

11-11-2009

On Families of Split Cuts that Can Be Generated Efficiently

G rard Cornu jols

Carnegie Mellon University, gc0v@andrew.cmu.edu

Giacomo Nannicini

Carnegie Mellon University

Follow this and additional works at: <http://repository.cmu.edu/tepper>

 Part of the [Economic Policy Commons](#), and the [Industrial Organization Commons](#)

On Families of Split Cuts that Can Be Generated Efficiently

GERARD CORNUÉJOLS¹, GIACOMO NANNICINI²

¹ *Tepper School of Business, Carnegie Mellon University, 15217 Pittsburgh, PA, USA, and LIF, Faculté des Sciences de Luminy, 13288 Marseille, France*

Email: gc0v@andrew.cmu.edu

² *Tepper School of Business, Carnegie Mellon University, 15217 Pittsburgh, PA, USA,*

Email: nannicin@andrew.cmu.edu

November 11, 2009

Abstract

Split cuts represent the most widely used class of cutting planes currently employed by state-of-the-art branch-and-cut solvers for mixed integer linear programming. Rank-1 cuts often have better numerical properties than higher rank cuts. In this paper, we study several heuristics to generate new families of strong rank-1 split cuts, by considering integer linear combinations of the rows of the simplex tableau, and deriving the corresponding mixed-integer Gomory cuts. In particular, we propose several cut generation algorithms that share the following aims: reducing the number of nonzeros, obtaining small coefficients, generating orthogonal cuts. A key idea is that of selecting a subset of the variables, and trying to generate a cut which cuts deeply on those variables. We show that variables with small reduced cost are good candidates for this purpose, yielding cuts that close a larger integrality gap. On a set of test instances where standard split cut generators close on average 28.8% of the integrality gap in the first pass, we can close 35.3% by investing 10 times as much cut generation time. Incorporating our new split cuts into a branch-and-cut algorithm yields an improvement in the overall performance: except on very easy instances, where our procedure is too slow to bring advantage, on average we can save 23% computing time on instances which are solved, and close more integrality gap on unsolved instances in a fixed amount of time.

1 Introduction

Mixed-integer linear programs (MILPs) are mathematical programs with linear objective and constraints where some of the variables are restricted to take on integer values. Although solving an integer program is NP-hard in general [15], there exist both commercial and free software packages that are able to efficiently solve many MILPs arising from real-life applications. A dramatic improvement in the performance of this software came from the introduction of general cutting planes, such as Gomory mixed-integer (GMI) cuts [12], and Mixed-Integer Rounding (MIR) cuts [14]. These cuts fall into the category of *split cuts* [8], that is, cutting planes that can be derived as intersection cuts [3] from a disjunction of the form $\pi^\top x \leq \pi_0 \vee \pi^\top x \geq \pi_0 + 1$ with (π, π_0) integer.

In this paper we study split cuts for MILPs, and in particular we focus on rank-1 split cuts; rank-1 denotes the fact that they can be derived directly from the simplex tableau, without iterated cut generation. It has been shown these cuts provide a tight approximation of the convex hull of the feasible integer points of a MILP [4, 5, 10], yet available cut generators exploit only a small part of this power. Furthermore, rank-1 cuts are known to enjoy better numerical properties compared their higher rank counterparts. Computational studies [13] show that available cut generators may lead into numerical problems, possibly cutting off the optimal solution to the mathematical program; rank-1 cuts are less likely to display this kind of detrimental behaviour, because of their sparsity (see e.g. [11]) and because the fractionalities involved in the cut tend to be not too small or too large if the initial constraint matrix is numerically well-behaved.

This paper describes several methods to generate split cuts by considering linear combinations with integer coefficients of the rows of the optimal simplex tableau, and deriving the corresponding GMI cut. This work is related to the Reduce-and-Split cut generation algorithm of Andersen, Cornuéjols and Li [2] in that they also consider linear combinations of the rows of the simplex tableau to generate GMI cuts, but our algorithm to compute these combination is significantly different. In particular, our algorithm is derived from the one proposed by Cornuéjols, Liberti and Nannicini [9] in the context of computing promising branching directions. In this paper we build upon their algorithm, employ it for cut generation, and develop different strategies to select the rows involved in the linear combination, as well as different criteria for computing their multipliers. We are guided in this study by three objectives: generating cuts with a small number of nonzeros, with small coefficients, and mutually orthogonal.

A key idea is that of selecting a subset of the nonbasic variables, and trying to generate a cut which has small coefficients (and hopefully many zeroes) on the corresponding columns. This is one of the original contributions of this paper. By iterating this process selecting disjoint variable sets, we aim to obtain cutting planes which are orthogonal with respect to each other. This way, we can obtain a large number of different cuts, which prove to be practically effective: on a test set of mixed-integer instances where the rank-1 cut generators available in the Cut Generation Library (CGL) [7] close on average 28.8% of the integrality gap, which represents our baseline, adding our cuts yields an additional 7.6% of closed gap. This comes at a computational cost which is roughly 100 times the amount of CPU time required by the existing rank-1 cut generators in CGL; however, by employing only the most successful cut generation strategies, we can decrease our computational time by a factor of 10, while still obtaining 6.5% more closed gap than the baseline. We provide extensive computational experiments to assess the usefulness of our approach.

