Combining Lift-and-Project and Reduce-and-Split

Egon Balas*
Tepper School of Business, Carnegie Mellon University, PA
Email: eb17@andrew.cmu.edu

Gérard Cornuéjols†
Tepper School of Business, Carnegie Mellon University, PA
Email: gc0v@andrew.cmu.edu

Tamás Kis‡
Computer and Automation Research Institute,
Hungarian Academy of Sciences, Hungary
Email: tamas.kis@sztaki.hu

Giacomo Nannicini§
Tepper School of Business, Carnegie Mellon University, PA
Email: nannicin@andrew.cmu.edu

July 17, 2011

Abstract

Split cuts constitute a class of cutting planes that has been successfully employed by the majority of Branch-and-Cut solvers for Mixed Integer Linear Programs. Given a basis of the LP relaxation and a split disjunction, the corresponding split cut can be computed with a closed form expression. In this paper, we use the Lift-and-Project framework [11] to provide the basis, and the Reduce-and-Split algorithm [19] to compute the split disjunction. We propose a cut generation algorithm that starts from a Gomory Mixed Integer cut and alternates between Lift-and-Project and Reduce-and-Split in order to strengthen it. This paper has two main contributions. First, we extend the Balas and Perregaard procedure for strengthening cuts arising from split disjunctions involving one variable, to split disjunctions on multiple variables. Second, we apply the Reduce-and-Split algorithm to non-optimal bases of the LP relaxation. We provide detailed computational testing of the proposed methods.
Keywords: Integer Programming, Computational Analysis, Branch-and-Cut, Lift-and-Project.

*Supported by NSF grant CMMI1024554 and ONR grant N00014-09-1-0033.
†Supported by NSF grant CMMI1024554 and ONR grant N00014-09-1-0033.
‡Supported by the Hungarian Research Fund OTKA K76810.
§Supported by an IBM Fellowship.

1 Introduction

Mixed Integer Linear Programs (MILPs), i.e. mathematical programs with linear objective and constraints and both continuous and integer variables, arise in a number of real-world applications, and their solution is therefore of great practical interest. The most successful softwares for solving general MILPs utilize a Branch-and-Cut algorithm, which combines cutting planes and Branch-and-Bound. Several classes of cutting planes used by these softwares, such as Gomory Mixed Integer (GMI) cuts [20], Mixed Integer Rounding (MIR) cuts [23] and Lift-and-Project cuts [8], fall into the category of split cuts [18], that is, disjunctive cuts derived from two parallel hyperplanes. It was shown in [4] that every split cut can be generated as an intersection cut [5] from an appropriate choice of a basis of the LP relaxation and a split disjunction. The advantage of generating split cuts as intersection cuts is that we can use closed form expressions, without having to resort to disjunctive programming [6]. In this paper, we propose a split cut generation procedure that is based on Lift-and-Project [9, 11] and Reduce-and-Split [3, 19]. In particular, we use the former to select a basis of the LP relaxation, and the latter to compute a split disjunction.

Lift-and-Project (L&P) cuts have been successfully used in the Branch-and-Cut framework since the 90s [9]. A significant improvement in their practical performance came a few years later, when a procedure to generate L&P cuts without solving the higher-dimensional Cut Generating Linear Program (CGLP) was introduced by Balas and Perregaard [11]. This procedure starts with a split cut arising from a violated two-term disjunction involving a single variable and the optimal basis of the LP relaxation (in other words, a GMI cut), and mimicks the solution of the CGLP by performing pivots in the original simplex tableau. The procedure yields a new (possibly infeasible) basis, from which a stronger cut than the initial GMI cut can be generated. This procedure has been incorporated into commercial solvers like Xpress-MP [24], MOPS [25], and several versions of it have been implemented in the open source project COIN-OR Cgl [16]. One of the main contributions of this paper consists in an extension of this procedure to split cuts arising from general split disjunctions, i.e. any violated two-term disjunction involving an integral linear combination of integer variables. This yields a procedure that, given any split disjunction and any basis, produces a different basis that gives rise to a stronger cut.

In order to apply this extended L&P procedure, we need a method for generating an initial split disjunction. We use the Reduce-and-Split (RS) algorithm for this purpose. RS, first introduced in [3] and then revisited in [19], is a cut generation algorithm that starts from an optimal LP basis and a split disjunction on one variable, and computes a split disjunction involving several variables that (heuristically) yields a better cut. Therefore, we have an algorithm to produce split disjunctions, which can be used to initialize the L&P procedure.

Another contribution of this paper is that we apply RS on non-optimal, possibly infeasible, tableaux. As a consequence, we have a procedure that, given any split disjunction and any basis, produces a new, often better split disjunction for cut generation. Thus, we can alternate between the two procedures introduced in this paper, and iteratively change both the basis and the split disjunction from which a split cut is generated.

We perform extensive computational experiments on a set of benchmark MILPs to assess the effectiveness of our ideas. Our computational results show that, within a Cut-and-Branch framework, the combination of the two cut generation algorithms yields stronger cutting planes than L&P or RS alone. We obtain the best results by alternating between the two more than once.

The rest of this paper is organized as follows. In Section 2 we introduce our notation and provide the necessary theoretical background. In Section 3 we review in more detail the Lift-and-Project procedure introduced in [11], and extend it to general split disjunctions. Section 4 reviews the Reduce-and-Split method, and discusses its application on non-optimal bases of the LP relaxation. In Section 5 we describe our cut generation algorithm, which alternates between the Lift-and-Project and the Reduce-and-Split procedures. Section 6 presents an extensive computational evaluation. Section 7 concludes the paper. Detailed tables of results can be found in the Appendix.

