TSP instances from the TSPLIB an additional value to compare can be used: the number of cuts added to obtain an integral LP solution in the root of the branch&cut tree. Fischetti and Lodi [14] separated for these instances the more general Chvátal-Gomory cuts exactly. Therefore we compare three scenarios for these instances: the results presented in the paper by Fischetti and Lodi [14], SCIP default (i.e., with its standard separators and heuristics, etc.) and SCIP without its standard separators and heuristics but with our $\{0, \frac{1}{2}\}$-cut separation. To obtain as few $\{0, \frac{1}{2}\}$-cuts as necessary our separator adds only one cut per callback, namely the most-violated one. In order to add $\{0, \frac{1}{2}\}$-cuts to the LP until integrality is reached, we adjust some of SCIP's parameters (e.g., maximal number of consecutive separation rounds without improvement of objective and integrality). The results are shown in Table 1. Compared to Fischetti and Lodi we are able to solve 80% of the problems with less cuts. This suggests that often stronger cuts are generated which probably can be explained by the fact that we are not restricted on those cuts with $u^T A = \lfloor u^T A \rfloor$. Compared with SCIP default, we are able to solve the problems with less cuts in about 40%, but SCIP needs more than one branch&cut node in more than half the cases. In addition whenever it solves one of the problems in the root node, it needs strictly more cuts than our separator (except for the pr226 instance).

Next, we investigate the added value of our $\{0, \frac{1}{2}\}$-cut separator for general integer programs. We consider the pure integer problems from MIBLIB that can be solved within 1 hour with SCIP's default settings. We first compare on the number of branch&cut nodes needed with and without $\{0, \frac{1}{2}\}$-cut separation. For this, we use the following settings: $\{0, \frac{1}{2}\}$-cuts are separated exactly using the auxiliary ILP (7). The separator is called in every node of the branch&cut tree up to a depth of 15. Note that not only violated cuts obtained from the optimal solution of the auxiliary ILP, but also from earlier (non-optimal) solutions are added to the pool. In addition, we apply a simple postprocessing: All single rows whose corresponding variables are zero in the auxiliary ILP solutions (i.e., rows that are not part of the most violated $\{0, \frac{1}{2}\}$-cut yet) are checked. If one of these rows yields a violated $\{0, \frac{1}{2}\}$-cut, it is added to the pool as well. This way, the number of branch&cut nodes needed to solve a problem can be reduced by 26% on average (ranging from a reduction by 84% to an increase by 157%) at the cost of a higher overall solving time: an increase by 158% on average over all instances (primary induced by fast instances with solving times of less than a minute). The results are shown in Figure 3. Details can be found in Table 2 in Appendix A.

Performance gain ILP solver. Since the computation of an optimal solution to the auxiliary ILP (7) is time consuming and results in few violated cuts, such an approach is not suitable for integration in general purpose ILP solvers. Therefore, we finally consider three cases for a CPU time comparison:

(i) SCIP default

(ii) SCIP with our implementation as additional separator using the auxiliary ILP (7) to separate $\{0, \frac{1}{2}\}$-cuts exactly at the root only (cut&branch). Like in the test we used to compare on the branch&cut nodes, not only violated cuts obtained from the optimal solution of the auxiliary ILP, but also from earlier (non-optimal) solutions and from single rows not part of the most violated $\{0, \frac{1}{2}\}$-cut are added to the pool.

(iii) SCIP with our separator using the heuristic described in Section 3.2 to separate $\{0, \frac{1}{2}\}$-cuts at the root node only. Results of (ii) showed us that almost all added $\{0, \frac{1}{2}\}$-cuts are generated from a relative small number of rows of $\bar{A}$: on average only 2 or less "preprocessed" rows. The "original" rows (i.e., rows in A) implied by the preprocessing exceeds 10 on average and goes up to as high as 351 (mzzv11). Inspired by this observation we studied several settings for k. Based on the results of this study we set $k = 1$, i.e., we check all single rows of $(\bar{A}, \bar{b})$ if they yield a violated $\{0, \frac{1}{2}\}$-cut.