Another contribution of this paper is providing evidence of the importance of reduced costs in determining the strength of a cut: we show that cutting planes which cut deeply on variables with small reduced costs achieve a larger improvement in the objective function, thus being of great practical importance. We obtain such cuts by applying our algorithm on a subset of variables with small reduced costs.

The rest of this paper is organized as follows. In Section 2 we provide the necessary theoretical background. In Section 3 we discuss the cut generation algorithm, and several related issues. In Section 4 we provide a computational evaluation.

2 Theoretical background

Consider the following Mixed Integer Linear Program in standard form:

$$\min\{c^\top x : Ax = b, x \geq 0, \forall j \in N_I x_j \in \mathbb{Z}\}, \quad \mathcal{P}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and $N_I \subset N = \{1, \dots, n\}$. The LP relaxation of \mathcal{P} is the linear program obtained by dropping the integrality constraints, and is denoted by $\bar{\mathcal{P}}$. Let $B \subset N$ be an optimal basis of $\bar{\mathcal{P}}$, and $J = N \setminus B$ is the set of nonbasic variables. The corresponding simplex tableau is given by:

$$x_i = \bar{x}_i - \sum_{j \in J} \bar{a}_{ij} x_j \quad \forall i \in B. \quad (1)$$

Consider an arbitrary equality $\sum_{j \in N} g_j x_j = d$ satisfied by all feasible solutions to \mathcal{P} . Define $f_0 := d - \lfloor d \rfloor$ and $f_j := g_j - \lfloor g_j \rfloor$ for all $j \in N_I$. Suppose $f_0 > 0$; the GMI cut associated with this equation is:

$$\sum_{j \in N_I: f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{j \in N_I: f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j + \sum_{j \in N \setminus N_I: g_j \geq 0} \frac{g_j}{f_0} x_j - \sum_{j \in N \setminus N_I: g_j < 0} \frac{g_j}{1 - f_0} x_j \geq 1. \quad (2)$$

It can be shown that (2) is valid for \mathcal{P} [12]. Let $B_I = B \cap N_I$, $J_I = J \cap N_I$, $J_C = J \setminus N_I$ be the sets of integer basic variables, integer nonbasic variables and continuous nonbasic variables respectively. Now consider a linear combination with integer coefficients λ_i of the rows (1) where the corresponding basic variable is constrained to be integer:

$$\sum_{i \in B_I} \lambda_i x_i = \tilde{x} - \sum_{j \in J} \tilde{a}_j x_j, \quad (3)$$

where

$$\begin{aligned} \tilde{x} &= \sum_{i \in B_I} \lambda_i \bar{x}_i \\ \tilde{a}_j &= \sum_{i \in B_I} \lambda_i \bar{a}_{ij} \text{ for } j \in J. \end{aligned} \quad (4)$$

(3) is an equation satisfied by all feasible solutions to \mathcal{P} , which yields $f_0 = \tilde{x} - \lfloor \tilde{x} \rfloor$, $f_j = \tilde{a}_j - \lfloor \tilde{a}_j \rfloor$ for all $j \in J$ according to the definition above. Note that in order to generate a GMI cut we need $\tilde{x} \notin \mathbb{Z}$ and therefore $\lambda \neq 0$. Applying (2) to (3) we obtain:

$$\sum_{j \in J_I: f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{j \in J_I: f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j + \sum_{j \in J_C: \tilde{a}_j \geq 0} \frac{\tilde{a}_j}{f_0} x_j - \sum_{j \in J_C: \tilde{a}_j < 0} \frac{\tilde{a}_j}{1 - f_0} x_j \geq 1. \quad (5)$$

This is the GMI cut associated with the row obtained through the row multipliers λ . In order to generate a deep cut, we want to find integer multipliers λ_i for all $i \in B_I$ such that the resulting coefficients in (5) are as small as possible. Note that the coefficients of the cut on the variables x_j for $j \in J_I$ are always between 0 and 1, whereas the coefficients on variables with index set J_C are not bounded. Furthermore, for $j \in J_I$ both the numerator and the denominator of the coefficient on x_j depend nonlinearly on λ , whereas for $j \in J_C$ the denominator of the coefficient on x_j depends nonlinearly on λ , but the numerator has a linear dependence. It seems difficult to optimize the coefficients of the cut on the integer nonbasic variables ($j \in J_I$); in this paper, we focus on the coefficients of the continuous nonbasic variables ($j \in J_C$), and try to generate cuts which have small coefficients on at least some of these variables.