2 Notation and preliminaries

We are considering a MILP of the form:

$$\left.\begin{array}{rrcl} \min & c^\top x & & \\ & Ax & \geq & b \\ & x & \geq & 0 \\ \forall j \in N_I & x_j & \in & \mathbb{Z}, \end{array}\right\} \qquad \text{(MILP)}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $N := \{1, \ldots, n\}$, $N_I := \{1, \ldots, p\}$ with $p \leq n$, and where upper bounding constraints are subsumed by $Ax \geq b$. In the sequel (LP) will stand for the linear programming relaxation of (MILP). A split cut for (MILP) is a valid inequality derived from a disjunction of the form

$$\pi x \leq \pi_0 \quad \vee \quad \pi x \geq \pi_0 + 1, \tag{1}$$

where π_j is integer for $j \in N_I$, $\pi_j = 0$ for $j \in N \setminus N_I$, and π_0 is an integer whose value depends on the fractional point we want to cut off. For a given fractional point $\bar{x}$, π_0 is chosen so as to have

$$\pi_0 < \pi\bar{x} < \pi_0 + 1, \tag{2}$$

which yields $\pi_0 = \lfloor \pi\bar{x} \rfloor$. If $\bar{x}$ is a basic solution to (LP) such that $\bar{x}_k$ is fractional and $k \in N_I$, then

$$x_k \leq \lfloor \bar{x}_k \rfloor \quad \vee \quad x_k \geq \lfloor \bar{x}_k \rfloor + 1 \tag{3}$$

is an *elementary disjunction* and it is a special case of (1).

A GMI cut from (1) (or from (3)) can be can be derived as follows. Rewrite (LP) in standard form:

$$\left.\begin{array}{rrcl} \min & c^\top x & & \\ & (A, -I)x & = & b \\ & x & \geq & 0 \end{array}\right\} \qquad \text{(LP)}_s$$

where $x \in \mathbb{R}^{n+m}$ and the last m components are surplus variables. Let $\bar{x}$ be a basic solution, $B(\bar{x})$ the set of indices of basic variables, and $J(\bar{x}) = N \setminus B(\bar{x})$ the set of nonbasic variables. Then, the corresponding simplex tableau can be written as:

$$x_i = \bar{x}_i - \sum_{j\in J(\bar{x})} \bar{a}_{ij} x_j \quad \forall i \in B(\bar{x}). \tag{4}$$

Let $B_I(\bar{x}) = B(\bar{x}) \cap N_I$, $J_I(\bar{x}) = J(\bar{x}) \cap N_I$, $J_C(\bar{x}) = J(\bar{x}) \setminus N_I$ be the sets of integer basic variables, integer nonbasic variables and continuous nonbasic variables, respectively. Consider a linear combination with integer coefficients π_i of those rows of (4) where

$i \in B_I(\bar{x})$:

$$\sum_{i \in B_I(\bar{x})} \pi_i x_i = \hat{x} - \sum_{j \in J(\bar{x})} \hat{a}_j x_j, \tag{5}$$

where

$$\begin{array}{rcl} \hat{x} & = & \sum_{i \in B_I(\bar{x})} \pi_i \bar{x}_i \\ \hat{a}_j & = & \sum_{i \in B_I(\bar{x})} \pi_i \bar{a}_{ij} \text{ for } j \in J(\bar{x}). \end{array} \tag{6}$$

Let $\pi_0 = \lfloor \hat{x} \rfloor$, and define $f_0 = \hat{x} - \pi_0$, $f_j = \hat{a}_j - \lfloor \hat{a}_j \rfloor$ for all $j \in J$. If $\hat{x} \notin \mathbb{Z}$, we can derive from (5) the following valid inequality for (MILP):

$$\begin{aligned} &\sum_{j \in J_I(\bar{x}): f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{j \in J_I(\bar{x}): f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j + \\ &\sum_{j \in J_C(\bar{x}): \hat{a}_j \geq 0} \frac{\hat{a}_j}{f_0} x_j - \sum_{j \in J_C(\bar{x}): \hat{a}_j < 0} \frac{\hat{a}_j}{1 - f_0} x_j \geq 1. \end{aligned} \tag{7}$$

This inequality is the GMI cut associated with the equation obtained through the row multipliers π_i; its validity is shown in [20]. Choosing $\pi_k = 1$, $\pi_i = 0 \ \forall i \neq k$ yields the GMI cut from row k of (4).

The derivation of (7) from (5) proceeds as follows. Consider the disjunction obtained by substituting the right hand side of (5) into (1):

$$\hat{x} - \sum_{j \in J(\bar{x})} \hat{a}_j x_j \leq \pi_0 \quad \vee \quad \hat{x} - \sum_{j \in J(\bar{x})} \hat{a}_j x_j \geq \pi_0 + 1.$$

Rewriting this gives

$$\sum_{j \in J(\bar{x})} \hat{a}_j x_j \geq f_0 \quad \vee \quad \sum_{j \in J(\bar{x})} (-\hat{a}_j) x_j \geq 1 - f_0.$$

The disjunctive cut obtained from the latter disjunction is

$$\sum_{j \in J(\bar{x}): \hat{a}_j \geq 0} \frac{\hat{a}_j}{f_0} x_j - \sum_{j \in J(\bar{x}): \hat{a}_j < 0} \frac{\hat{a}_j}{1 - f_0} x_j \geq 1. \tag{8}$$

By applying the *integer modularization procedure* of Balas [6] and Balas and Jeroslow [7] to (8), we obtain (7).

3 Lift-and-Project on general disjunctions

We start this section by reviewing the original Lift-and-Project procedure, then we introduce one of the main contributions of this paper, namely the extension of the Lift-and-Project procedure on the original simplex tableau to general two-term disjunctions.