Based on extensive experiments, we restrict in all cases on those $\{0, \frac{1}{2}\}$-cuts with violation greater than 0.35 (i.e., $\varepsilon = 0.7$ in (8)) to avoid the generation of many weak cuts. Hence not all violated $\{0, \frac{1}{2}\}$-cuts are separated. We add all violated $\{0, \frac{1}{2}\}$-cuts from preprocessing to the pool and additionally up to 100 violated $\{0, \frac{1}{2}\}$-cuts found by the procedures described in case (ii) respectively (iii). The $p = 100$ best (w.r.t. their efficacy) cuts of the pool are added to SCIP which decides if they enter the LP (as it does for all its standard separators, as well).

Figure 4 shows the relative solving times of cases (ii) and (iii) w.r.t. case (i), e.g., a value of 0.8 means that solving the instance takes only 80% time compared to SCIP default (case (i)). There are two bars for each instance, the first one refers to case (ii), the second one to case (iii). Each bar is divided into two parts: the lower black part shows the fraction of the solving time spent by SCIP methods and the upper grey part shows the fraction spent within the $\{0, \frac{1}{2}\}$-cuts separator. The black boxes refer to the second y-axis on the right hand side which displays the absolute solving times of SCIP default (case (i)) in seconds, according to which the instances are non-decreasingly sorted. Details of the computations can be found in Table 2 in Appendix A.

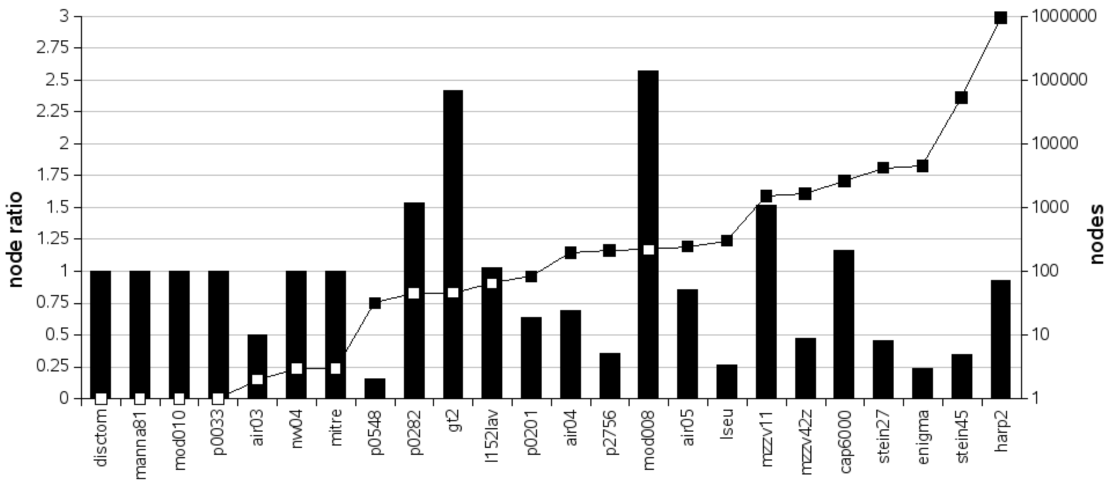


Figure 3: Efficiency of separation: ratio of branch&cut nodes without and with $\{0, \frac{1}{2}\}$-cuts [bars] and absolute numbers without [line with markers]

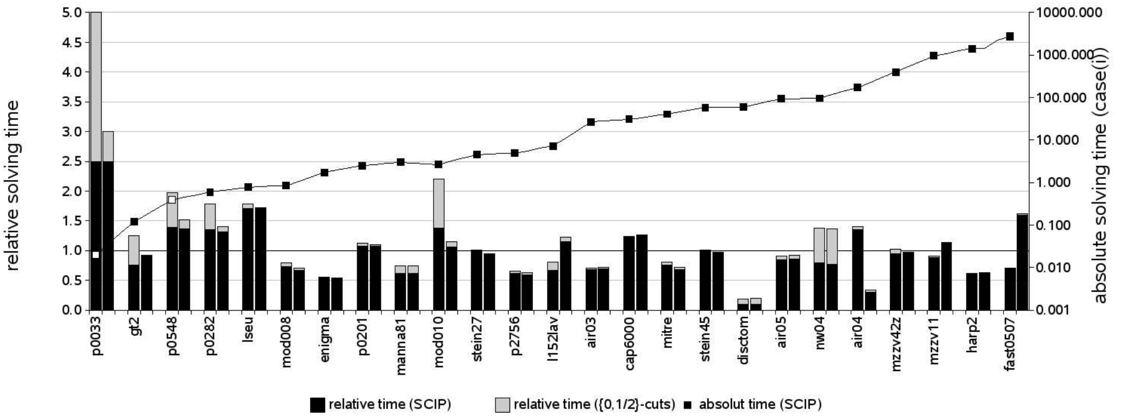


Figure 4: Efficiency of separation: quotient of solving times of cases (ii) and (iii) w.r.t. case (i) with a split in SCIP and separation time [bars], as well as absolute solving time of case (i) [line with markers]