3 The reduction algorithm

In order to obtain cutting planes with small coefficients (and hopefully zeroes) on the continuous nonbasic columns, we select $J_W \subseteq J_C$, and we attempt to minimize \tilde{a}_j for $j \in J_W$ over vectors $\lambda \in \mathbb{Z}^{|B_I|}$. More specifically, we would like to minimize $\|\tilde{d}\|$, where

$$\tilde{d} = (\tilde{a}_j)_{j \in J_W}. \quad (6)$$

This idea has been pursued also in [2, 9], although those papers always consider the whole set of continuous nonbasic columns, i.e. $J_W \equiv J_C$. Our algorithm to compute λ in order to reduce the norm of \tilde{d} is derived from the one presented in [9] in the context of generating good branching directions. We now describe this algorithm, which we call *reduction algorithm* in the following.

Choose $J_W \subseteq J_C$; apply a permutation to the simplex tableau in order to obtain $B_I = \{1, \dots, |B_I|\}$, $J_W = \{1, \dots, |J_W|\}$, and define the matrix $D \in \mathbb{R}^{|B_I| \times |J_W|}$, $d_{ij} = \tilde{a}_{ij}$. Thus, we can rewrite the problem of minimizing $\|\tilde{d}\|$ as

$$\min_{\lambda \in \mathbb{Z}^{|B_I|} \setminus \{0\}} \left\| \sum_{i \in B_I} \lambda_i d_i \right\|. \quad (7)$$

This is a shortest vector problem in the additive group generated by the rows of D . Assuming linear independency between the rows, the group defines a lattice, and (7) becomes the shortest vector problem in a lattice, which is NP-hard under randomized reductions [1].

In [9], for each row d_k of D , a subset $R_k \subset B_I$ of the rows of the simplex tableau with $d_k \in R_k$ is chosen; then, integral multipliers are calculated, in order to reduce $\|d_k\|$ as much as possible with a linear combination of the rows d_i for all $i \in R_k \setminus \{k\}$. Relaxing the integrality requirements on λ , for each row d_k that we want to reduce we have the following convex minimization problem:

$$\min_{\lambda^k \in \mathbb{R}^{|R_k|}, \lambda_k^k = 1} \left\| \sum_{i \in R_k} \lambda_i^k d_i \right\|. \quad (8)$$

An integer (not necessarily optimal) solution can be found by rounding the coefficients λ_i^k to the nearest integer $\lfloor \lambda_i^k \rfloor$. The optimal continuous multipliers λ that solve (8) can be obtained through the solution of a $|R_k| \times |R_k|$ linear system (see [9] for details).

Once the linear systems are solved and the optimal continuous coefficients $\lambda^k \in \mathbb{R}^{|R_k|}$ for all $k \in \{1, \dots, |B_I|\}$ are available, they are rounded to the nearest integer. Then, consider the norm of $\sum_{i \in R_k} \lfloor \lambda_i^k \rfloor d_i$. If $\|\sum_{i \in R_k} \lfloor \lambda_i^k \rfloor d_i\| < \|d_k\|$, there is an improvement with respect to the

original row of the simplex tableau, and we expect the associated GMI cut to be stronger, at least on the working set of columns J_W ; in this case, the equation

$$\sum_{i \in R_k} \lambda_i^k x_i = \sum_{i \in R_k} \lambda_i^k \bar{x}_i - \sum_{j \in J} \sum_{i \in R_k} \lambda_i^k \bar{a}_{ij} x_j, \quad (9)$$

is used in order to compute a GMI cut according to (5).

We now propose several strategies to choose the set of working variables J_W , and a corresponding set of rows R_k for the coefficient reduction algorithm.

3.1 Selection of the working set of nonbasic continuous variables

In order to generate a large number of split cuts, we work on different sets of continuous nonbasic variables, aiming each time to reduce the coefficients for the variables in the working set J_W . This serves two purposes: one is to generate more cuts, and the second is to obtain more orthogonal cutting planes. We generate more cuts because, given a reduction algorithm to compute the row multipliers λ (such as the one described in Section 3) and a set of rows R_k , we potentially obtain one new cut for each different J_W . We obtain more orthogonal cutting planes because we generate cuts with small coefficients, potentially zeroes, on the columns in the set J_W ; by iterating the cut generation algorithm choosing disjoint sets J_W at each iteration, we obtain a collection of cutting planes which should be more orthogonal with each other than the GMI cuts obtained from the original simplex tableau rows.