3.1 Review of Lift-and-Project

Lift-and-Project cuts were introduced in [8], and [9]. These are cuts obtained from a two-term disjunction of the form

$$\begin{pmatrix} Ax \geq b \\ x \geq 0 \\ -x_k \geq 0 \end{pmatrix} \vee \begin{pmatrix} Ax \geq b \\ x \geq 0 \\ x_k \geq 1 \end{pmatrix}$$

where $1 \leq k \leq p$ with $0 < \bar{x}_k < 1$. A lift-and-project cut from this disjunction is derived by solving the so-called *cut generating linear program* $(\mathrm{CGLP})_k$:

$$\begin{array}{llllllllllll}
\min & \alpha\bar{x} & - & \beta & & & & & & & & \\
 & \alpha & & & - & uA & + & u_0 e_k & & & \geq & 0 \\
 & \alpha & & & - & & & & - & vA \; - \; v_0 e_k & \geq & 0 \\
 & & & \beta & - & ub & & & & & = & 0 \\
 & & & \beta & & & & & - & vb \; - \; v_0 & = & 0 \\
 & & & & & \sum_{i=1}^m u_i & + & u_0 & + & \sum_{i=1}^m v_i \; + \; v_0 & = & 1 \\
 & & & & & & & & & u, u_0, v, v_0 & \geq & 0.
\end{array}$$

Here, e_k is the k-th unit vector. The objective function maximizes the violation of the cut $\alpha x \geq \beta$ in point $\bar{x}$. The last equation is a normalization constraint which ensures that $(\mathrm{CGLP})_k$ always has a finite optimum.

Balas and Perregard observed that $(\mathrm{CGLP})_k$ can be solved to optimality in the original simplex tableau [11]. In fact, they describe the precise correspondence between the *feasible bases* of $(\mathrm{CGLP})_k$ and the bases of $(\mathrm{LP})_s$. Notice that in this correspondence, the bases of $(\mathrm{LP})_s$ are generally infeasible, i.e., in the corresponding LP solution, x may have negative coordinates.

The L&P procedure is illustrated in Fig. 1, where the basic solution $\bar{x}$ is to be cut off, but the corresponding cut $\alpha x \geq \alpha_0$ is weaker than the one that can be derived after pivoting to $\bar{x}'$ and deriving the cut $\alpha' x \geq \alpha'_0$ from that (infeasible) basic solution. The rays r^i for $i = 1, 2$ correspond to the non-basic columns of the simplex tableau in the basic solution $\bar{x}$, whereas r^1 and r^3 are those in the basis corresponding to $\bar{x}'$.

3.2 Lift-and-Project applied to general split disjunctions

For the sake of compact notation, we define $(\tilde{A}, \tilde{b})$ as

$$\left(\tilde{A}, \tilde{b}\right) := \begin{pmatrix} A & b \\ I & 0 \end{pmatrix},$$

where I is the $n \times n$ identity matrix. In order to apply the Lift-and-Project procedure to a disjunction of the form (1), one could simply formulate the (CGLP) corresponding to

$$\begin{pmatrix} \tilde{A}x & \geq & \tilde{b} \\ -\pi x & \geq & -\pi_0 \end{pmatrix} \bigvee \begin{pmatrix} \tilde{A}x & \geq & \tilde{b} \\ \pi x & \geq & \pi_0 + 1 \end{pmatrix}. \tag{9}$$

However, in order to take full advantage of the correspondence between the (CGLP) and the (LP) established in [11], it will be preferable to introduce a new integer variable x_{n+m+1} to represent the difference between πx and π_0:

$$\pi x - x_{n+m+1} = \pi_0 \tag{10}$$

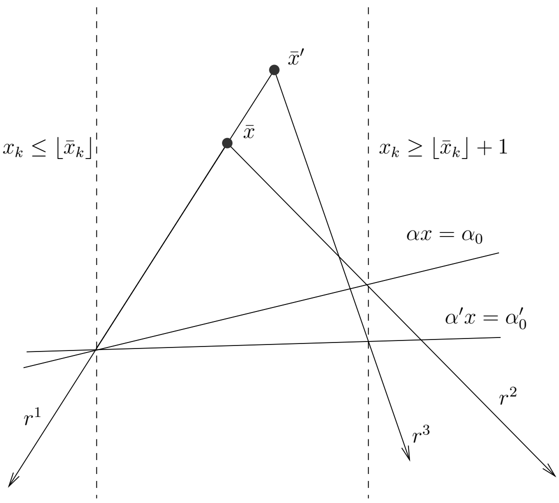

Figure 1: Illustration of the Lift-and-Project procedure.

Notice that, since πx is integer in any feasible solution of (MILP) and π_0 is integer, x_{n+m+1} has to be integer as well. Moreover, if $x_{n+m+1} \le 0$, then $\pi x \le \pi_0$, while if $x_{n+m+1} \ge 1$, then $\pi x \ge \pi_0 + 1$, as desired. Using the new variable, we can rewrite (9):

$$\left(\begin{array}{rcl} \tilde{A}x & \ge & \tilde{b} \\ \pi x - x_{n+m+1} & = & \pi_0 \\ x_{n+m+1} & \le & 0 \end{array} \right) \bigvee \left(\begin{array}{rcl} \tilde{A}x & \ge & \tilde{b} \\ \pi x - x_{n+m+1} & = & \pi_0 \\ x_{n+m+1} & \ge & 1 \end{array} \right). \tag{11}$$

The important difference from the previous applications of the Lift-and-Project procedure to single rows (4) of the simplex tableau is the following. The equation (10) is constructed in order to derive a cut from it. Once the cut is derived, the equation is no longer needed and therefore it is discarded, along with the variable x_{n+m+1}. On the other hand, the variable x_{n+m+1}, and its expression in terms of the current nonbasic variables, is needed throughout the pivoting process carried out in order to (implicitly) optimize the CGLP. Thus, we have to add a new row to the optimal (LP) tableau and keep it until the cut is optimized. This could be done by simply adding the equation $\pi x - x_{n+m+1} = \pi_0$ to the constraint set of (MILP), and then computing the amended simplex tableau corresponding to the current basis. Instead, one can derive the new row as a closed form expression.