From Figure 4, we conclude that our heuristic approach (case (iii)) performs mostly faster than the exact approach (case (ii)) (with a few exceptions like mzzv11 or fast0507). In particular, the more difficult instances in case (i) with solving times of more than 10 seconds (i.e., instances air03 and on) can often be solved faster. A slowed-down instance like nw04 might be sped up by a faster implementation of our separator, reducing the additional time. Finally we observe that our heuristic approach (case (iii)) reduces the solving time by 7% averaged over all instances (geometric mean), and by 21% averaged over the instances with absolute solving time greater than 10 seconds in case (i).

5 Conclusions

In this paper, we have developed algorithms to separate $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts in general integer programs, despite the $\mathcal{NP}$-completeness of the problem. Preprocessing rules turned out to be an indispensable part of such algorithms as they do not only reduce the size of the remaining separation problem but in many cases also provide violated inequalities directly, canceling the need for further processing. Separating the most violated $\{0, \frac{1}{2}\}$-cut yields a substantial reduction of the number of branch&cut nodes to be searched until optimality can be proven, whereas heuristic separation achieved the best time performance. The savings in computation time have already aroused the interest of both commercial and academic developers of integer programming solvers (e.g., they will be available in a future version of SCIP [2]).

The developed algorithms are at present only applicable to pure integer programs, i.e., without continuous variables. The extension of the separation procedure to general mixed integer programming problems remains as an important further research direction as these would enhance ILP-solvers further, like the recent work of Bonami et al. [7] showed for general CG cuts.

Acknowledgment

Most of the research has been carried out while the first and second author were at Zuse Institute Berlin (ZIB). The first author has been supported by the DFG research group “Algorithms, Structure, Randomness” (Grant number GR 883/9-3, GR 883/9-4) during that time.

References

[1] T. Achterberg. SCIP – a framework to integrate constraint and mixed integer programming. ZIB-Report 04–19, Zuse Institute Berlin, 2004. `http://www.zib.de/Publications/abstracts/ZR-04-19/`.

[2] T. Achterberg, T. Berthold, T. Koch, A. Martin, and K. Wolter. SCIP (Solving Constraint Integer Programs), 2006. `http://scip.zib.de/`.

[3] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. `http://miplib.zib.de`, 2003.

[4] G. Andreello, A. Caprara, and M. Fischetti. Embedding cuts in a branch&cut framework: a computational study with $\{0, \frac{1}{2}\}$-cuts. Technical report, University of Padova, 2003.

[5] R. Bixby and G. Reinelt. TSPLIB. `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html`.

[6] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. MIPLIB 3.0. `http://www.caam.rice.edu/~bixby/miplib/miplib.html`.

[7] P. Bonami, G. Cornuéjols, Sanjeeb Dash, Matteo Fischetti, and Andrea Lodi. Projected chvtalgomory cuts for mixed integer linear programs. *Mathematical Programming*, to appear, 2007.

[8] A. Caprara and M. Fischetti. $\{0, 1/2\}$-Chvátal-Gomory cuts. *Mathematical Programming*, 74:221–235, 1996.

[9] A. Caprara and M. Fischetti. Odd cut-sets, odd cycles, and $0 - 1/2$ Chvátal-Gomory cuts. *Ricerca Operativa*, 26:51–80, 1996.

[10] A. Caprara, M. Fischetti, and A. N. Letchford. On the separation of maximally violated mod-k cuts. *Mathematical Programming*, 87:37–56, 2000.

[11] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[12] J. Edmonds and E. L. Johnson. Matching: a well-solved class of integer linear programs. In R. K. Guy, H. Hanani, and N. Sauer, editors, *Combinatorial Structures and Their Applications*, pages 80–92. Gordon and Breach, New York, 1970.

[13] S. Fiorini. $\{0, \frac{1}{2}\}$-cuts and the linear ordering problem: Surfaces that define facets. *SIAM Journal on Discrete Mathematics*, 20(4):893–912, 2006.