We take into account the reduced costs of the variables with index set J_C for choosing J_W . The motivation is that there is a direct relationship between the improvement in the objective function given by a cut, its coefficients, and the reduced costs of the variables. Consider the current solution to the LP relaxation \bar{x} and the extreme rays $r^j, j \in J$ of the cone $C = \{Ax = 0, x_j \geq 0 \forall j \in J\}$ associated with the corresponding basic solution. Note that $c^\top r^j$ is exactly the reduced cost associated with x_j ; we denote it by \bar{c}_j in the following. Given a GMI cut (5), rewrite it as an intersection cut $\sum_{j \in J} \frac{x_j}{\alpha_j} \geq 1$, so that the cutting plane supports the points $x^j = \bar{x} + \alpha_j r^j$ (see [3] for details). Now the objective function value of \mathcal{P} after the addition of the cut is at least $\min_{j \in J} c^\top x^j = \min_{j \in J} (c^\top \bar{x} + \alpha_j c^\top r^j)$. In order to maximize this quantity, it is clear that for the variables with small reduced cost we should aim for a large α_j , i.e. a small cut coefficient. We give another reason for taking into account reduced costs. Variables with large reduced costs are likely to stay nonbasic even after the addition of cutting planes, whereas variables with small reduced costs have a larger probability of entering the basis; therefore, cutting planes which cut deeply on variables with small reduced costs are more likely to have a large effect on the solution to the LP relaxation, i.e. they “move” \bar{x} by a larger amount.

We choose the working sets J_W 's by partitioning the set of continuous nonbasic columns into k disjoint sets; each of these partitions gives rise to k disjoint J_W 's, hence a collection of potentially orthogonal cuts. However, we do not always partition the whole set of continuous nonbasic columns J_C : in some cases, we only consider the variables with smallest reduced costs. When computing the partitions, we always sort the variables in J_C by increasing reduced costs; we denote by S the ordered set obtained by sorting J_C by increasing reduced costs \bar{c}_j . We distinguish between contiguous k -partitions and k -partitions with interleaving pattern. By *contiguous k -partition* we denote a partition S_1, S_2, \dots, S_k such that for $i < j, \forall p \in S_i, q \in S_j$ we have $\bar{c}_p \leq \bar{c}_q$, and the cardinality of S_i differs by at most one from the cardinality of any other set S_j . In other words, each set of the partition contains variables which are contiguous in

S . By k -partition with interleaving pattern we denote a partition S_1, S_2, \dots, S_k of the first $r|J_C|$ variables of S where $0 < r \leq 1$, the cardinality of S_i differs by at most one from the cardinality of any other set S_j , and from each set of $2k$ contiguous variables, we assign 2 variables to each of the k sets S_i . We denote a contiguous k -partition by C- k P, and a k -partition with interleaving pattern by I- k P- r or simply by I- k P if $r = 1$. We consider the following 7 partitions: C-3P, C-5P, I-2P-1/2, I-2P-2/3, I-2P-4/5, I-3P, I-4P. We believe that the particular pattern used to generate the partition, i.e. which variables are included in each set of the partition, is not relevant except for the contiguous partitions; indeed, preliminary computational tests conducted by generating the same number of random partitions with the same characteristics, showed very similar results. We chose to eliminate the computational overhead of generating random partitions by hard-coding different variable selection patterns. The two contiguous partitions do not follow the scheme, in that each set in the partition contains variables which are contiguous in S . This way we try to create a ranking of cutting planes: the first set of each of these two partitions should give stronger cutting planes, whereas the remaining sets of the partition should yield progressively weaker cuts. This allows to verify our claim that reduced costs play an important role in determining the quality of a cut; a computational analysis of this conjecture is given in Section 4.1.

3.2 Selection of the rows for the reduction algorithm

In order to reduce the norm of row d_k , the reduction algorithm of Section 3 needs a set of candidate rows R_k for the linear combination. Here we propose 4 different criteria to choose R_k . There are two natural objectives for an effective row selection strategy: on the one hand, we want to select rows which allow for a large reduction of the coefficients on the continuous nonbasic variables; on the other hand, we do not want to deteriorate the cut on the integer nonbasic variables, as those are not taken into consideration when solving the optimization problem (8). For each row selection strategy we have a parameter μ , which represents the maximum number of rows that should be selected; that is, $|R_k| \leq \mu$.

The first row selection strategy (RS1) is the original one proposed in [9], and it consists in ranking the rows with index set $B_I \setminus \{k\}$ by increasing number of nonzeros on the nonbasic integer columns where d_k is zero (in this section we always use the row number as a tie breaker), and picking the first μ such rows. The motivation is that keeping under control the coefficients of the cut on the integer nonbasic columns is a difficult task, therefore we simply try not to introduce many nonzeros on the columns where the original row has a zero.

The second row selection strategy (RS2) is a variant of the first one: we employ a greedy algorithm to select, at each iteration, the row which introduces the smallest number of nonzeros on the integer nonbasic columns where the original row d_k is zero and we did not introduce a nonzero at previous iterations. We use the row number as a tie breaker. The greedy algorithm is iterated until μ rows are selected. This strategy is computationally more expensive than the first one, but should result in a smaller number of nonzeros in the final cut.