3.1 Proposition
Let (A_B, A_J) be the partition of $(A, -I)$ into basic and nonbasic columns. Then the expression for $x_{n+m+1} = \pi x - \pi_0$ in terms of the nonbasic variables is

$$x_{n+m+1} + (\pi_B A_B^{-1} A_J - \pi_J) x_J = (\pi_B A_B^{-1}) b - \pi_0 \tag{12}$$

Proof. The simplex tableau corresponding to the basis indexed by B is

$$x_B + A_B^{-1} A_J x_J = A_B^{-1} b.$$

If $\pi = (\pi_B, \pi_J)$ and $\pi_j = 0$ for all $j \in N \setminus N_I$, then $\pi x - x_{n+m+1} = \pi_0$ can be written as

$$-x_{n+m+1} + \pi_B x_B + \pi_J x_J = \pi_0.$$

Appending this equation to $(A_B, A_J)x = b$ gives

$$\begin{array}{ccccccc} A_B x_B & & & + & A_J x_J & = & b \\ \pi_B x_B & - & x_{n+m+1} & + & \pi_J x_J & = & \pi_0. \end{array} \tag{13}$$

The inverse of the $(m+1) \times (m+1)$ matrix $\begin{pmatrix} A_B & 0 \\ \pi_B & -1 \end{pmatrix}$ is $\begin{pmatrix} A_B^{-1} & 0 \\ \pi_B A_B^{-1} & -1 \end{pmatrix}$.

Multiplying (13) with this augmented basis inverse gives

$$\begin{array}{ccccc} x_B & & + & (A_B^{-1} A_J) x_J & = & A_B^{-1} b \\ & x_{n+m+1} & + & (\pi_B A_B^{-1} A_J - \pi_J) x_J & = & \pi_B A_B^{-1} b - \pi_0 \end{array}$$

□

The new source row (12) could of course be used to directly generate a generalized GMI cut; instead, we apply to it the L&P procedure of [11], or one of its variants discussed in [12] in order to obtain a stronger cut. Such a cut will be valid throughout the search tree in case of a mixed 0-1 program, but only at the descendants of the current search tree node for a general mixed integer program (see [10]).

3.3 Implementation of the generalized Lift-and-project procedure

It is not too difficult to modify any implementation of the L&P cut generation procedure that works on the simplex tableau and strengthens cuts derived from a disjunction (3), so that it can strengthen split cuts derived from a more general split disjunction (1). Namely, the L&P procedure must have a subroutine to extract the source row from the simplex tableau before pivoting and after each pivot. This is usually done by using the basis inverse, which is typically readily available: most Branch-and-Cut (or Cut-and-Branch) solvers use the revised dual simplex method, which maintains the basis inverse rather than the full simplex tableau. It suffices to modify this subroutine so that it computes the source row using (12). This can be implemented rather efficiently using the standard Ftran or Btran subroutines, available in many commercial and free state-of-the-art LP solvers.

4 Reduce-and-Split from non-optimal bases

As in Section 3, we first recall the basic concepts of Reduce-and-Split, then we discuss our contribution: the application of Reduce-and-Split on non-optimal bases.

4.1 Review of Reduce-and-Split

The idea of looking for a linear combination (5) of rows of the simplex tableau (4) to generate strong cutting planes is not new in the integer programming literature: see e.g. [3, 14, 19]. As discussed in Section 2, every equation (5) such that $\sum_{i \in B_I(\bar{x})} \pi_i \bar{x}_i$ is fractional yields a valid GMI cut. Here, $\bar{x}$ need not be an optimal solution to (LP);

however, this is the only case that is typically studied in the literature. In the next section we consider the case where $\bar{x}$ is basic but not optimal for (LP). In particular, our discussion focuses on the case where $\bar{x}$ is a basic solution for a L&P tableau, i.e. a (possibly primal infeasible) tableau obtained by pivoting following the L&P procedure.

We now review the RS algorithm, as given in [19]. Let $\bar{x}$ be the optimal solution to (LP). RS first determines a working set of continuous nonbasic columns $J_W \subset J_C(\bar{x})$, then generates an integral combination (5) of the rows of the simplex tableau corresponding to the basic variables in $B_I(\bar{x})$ by minimizing:

$$\min_{\pi \in \mathbb{Z}^{|B_I(\bar{x})|}} \|(\hat{a}_j)_{j \in J_W}\|_2, \tag{14}$$

where $\hat{a}_j$ is defined as in (6). Observe that the linear combination (5) can involve rows with an integer valued basic variable, as long as $\sum_{i \in B_I(\bar{x})} \pi_i \bar{x}_i$ is fractional. The minimization problem (14) yields row multipliers π_i from which we derive (7). As can be seen from (7), small $\hat{a}_j$ on continuous nonbasic columns should yield good (i.e. small) cut coefficients on the corresponding variables. Note that our aim is to improve the cut coefficients on continuous variables only. The reason for focusing on continuous variables only is that the cut coefficients on integer variables are much more difficult to control, because of the modular arithmetic involved in their expression (see (7)). (14) is solved by relaxing integrality on π, determining the optimal continuous multipliers (imposing an additional normalization constraint to avoid the all zero solution), then rounding the fractional components π to the nearest integer. In [19] it is experimentally shown that variables with small reduced cost are good candidates for the set J_W, as they yield cuts which close a larger integrality gap in practice. Furthermore, instead of considering all rows whose corresponding basic variable is in $B_I(\bar{x})$, it is shown that better results can be obtained by considering only a subset of carefully chosen rows. Since we are interested in finding a linear combination that yields small coefficients, the chosen rows should ideally be linearly dependent or almost.

In its default configuration, the Reduce-and-Split cut generation algorithm proceeds as follows: for each row r of the simplex tableau with an integer basic variable x_k, a subset of columns $J_W \subset J_C(\bar{x})$ and a subset of rows other than r with a basic integer variable is chosen. Then, a linear combination of these rows is sought using the procedure outlined above. The normalization condition to avoid the all zero solution to (14) consists in requiring $\pi_k = 1$. This loop is iterated several times using different strategies to select J_W and the set of rows. This is the basic variant of the Reduce-and-Split algorithm: we refer to [19] for a thorough discussion.