[14] M Fischetti and A. Lodi. Optimizing over the first Chvátal closure. *Math. Programming*, 110(1):3–20, June 2007.

[15] C. Gentile, P. Ventura, and R. Weismantel. Mod-2 cuts generation yields the convex hull of bounded integer feasible sets. *SIAM Journal on Discrete Mathematics*, 20(4):913–919, 2006.

[16] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[17] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[18] ILOG. CPLEX version 10.0, 2006. `http://www.ilog.com/products/cplex`.

[19] A. M. C. A. Koster and A. Zymolka. Stable Multi-Sets. *Mathematical Methods of Operations Research*, 56(1):45–65, 2002.

[20] A. M. C. A. Koster and A. Zymolka. On cycles and the stable multi-set polytope. *Discrete Optimization*, 2(3):241–255, 2005.

[21] M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[22] A. Schrijver. On cutting planes. *Annals of Discrete Mathematics*, 9:291–296, 1980.

A Detailed computational results

Table 2 reports on the computational results for the MIPLIB instances. Four scenarios are compared: SCIP default, exact separation in every node of the branch&cut tree with depth at most 15, exact separation at the root node only, and heuristic separation, cf. Section 4. All computation times are in seconds.

	Size of A					$\{0, \frac{1}{2}\}$-cuts			$\{0, \frac{1}{2}\}$-cuts			$\{0, \frac{1}{2}\}$-cuts		
			#non-	SCIP default		exactly, depth $\leq$ 15			exactly, root only			heuristically		
name	#rows	#cols	zeros	#nodes	time	#nodes	time	#cuts	#nodes	time	#cuts	#nodes	time	#cuts
air03	124	10757	91028	2	26.70	1	20.71	38	1	19.00	15	1	19.32	15
air04	823	8904	72965	196	173.12	136	579.87	1980	347	243.30	104	5	59.25	258
air05	426	7195	52121	244	93.27	208	233.56	1157	357	85.09	56	335	86.21	119
cap6000	2176	6000	48243	2621	30.79	3045	129.39	1359	3207	38.06	3	3207	38.91	3
disctom	399	10000	30000	1	59.57	1	65.50	67	1	11.39	64	1	11.44	65
enigma	21	100	289	4455	1.72	1046	5.77	297	1698	0.95	1	1698	0.94	1
fast0507	507	63009	409349	1991	2758.62	timelimit was reached			1215	1955.17	33	2306	4458.55	52
gt2	29	188	376	46	0.12	111	0.46	1	46	0.15	0	46	0.11	0
harp2	112	2993	5840	951103	1408.72	878343	2288.00	1006	422090	862.96	3	422090	883.03	3
l152lav	97	1989	9922	65	7.23	67	136.42	401	16	5.82	193	39	8.89	228
lseu	28	89	309	302	0.78	80	0.93	42	1329	1.39	9	1329	1.34	9
manna81	6480	3321	12960	1	2.67	1	2.68	116	1	1.97	272	1	1.97	272
mitre	2054	10724	37671	3	40.90	3	34.31	337	1	33.00	317	4	29.52	305
mod008	6	319	1243	217	0.85	558	3.89	120	137	0.68	2	137	0.60	2
mod010	146	2655	11203	1	2.99	1	3.10	7	10	6.59	147	1	3.42	82
mzzv11	2054	10724	37671	1525	964.05	2317	3483.63	8365	1771	870.49	140	4983	1098.48	107
mzzv42z	10460	11717	151261	1638	401.13	774	681.17	1504	1110	408.57	165	2504	389.08	80
nw04	36	87482	636666	3	96.08	3	188.13	86	5	132.64	6	7	130.60	10
p0033	16	33	98	1	0.02	1	0.40	12	2	0.14	9	2	0.06	9
p0201	133	201	1923	83	2.48	53	3.99	65	194	2.78	25	194	2.72	25
p0282	241	282	1966	45	0.60	69	2.15	106	45	1.07	29	45	0.84	29
p0548	176	548	1711	32	0.39	5	0.56	34	18	0.77	17	18	0.59	17
p2756	755	2756	8937	212	4.94	76	4.10	126	94	3.23	58	94	3.09	58
stein27	118	27	378	4173	4.57	1912	59.47	3211	4194	4.60	1	4194	4.34	1
stein45	331	45	1034	53189	58.36	18579	698.57	48786	50835	58.77	3	50122	56.71	3

Table 2: Detailed computational results for the MIPLIB instances