The third row selection strategy (RS3) is a variant of the second one: we employ a greedy algorithm to select, at each iteration, the row which introduces the smallest number of nonzeros on the variables with index set $J_I \cup J_W$ where the original row d_k is zero and we did not introduce a nonzero at previous iterations. The motivation is that we want to select rows that can be combined to reduce the coefficients on the continuous nonbasic variables, but overall we would like to have a small number of nonzeros in the cut.

The fourth row selection strategy (RS4) chooses the first μ rows with index in $B_I \setminus \{k\}$ such that their angle with respect to d_k in the space $J_I \cup J_W$ is the smallest.

4 Computational experiments

In this section we provide a detailed computational evaluations of the ideas discussed above. All the experiments were run on a machine with a multicore CPU Intel Xeon X3220 and 4 GB RAM, running Linux. We used only one core out of the 4 available. We employed CBC 2.3 [6] as Branch-and-Cut software with CLP 1.10 as LP solver, and the cutting plane generators are provided by CGL 0.54 [7]. We implemented the algorithms presented in Section 3 and subsections as a stand-alone cut generator which accepts as a parameter the maximum number of rows to be considered at each reduction step μ , the column selection strategy, and the row selection strategy. This allows us to test the individual contribution of the different parameters. We employed a naive implementation to be able to conduct tests quickly, thus we are confident that we will improve computing times in the future. An important issue raised in [13] is the methodology that should be used to test quality of cut generators: indeed, two cut generators should be compared only if they have the same level of safety. We want to avoid cases when one cut generator looks stronger in terms of closed gap, only because it is more aggressive and less safe, thus prone to cutting off the optimal solution. A discussion on testing cut generators is beyond the scope of this paper. In order to have a fair comparison, we implemented our own GMI cut generator, and we use the same procedures and parameters to generate both the GMI cuts that can be read directly from the simplex tableau, and the split cuts that we obtain by applying the GMI formula to a linear combinations of rows. This way, both traditional GMI cuts and our split cuts should have the same numerical safety; furthermore, in this paper we focus on rank-1 cuts, which tend to be numerically clean. We did not reimplement the other cut generators included in CGL that we employed in our computational experiments. In our GMI cut generator, a cutting plane is computed from an equality of the form (3) only if $\min\{\tilde{x} - \lfloor \tilde{x} \rfloor, \lceil \tilde{x} \rceil - \tilde{x}\} \geq 0.1$. If the fractionality of the right hand side of the equation is < 0.1 , the row is discarded and no cut is generated. Moreover, in order to have better numerical accuracy, we recompute the basis inverse and the optimal simplex tableau from scratch, instead of using the rows of the simplex tableau provided by CLP.

As a test set, we selected all mixed-integer instances from MIPLIB3_C_V2, available at <http://wpweb2.tepper.cmu.edu/fmargot/index.html>, plus the following mixed-integer instances from MIPLIB 2003: `a1c1s1`, `aflow30a`, `aflow40b`, `roll3000`, `timtab1`, `timtab2`, `tr12-30`. These instances were chosen because they provide a challenge, yet the underlying LP is of manageable size, and allows computational experiments in reasonable time. We applied to the MIPLIB 2003 instances the modification procedure described in [13], so that we have a 0-feasible solution, akin to the MIPLIB3_C_V2 instances. We consider the 0-feasible solution as the optimal solution to provide a cutoff value. Table 1 provides a list of the 43 instances that we used through this whole section. In the following, all reported averages are geometric; to deal with quantities that can assume the value 0, we add 1 to each value before computing the average, and subtract 1 from the final result. When computing the average amount of integrality gap closed, we will not consider instances where all tested methods close 0 integrality gap. For space reasons, through this section we cannot give detailed results for each instance, hence we will report the average values over the full test set only.

a1c1s1_c (H)	aflow30a_c (M)	aflow40b_c (H)	arki001_c (H)	bell3a_c (E)	bell4_c (M)
bell5_c (E)	blend2_c (M)	danoint_c (H)	dcmulti_c (E)	egout_c (E)	fiber_c (M)
fixnet3_c (E)	fixnet4_c (E)	fixnet6_c (M)	flugpl_c (E)	gen_c (E)	gesa2_c (M)
gesa2_o_c (M)	gesa3_c (E)	gesa3_o_c (E)	khb05250_c (E)	misc06_c (E)	mkc_c (H)
mod011_c (H)	modglob_c (H)	noswot_c (H)	pk1_c (M)	pp08a_c (M)	pp08aCUTS_c (M)
qiu_c (H)	qnet1_c (M)	qnet1_o_c (E)	rgn_c (E)	roll3000_c (H)	rout_c (H)
set1ch_c (M)	swath_c (H)	timtab1_c (H)	timtab2_c (H)	tr12-30_c (H)	vpm1_c (E)
vpm2_c (M)					

Table 1: Test instances and corresponding difficulty level (E = easy, M = medium, H = hard; see Section 4.2 for details).