The geometric interpretation is as follows. RS keeps the basis $B(\bar{x})$ fixed, and tries to modify the split disjunction (1) in order to obtain a cut with stronger coefficients. This is exemplified in Figure 2: the elementary disjunction $x_k \leq \lfloor \bar{x}_k \rfloor \vee x_k \geq \lfloor \bar{x}_k \rfloor + 1$, which yields the cut $\alpha x \geq \alpha_0$, is modified to obtain a stronger cut $\alpha' x \geq \alpha'_0$ (from disjunction $\pi x \leq \pi_0 \vee \pi x \geq \pi_0 + 1$). Again, the rays r^1 and r^2 correspond to the non-basic columns of the simplex tableau with basic solution $\bar{x}$.

4.2 Modifications to Reduce-and-Split

In Section 3 we proposed a method to start with *any* split disjunction, and modify the basis via L&P to obtain a stronger cut. What we want to do now is to use the basis computed by L&P, and modify ("tilt") the split disjunction to derive a better cut.

A problem arises: a cut derived from a non-optimal basis of (LP) will certainly be

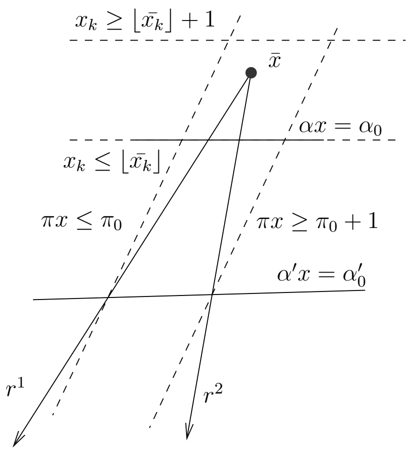

Figure 2: Illustration of the Reduce-and-Split procedure.

valid, but how do we make sure that it will be violated by the point that we want to cut off? To show why such a cut might not be violated, we need to introduce some notation. Let $\bar{x}$ be the optimal solution to (LP), where the corresponding optimal tableau $\bar{A}$ has elements $\bar{a}_{ij}$. Let $\bar{x}'$ be the basic solution associated with the tableau $\bar{A}'$ (with elements $\bar{a}'_{ij}$) obtained by applying L&P starting from $\bar{x}$. A GMI obtained from tableau $\bar{A}$ has the form

$$\sum_{j \in J(\bar{x})} \alpha_j x_j \geq \alpha_0,$$

with $\alpha_j \geq 0, \alpha_0 > 0$. Since $\bar{x}_j = 0 \ \forall j \in J(\bar{x})$, this cut is violated by $\bar{x}$. On the other hand, a split cut obtained as a GMI cut from tableau $\bar{A}'$ has the form:

$$\sum_{j \in J(\bar{x}')} \alpha'_j x_j \geq \alpha'_0,$$

and cuts off $\bar{x}'$ but is *not* necessarily violated by $\bar{x}$. Indeed, $\bar{x}_j = 0 \ \forall j \in J(\bar{x}') \cap J(\bar{x})$ but $\bar{x}_j \geq 0 \ \forall j \in J(\bar{x}') \cap B(\bar{x})$, therefore the left hand side may be > 0 at $\bar{x}$. The cut will be violated if and only if $\sum_{j \in J(\bar{x}') \cap B(\bar{x})} \alpha'_j \bar{x}_j < \alpha'_0$. This suggests that we should aim for small (hopefully zero) cut coefficients on the columns with indices in $J(\bar{x}') \cap B(\bar{x})$. In Figure 3, we picture an example of a non violated cut: the L&P cut obtained from the new basic solution $\bar{x}'$ and the initial disjunction $x_k \leq \lfloor \bar{x}_k \rfloor \vee x_k \geq \lfloor \bar{x}_k \rfloor + 1$ cuts off $\bar{x}$ by construction, but as soon as the disjunction is modified, we are only guaranteed to cut off $\bar{x}'$ (as shown by the cut $\alpha' x \geq \alpha'_0$).

In order to generate cuts from $\bar{A}'$ that are likely to cut off $\bar{x}$, we modify the RS algorithm as follows. Let $B^* = J(\bar{x}') \cap B(\bar{x})$ be the set of variables which are basic in the optimal LP tableau but are nonbasic in the L&P tableau on which we apply RS. Given $J_W \subset J_C(\bar{x}')$ (e.g. using one of the techniques described in [19]) and scalars $\sigma_j > 0 \ \forall j \in J_W \cup B^*$, we compute:

$$\min_{\pi \in \mathbb{Z}^{|B_I(\bar{x}')|}} \| \sum_{i \in B_I(\bar{x}')} \pi_i d'_i \|_2, \tag{15}$$

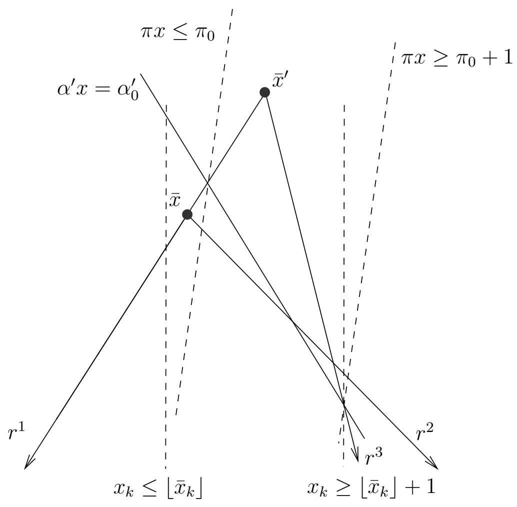

Figure 3: A Reduce-and-Split cut from the disjunction $\pi x \leq \pi_0 \vee \pi x \geq \pi_0 + 1$, obtained from the tableau associated with the basic solution $\bar{x}'$, that does not cut off the point $\bar{x}$.

where $d'_i = (\sigma_j \bar{a}'_{ij})_{j \in J_W \cup B^*}$; in other words, d'_i are rows of a submatrix of $\bar{A}'$ (corresponding to the set of columns $J_W \cup B^*$), where each column is rescaled with multipliers σ_j. The effect of these multipliers is to modify the importance of the columns when determining π that minimizes the norm in (15), by increasing it (if σ_j is large) or decreasing it (if σ_j is small). Observe from (15) that we try to reduce the coefficients of (7) on all columns with indices in B^*: for continuous variables in B^*, this should yield a reduction on the resulting cut coefficient; for integer variables, the end result is not so clear because of the integer modularization (cf. end of Section 2), but $\hat{a}_j = 0$ always results in a zero cut coefficient in the corresponding column. Since we want to reduce the coefficients relative to B^* as much as possible, we set $\sigma_j = 2 \; \forall j \in \{i \in B^* : \bar{x}_i > 0\}$, and $\sigma_j = 1$ otherwise. This prioritizes the reduction of the source row coefficients on the variables with indices B^* such that the corresponding component in $\bar{x}$ is nonzero. We experimentally tried other strategies to choose σ_j, but this simple idea turned out to work well in practice. A discussion is given in Section 6. The rest of the RS algorithm is unmodified.

Note that this method offers no guarantee of finding a violated cut, nor does it guarantee to increase the cut violation with respect to the cut associated with the original source row. However, RS has proven to generate strong cuts in practice, therefore we are interested in testing whether it is equally effective if applied to non-optimal bases of (LP), and in particular those generated by L&P.

5 Combining Lift-and-Project and Reduce-and-Split

We combined the methods described in Section 3 and Section 4 into a single cut generation algorithm, that alternates between the L&P and the RS cut improvement procedures.

Our cut generation algorithm always starts with determining the set of basic integer variables that have a fractional value (at least 10^{-2} away from an integer) in the current solution to the LP relaxation; the corresponding (elementary) disjunctions are processed by nonincreasing violation (i.e. those with a violation closest to 0.5 are processed first), until a given maximum number of cuts M is generated, or there are no more violated disjunctions available. In this paper, we always use $M = 50$. This method for processing elementary disjunction is taken from [12]. Recall that these disjunctions give rise to the traditional GMI cuts. Then, we iteratively modify each GMI cut, changing either the underlying disjunction (through RS), or the underlying basis of the LP relaxation (through L&P). One parameter of our cut generation algorithm is the maximum number η of cut improvement steps that we want to perform, i.e. the number of times that we alternate between L&P and RS. When $\eta = 0$, we use the initial GMI cuts. Another parameter `start` is whether to apply L&P or RS at the first cut modification step. For instance, if `start` = L&P and $\eta = 3$, the GMI cuts are strengthened by L&P, then the underlying disjunction is modified by RS (using the simplex tableau computed by L&P), finally we change the basis again (using the new disjunction) with L&P. After each cut improvement step, we check the outcome of the routine (L&P or RS). If the routine fails, either because it could not improve the cut (i.e. L&P could not perform improving pivots, or RS could not find a disjunction that improves the cut coefficients) or because numerical problems were detected, then the improvement procedure for that particular cut is stopped, and we generate the cut computed at the previous iteration. For instance, for `start` = L&P and $\eta = 3$, if the RS algorithm at step 2 fails, we generate the L&P cut obtained at step 1. If the cut computed at the previous iteration does not satisfy the numerical requirements, then the cut is discarded and we restart the process with another elementary disjunction. Note that if the first improvement step fails, then we simply generate the initial GMI cut.

This method is designed to be balanced between L&P and RS: since we always start with the same M elementary disjunction, we can compare the effects of starting with L&P or with RS. Note that this method is based on simple GMI cuts, which have proved to be one of the most effective and reliable general-purpose classes of cutting planes: our method tries to improve on the GMI cuts, but in case of failure, we revert back to the GMI cuts.

6 Computational experiments

The cut generation algorithms presented in this paper were implemented in C++ within the COIN-OR Cgl [16] framework. Our L&P generator is a modification of the existing CglLandP generator [12]; likewise, the RS implementation is based on the existing CglRedSplit2 generator [19]. The CglLandP generator employs advanced simplex algorithm functions, and for this reason it only works with the COIN-OR Clp [17] LP solver. Traditional GMI cuts were generated using the CglLandP generator, setting the maximum number of pivots to zero. We used Cplex 12.1 [21] to perform instance preprocessing and Branch-and-Bound. More details on the interaction between Clp and Cplex are given in Section 6.1.

Our set of test instances is a subset of the mixed-integer instances in the union of MIPLIB3 [13], MIPLIB2003 [2] and the set of test instances of the University of Bologna available from `http://plato.asu.edu/ftp/unibo/`. We selected all mixed-integer instances such that the LP has fewer than 500 000 nonzero elements, and such that we were able to generate 10 rounds of cutting planes with the original CglLandP generator in less than 20 minutes. The instance `bell5` was not selected because of its poor numerical properties, which made computational experiments give erratic results, thus producing noise in the data instead of useful information. We divide the instances in three difficulty classes, depending on the performance of our Cut-and-Branch algorithm (see Section 6.1) with cutting planes generated by the original CglLandP. Instances are labeled Easy if they can be solved requiring less than one minute of CPU time and 1000 nodes; they are labeled Medium if they are not Easy but can be solved in less than 2 hours; they are Hard if they cannot be solved in 2 hours of total CPU time. A list of instances is given in Table 1. In *all* tests reported in this section, the value of the optimal solution is given to the solver as a cutoff value so that the time of discovery of integer solutions does not affect the size of the enumeration tree.

Easy	Medium	Hard
10teams	aflow30a	a1c1s1
blend2	arki001	aflow40b
dcmulti	bell3a	b1c1s1
dsbmip	gesa2	b2c1s1
egout	gesa2_o	bg512142
fiber	glass4	dano3mip
fixnet6	mas74	danoint
flugpl	mas76	dg012142
gen	misc07	mkc
gesa3	mod011	momentum1
gesa3_o	modglob	momentum2
khb05250	noswot	nsrand-ipx
misc06	pk1	opt1217
qnet1	pp08aCUTS	roll3000
qnet1_o	pp08a	set1ch
rentacar	qiu	swath
	rgn	timtab1
	rout	timtab2
	vpm1	tr12-30
	vpm2	

Table 1: List of test instances.