4.1 Experiments with split cuts at the root node

In this section we consider the performance of split cuts at the root node: we generate *only one pass* of cutting planes, and analyze the strength of the resulting LP model. In particular, we want to answer the following questions: can we obtain, through the cut generation algorithms proposed in Section 3, some rank-1 cuts which are not generated by existing cut generators? If so, how strong are these new cutting planes?

From a practical point of view, we believe that new cut generators should be tested by adding them on top of the existing generators rather than as stand alone. The reason is that a cut generator which performs well on its own, but such that its performance can be replicated by existing generators, is useless for practical purposes. Therefore, we consider as our baseline the union of a GMI cut generator (our implementation, see Section 4) and of the CGL versions of Lift and Project (CGLLANDP), MIR (CGLMIXEDINTEGERROUNDING and CGLMIXEDINTEGERROUNDING2), knapsack cover inequalities (CGLKNAPSACKCOVER) and flow cover inequalities (CGLFLOWCOVER). For the CGL cut generators, we use the default parameters. We denote this collection of cut generators by CGLCUTS. Cuts obtained through CGLCUTS are always added to the formulation.

As mentioned earlier, our cut generator has 3 parameters: the maximum number of rows involved in the reduction algorithm μ , the column selection strategy, and the row selection strategy. We consider 5 possible different values for μ : 5, 10, 15, 20, 50. Since this parameter determines the size of the linear systems involved in the computations, it also mostly determines the computing time. Already for $\mu = 50$ the CPU time increase with respect to CGLCUTS is significant, thus we did not try larger values. As for the column selection strategies, there are, in total, 24 different working sets of variables J_W , as described in Section 3.1, each one of which could give rise to different cutting planes. Finally, there are 4 row selection methods, presented in Section 3.2. This yields 480 different cut generators: potentially, for each row where the corresponding basic integer variable is fractional we could generate 480 different split cuts; we use a simple duplicate detection system, so that the same cut is not added twice to the LP. At this stage, we are not interested in computational times. We now give the average amount of integrality gap closed on our test set by CGLCUTS alone and after adding all the cuts obtained through the 480 cut generators described above:

- CGLCUTS alone: 28.80% (120.1 cuts on average, CPU time 0.35 seconds);
- CGLCUTS + all new cuts: 36.38% (721.23 cuts on average, CPU time 34.5 seconds).

Number of rows μ		
$\mu = 5$:	36.28%	11
$\mu = 10$:	36.34%	8
$\mu = 15$:	36.35%	10
$\mu = 20$:	36.31%	10
$\mu = 50$:	36.21%	15

Column selection strategy								
	Single set						Full part.	
C-3P:	36.35% 3	36.37% 3	36.37% 3					36.31% 9
C-5P:	36.36% 8	36.37% 6	36.33% 4	36.36% 8	36.38% 2			36.16% 19
I-2P-1/2:	36.32% 5	36.36% 5						36.29% 8
I-2P-2/3:	36.36% 5	36.37% 5						36.34% 9
I-2P-4/5:	36.34% 3	36.37% 5						36.33% 7
I-3P:	36.38% 5	36.33% 4	36.15% 8					36.10% 12
I-4P:	36.35% 3	36.34% 6	36.34% 6	36.38% 3				36.24% 12

Row selection strategy		
RS1:	36.35%	8
RS2:	36.36%	2
RS3:	35.88%	14
RS4:	36.10%	19

Table 2: Results obtained by *excluding* all cut generators with a given parameter value; we report, for each excluded parameter, the integrality gap closed by the remaining cut generators and the number of instances where the gap closed decreases with respect to employing all generators. For the column selection strategy, we report on each row the working sets of a different partition: for the contiguous partition, the sets are ordered from lowest reduced cost (left) to largest reduced cost (right). The final column reports results obtained excluding all working sets of the partition.

This gives a first indication of the strength of the cut generation algorithms that we propose: they are able to contribute an additional 7.58% of closed integrality gap, by generating 5 times as many distinct cuts as those computed by CGLCUTS. This suggests that most of the 480 generators are redundant and generate the same cutting planes. Thus, our objective is to measure the marginal contribution of each cut generator, so that we can select only the most successful ones, and develop a cut generation algorithm that is competitive also in terms of computing time. To this end, we design two experiments.

We analyze the difference in the gap closed by the 480 cut generators described above and by a specific subset of those only, when added on top of CGLCUTS. In the first experiment we design the subsets in order to contain all generators *except* those with a specific parameter value; this allows us to quantify the strength of the cutting planes which are contributed by generators with the given parameter value and which cannot be obtained through any other generator. Results can be found in Table 2; besides the gap, we also report, for each parameter value, the number of instances for which excluding that parameter value results in a decrease of the gap closed. We notice that all cut generators give at least some contribution: there is no parameter that can be safely removed while keeping the same amount of closed gap on all the instances. Interestingly, removing cut generators with $\mu = 50$ yields a significant decrease in the average closed gap, which implies that considering a large amount of rows for the linear combination produces effective cutting planes that we are not able to obtain through other means. Furthermore, row strategies RS1 and RS2 are clearly less effective than RS3 and RS4. In particular, RS3 seems to be stronger than the rest, suggesting that combining rows in such a way that they introduce a small number of nonzeros is indeed the best choice. It seems that no conclusion can be drawn regarding the effectiveness of the partitions: since there is redundancy, excluding just one set of working variables does not yield a large decrease in the average closed gap. The next experiment gives a clearer indication on this point.

In the second experiment we still consider the performance of subsets of the 480 cut generators, but in this case, the subsets are selected in such a way that they contain exactly those

Number of rows μ		
$\mu = 5$:	34.24%	2.8
$\mu = 10$:	35.30%	4.5
$\mu = 15$:	36.64%	6.4
$\mu = 20$:	35.70%	8.2
$\mu = 50$:	35.23%	18.7

Column selection strategy										
	Single set								Full part.	
C-3P:	32.31%	2.1	31.80%	2.7	29.89%	2.8			34.08%	6.4
C-5P:	32.06%	1.8	31.05%	1.7	31.71%	2.3	30.22%	2.2	29.95%	2.3
I-2P-1/2:	32.04%	2.2	31.55%	2.2						
I-2P-2/3:	32.69%	2.6	31.91%	2.6						
I-2P-4/5:	32.65%	2.8	32.09%	2.9						
I-3P:	32.44%	2.8	31.89%	2.8	31.38%	2.9				
I-4P:	30.61%	2.6	30.85%	2.6	31.48%	2.6	31.39%	2.6		
									33.85%	7.8

Row selection strategy		
RS1:	34.53%	6.8
RS2:	34.61%	8.8
RS3:	35.88%	17.2
RS4:	35.08%	6.7

Table 3: Results obtained by employing only the cut generators with a given parameter value; we report, for each parameter value, the integrality gap closed by all generators with that parameter value, and the corresponding CPU time in seconds. For the column selection strategy, we report on each row the working sets of a different partition: for the contiguous partition, the sets are ordered from lowest reduced cost (left) to largest reduced cost (right). The final column reports results obtained by generating cuts employing all working sets of the partition.

generators that have a specified parameter value. Results can be found in Table 3: for each parameter value, we report the average gap closed by the cut generators with that parameter value, and the average CPU time for cut generation; this allows us to compare the relative speed of cut generators. We now analyze the figures in Table 3. First of all, using $\mu = 15$ clearly yields the best results on average; all remaining values for μ close significantly less gap. It is interesting to note the difference with Table 2 where most cuts generated by generators with $\mu = 15$ can be obtained through the other generators, whereas they are the most effective when used alone. The converse applies to $\mu = 50$. In terms of computation times, $\mu = 50$ seems too slow for practical purposes. Comparing the row selection strategies, we have another clear indication that RS3 and RS4 are stronger than RS1 and RS2; RS3 is computationally expensive, but leads to a noticeably larger average closed gap. Analyzing the gap closed by the different working sets, and in particular those of the contiguous partitions C-3P and C-5P, it is evident that cutting planes with small coefficients on variables with small reduced costs are more effective: the average closed gap *significantly* decreases as we move from reducing the coefficients on a column set with the smallest reduced costs to reducing the coefficients on a column set with the largest reduced cost.

Finally, building on the insight gained with the previous experiments, we only choose the most successful cut generators, and compare strength, number of cuts and computational time. In particular, we only employ cut generators with $\mu = 15$ and row selection strategy RS3 or RS4. We employ all column selection strategies; we call this collection of cut generators FASTRS (Fast Reduce-and-Split) in the following. We obtain the following results:

- CGLCUTS alone: 28.80% (120.1 cuts on average, CPU time 0.35 seconds);
- CGLCUTS + FASTRS: 35.29% (278.0 cuts on average, CPU time 3.72 seconds).

On average, FASTRS generates roughly the same number of distinct cutting planes as CGLCUTS, and requires ≈ 10 times as much CPU time. The increase in the average closed gap is 6.4% in absolute terms, which represents an increase of 22.5% with respect to CGLCUTS alone in relative terms. For difficult MILP problems, this increase in closed gap, obtained through rank-1 cuts

instead of resorting to higher rank cuts, should prove to be practically important; this is what we want to assess in the next section.