6.1 Cut-and-Branch

To assess the effectiveness of our cut generation procedure, and compare our cut generator to the traditional L&P and RS cuts, we implemented a Cut-and-Branch algorithm on top of Cplex [21] and Clp [17]. Recall that the L&P cut generator requires a simplex tableau in Clp format. However, we decided to employ Cplex instead of COIN-OR Cbc as Cut-and-Branch code because of its better reliability. Therefore, we proceed as follows: each problem instance is read and preprocessed by Cplex with default settings. The presolved reduced problem is then loaded with Clp, and cutting planes are generated for a maximum of 10 rounds or 20 minutes of CPU time. At each round of cut generation, we perform this sequence of operations. First, we generate at most 50 cuts, and add all of them to the LP formulation. Then, we check if any of the cutting planes generated at previous rounds (and subsequently removed from the LP) is violated by the current fractional point; if so, we add all such cuts to the LP. Finally, the LP is reoptimized, and all inactive cutting planes are removed. The LP formulation obtained after 10 rounds is loaded into Cplex, where another pass of presolve is executed before switching to Branch-and-Bound. To simulate a bare Branch-and-Bound algorithm within the Cplex environment, we apply the following settings:

- Cutting planes are disabled (`cutsfactor = 0`, and all cut generation algorithms manually disabled);
- Emphasis on proving optimality (`mipemphasis = bestbound`);
- Heuristics are disabled (`heurfreq = -1`, and all heuristics manually disabled);
- Absolute and relative integrality gap for optimality set to zero (`epgap = 0`, `epagap = 0`).

Constraint and integrality precision were set to 10^{-7}. All other parameters are left to their default value.

6.2 Parameters for cut generation

Our cut generation algorithm is described in Section 5, and has two main parameters: the maximum number η of cut improvement steps that we want to perform, and whether to apply L&P or RS at the first cut modification step. Additionally, both the L&P and the RS cut generators require some parameters to perform each improvement step. For L&P, the pivot selection rule is set to "most negative reduced cost", the maximum number of pivots is set to 10, and we do not apply the iterative modularization technique discussed in [12]. For RS, the maximum support of the disjunction is set to 5, the maximum 1-norm of the disjunction is set to 10, the column selection strategy (i.e. the choice of the set J_W) is set to "first 1/3 of the columns with smallest reduced cost", and the row selection strategy is set to "rows with smallest angle with respect to the source row in the space $J_W \cup J_I$" (the latter two parameters correspond to the strategies CS1, RS8 in [19]). These parameters were chosen for their performance based on the computational experiences reported in [12, 19]. Even though other values for the L&P and RS cut generators were tested, for space reason we only report results with this set of values. Our configuration of the L&P generator is very similar to the default parameters discussed in [12], whereas the RS configuration is different than in [19] because in that paper a large number of cuts is generated at each round, but here we want to generate at most 50 cuts per round to facilitate comparisons. In Section 6.4 we compare the performance of our parameter selection with the default values provided in the implementations described in [12, 19].

For the combined cut generation algorithm, we tested up to 6 cut improvement iterations, starting either with L&P or with RS. Each combination of parameters yields a different cut generator, which we label as L&P-η if L&P is applied first and we perform up to η improvement steps alternating between RS and L&P, or as RS-η if RS is applied first and we perform up to η improvement steps alternating between L&P and RS. Note that L&P-1 and RS-1 correspond to simple L&P and RS cuts respectively.

6.3 Results with Cut-and-Branch

We now report and discuss the results obtained within the framework presented in this section, for several cut generators. For each cut generator and each instance, we report: the amount of integrality gap closed at the root after 10 rounds of cut generation (*root gap %*), the CPU time required for cut generation which includes the running of the separation procedures as well as the repeated solutions of the node LPs (*cut time*), the number of generated cuts (*#cuts*), the amount of integrality gap closed at the end of the Cut-and-Branch algorithm (100% if optimality is proven within the time limit, $< 100\%$ if the two hours limit is hit) (*final gap %*), the number of enumerated nodes (*#nodes*), and the total CPU time required by Cut-and-Branch (*total time*). All times are measured in seconds. Detailed results can be found in Tables 6 through 11, whereas averages are given in Table 2. The average integrality gap and number of cuts are computed as arithmetic averages; the average CPU time and number of nodes are geometric averages (to deal with zero values, we added one to each value before computing the average, and subtracted one from the result). For comparison, we also report, in Tables 3 and 12, results obtained within the same framework using traditional GMI cuts from the optimal tableau.

Table 2: Average values for Tables 6 through 11.

		L&P						RS					
η	instances	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
	Easy	66.69	1.88	242.81	100.00	66.73	2.34	62.70	1.48	177.12	100.00	84.81	1.87
1	Medium	42.82	1.47	283.05	100.00	36783.44	27.97	40.89	0.88	247.45	100.00	27401.35	25.64
	Hard	31.67	17.73	362.95	63.99	491855.55	7200.02	27.07	9.02	328.26	65.86	529225.83	7200.03
	Easy	70.18	2.97	239.94	100.00	58.13	3.36	71.07	2.45	219.50	100.00	61.68	2.99
2	Medium	45.10	1.92	281.70	100.00	22605.34	25.17	45.34	1.67	275.60	100.00	22631.97	28.96
	Hard	30.17	29.68	368.58	64.76	488767.55	7200.03	28.76	17.71	334.21	63.17	527506.47	7200.03
	Easy	72.41	2.93	229.75	100.00	55.04	3.32	70.13	3.49	217.19	100.00	51.24	3.92
3	Medium	45.38	2.12	276.95	100.00	20928.22	26.49	46.98	1.99	266.55	100.00	25529.14	30.80
	Hard	30.53	33.18	371.89	65.42	505878.43	7151.24	30.42	27.89	334.95	63.16	511616.51	7200.03
	Easy	72.54	3.00	225.25	100.00	49.87	3.32	70.33	3.28	215.88	100.00	49.71	3.81
4	Medium	46.99	2.14	272.35	100.00	18443.68	26.46	47.13	2.03	267.80	100.00	21863.21	29.03
	Hard	31.32	34.45	368.53	65.59	488861.77	7080.50	30.30	29.63	332.11	63.14	489310.36	7200.04
	Easy	72.46	3.27	234.50	100.00	55.67	3.72	71.22	3.43	219.75	100.00	47.71	3.94
5	Medium	45.99	2.21	279.20	100.00	23520.81	29.86	47.06	2.08	268.65	100.00	22320.37	28.25
	Hard	31.12	38.10	362.42	65.85	501422.54	7200.03	29.40	29.93	335.42	63.48	520554.34	7200.04
	Easy	72.82	3.34	234.25	100.00	50.10	3.64	72.06	3.24	219.12	100.00	48.37	3.77
6	Medium	45.04	2.22	280.15	100.00	24948.13	30.80	46.88	2.10	268.60	100.00	23022.37	27.81
	Hard	32.06	37.66	369.74	65.29	468780.03	7200.04	29.99	29.76	329.26	62.84	482300.74	7200.03