4.2 Experiments with Branch-and-Cut

In this section we employ the split cuts that we generate in a Branch-and-Cut framework, and discuss their impact on the practical performance of the algorithm. Our basic setup for the experiments is the following: we apply 5 rounds of cutting planes with all the generators in CGLCUTS, then we start branching for 30 minutes using a *best bound* node selection criterion. At each node whose depth is multiple of 4, we apply one round of GMI cuts. The value of the optimal solution is given as a cutoff, and all heuristics are disabled; this setup is meant to be effective for proving optimality of a solution. We test 3 different Branch-and-Cut algorithms. The first one is the basic setup described above, which we call CBCBASIC in the following. The second one is the basic setup with the addition of one round of cutting planes generated by FASTRS at the root node, which we call CBCFASTRS. The third version is a variant of CBCFASTRS, where all rank-1 cuts generated by CGLCUTS and FASTRS are stored in a cut generator which is called at each pass at the root node, and at each node of the tree whose depth is multiple of 2. When this cut generator is called to cut off a fractional point \bar{x} , it returns all stored cuts which are violated by \bar{x} , if any exist. We label this variant CBCRANK1.

We report results for easy, medium and hard instances. Easy instances are those which are solved in less than 10 seconds by CBCBASIC. Medium instances are those which are solved in less than 30 minutes by CBCBASIC and are not easy. Hard instances are the remaining ones, i.e. instances which are not solved within 30 minutes. For the latter, we are mainly interested in the number of nodes and amount of integrality gap closed; for easy and medium instances, CPU time and number of nodes are the most interesting figures for practical purposes. A summary of the results is reported in Table 4; all methods solve to optimality 28 instances out of 43. CBCBASIC is superior only on the Easy instances, where spending time to generate more split cuts results in a slow down of the solution process. On Medium instances, our approach is able to cut down both the number of nodes and computational times: CBCFASTRS yields average CPU time savings of 10% with respect to CBCBASIC, whereas with CBCRANK1 the savings increase to 23%. This can be explained by the significant reduction in the number of nodes. Finally, on Hard instances both CBCFASTRS and CBCRANK1 are able to process fewer nodes than CBCBASIC in the 30 minutes time limit, still they close more integrality gap. In particular, CBCRANK1 processes 25% fewer nodes and closes 1.8% more gap, and ranks first among the tested methods in terms of average gap per node, suggesting that in practice it is a more effective algorithm. Finding the best way to employ these families of rank-1 cuts that can be generated efficiently in a Branch-and-Cut framework is a subject for future research.

Instances	CBCBASIC				CBCFASTRS				CBCRANK1			
	Gap		Nodes	Time [sec]	Gap		Nodes	Time [sec]	Gap		Nodes	Time [sec]
	Root	Tot.			Root	Tot.			Root	Tot.		
Easy	78.2%	100.0%	68.2	2.4	86.3%	100.0%	55.5	3.7	86.3%	100.0%	61.2	3.9
Medium	51.2%	100.0%	5443.6	69.7	52.6%	100.0%	4139.3	62.6	52.6%	100.0%	3088.9	53.9
Hard	26.4%	46.8%	21072.9	1800.0	27.0%	49.4%	18445.6	1800.0	27.0%	48.6%	16025.7	1800.0

Table 4: Summary of the results in a Branch-and-Cut framework.

References

- [1] M. Ajtai. The shortest vector problem in l_2 is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, Dallas, TX, 1998.
- [2] K. Andersen, G. Cornuéjols, and Y. Li. Reduce-and-split cuts: Improving the performance of mixed integer Gomory cuts. *Management Science*, 51(11):1720–1732, 2005.
- [3] E. Balas. Intersection cuts - a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.
- [4] E. Balas and A. Saxena. Optimizing over the split closure. *Mathematical Programming*, 113(2):219–240, 2008.
- [5] P. Bonami, G. Cornuéjols, S. Dash, M. Fischetti, and A. Lodi. Projected Chvátal-Gomory cuts for mixed integer linear programs. *Mathematical Programming*, 113:241–257, 2008.
- [6] COIN-OR Branch-and-Cut. <https://projects.coin-or.org/Cbc/>
- [7] COIN-OR Cut Generation Library. <https://projects.coin-or.org/Cgl/>
- [8] W. Cook, R. Kannan, and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47:155–174, 1990.
- [9] G. Cornuéjols, L. Liberti, and G. Nannicini. Improved strategies for branching on general disjunctions. Technical Report 2071, Optimization Online, 2008.
- [10] S. Dash, O. Günlük, and A. Lodi. On the MIR closure of polyhedra. In M. Fischetti and D. P. Williamson, editors, *Proceedings of the 12th IPCO Conference*, volume 4513 of *Lecture Notes in Computer Science*, pages 337–351. Springer Berlin, 2007.
- [11] M. Fischetti, A. Lodi, and A. Tramontani. On the separation of disjunctive cuts. *Mathematical Programming*, 2009. to appear.
- [12] R. E. Gomory. An algorithm for the mixed-integer problem. Technical Report RM-2597, RAND Corporation, 1960.
- [13] F. Margot. Testing cut generators for mixed-integer linear programming. *Mathematical Programming Computation*, 1(1):69–95, 2009.
- [14] G. Nemhauser and L. Wolsey. A recursive procedure for generating all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.
- [15] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, New York, 1998.