Table 3: Average values for Table 12.

	GMI					
instances	root gap %	cut time	#cuts	final gap %	#nodes	total time
Easy	52.38	0.31	197.81	100.00	104.12	0.76
Medium	36.51	0.30	256.50	100.00	47544.46	27.02
Hard	26.45	1.86	328.63	64.33	571427.50	7200.03

In the integer programming community it is known that comparing the strength of different cut generators is a difficult task, especially when we are interested in the performance of Cut-and-Branch, and average values alone can be misleading. [22] proposes a framework for statistical tests. In this paper, in addition to reporting average values, we opted for a simple pairwise comparison between the 13 cut generators tested; in each comparison, we count the number of instances on which the first method is superior to the second one. The comparison is carried out on Medium and Hard instances, because on Easy instance we expect simple GMI cuts to outperform more powerful but time-consuming cut generation methods. Our comparison criteria are as follows: method A is superior to method B on a given instance if:

- Medium instance: A solves the instance in 10% less CPU time than B and the difference is at least 2 seconds;
- Hard instance: A closes at least $\rho = 1\%$ more integrality gap than B in the two hours, or A solves the instance within the time limit whereas B does not solve it and has more than 5% integrality gap left.

On Medium instances, we require a difference of at least 2 seconds of CPU time, because for small values the fluctuations may be due to other factors than the strength of the cutting planes. If no method is superior to the other, then the 2 methods have comparable strength on that instance. This pairwise comparison has been inconclusive on Medium instances, no method came out as a clear winner. In contrast, on Hard instances, the performance of L&P-η improves as η increases from 1 to 5. In fact, the number of times L&P-η outperforms the other cut generators increases from 75 to 118, whereas the number of occasions L&P-η is inferior to other methods decreases from 81 to 34. However, L&P-6 is inferior to L&P-5. In Table 4 we report results on Hard instances. We verified that as long as a "reasonable" value for ρ is used ($\rho \in [1, 10]$), the conclusions that can be drawn are essentially the same.

Table 4: Pairwise comparison of cut generators on Hard instances: number of instances on which the cut generation algorithm on the row is superior to the one on the column. Comparison with 1% absolute difference in final gap closed.

	L&P-1	L&P-2	L&P-3	L&P-4	L&P-5	L&P-6	RS-1	RS-2	RS-3	RS-4	RS-5	RS-6	GMI	Sum Sup.
L&P-1	-	4	3	3	2	5	6	8	12	7	9	9	7	75
L&P-2	6	-	4	4	2	4	7	8	8	9	10	8	7	77
L&P-3	10	7	-	6	4	5	8	11	10	9	9	10	10	99
L&P-4	9	9	7	-	4	5	7	13	10	10	10	12	9	105
L&P-5	9	9	7	6	-	8	8	11	13	11	14	13	9	118
L&P-6	9	9	6	5	2	-	8	12	13	11	12	14	10	111
RS-1	6	4	4	6	6	6	-	9	7	9	8	8	9	82
RS-2	6	4	3	2	1	3	4	-	7	7	6	8	8	59
RS-3	3	6	4	4	1	3	5	6	-	5	6	5	6	54
RS-4	7	5	4	1	2	2	6	8	7	-	8	5	8	63
RS-5	5	5	3	3	3	2	6	9	7	6	-	6	9	64
RS-6	5	4	3	1	2	3	5	6	6	5	5	-	7	52
GMI	6	7	5	4	5	5	5	7	6	7	8	8	-	73
Sum Inf.	81	73	53	45	34	51	75	108	106	96	105	106	99	

We have observed that the number of generated cuts is very similar for all methods, except on Easy instances: for Easy instances, RS-1 and GMI generate fewer cuts than other methods. However, our analysis focuses on Medium and Hard instances, and on these two problem classes all algorithms generate a similar number of cuts (the difference can be $\pm 10\%$). Therefore, we can compare the cut generators on equal footing.

We can see (Tables 2 and 3) that all proposed methods appear to be stronger than simple GMI cuts in terms of average gap closed at the root node, and in terms of number of nodes for Easy and Medium instances. On Hard instances, the gap closed by GMI cuts after Branch-and-Bound is comparable to some of the other tested method, even though GMI cuts appear to be weaker at the root. However, the number of nodes processed in two hours is larger for GMI cuts, which explains why the amount of gap closed after Branch-and-Bound is similar. GMI cuts are also the fastest method for Easy instances on average, and one of the fastest methods for Medium instances. On some hard problems, like `danoint` and `opt1217`, GMI cuts are still a good choice (see Table 12), due to a larger number of enumerated nodes within the time limit. This is to be expected: on some problems, investing CPU time in more expensive cut generation techniques does not pay off and GMI cuts come out as the winner, but in other cases, there is a large advantage to be gained by heavy cut generation.

We also observe that RS-1 performs very well on average on Medium and especially Hard instances, and appears to be stronger than L&P-1 by looking at Table 2 only; a more detailed analysis of the results reveals that its good average behaviour depends on some Hard and Medium instances on which RS-1 is considerably stronger than other cut generators (examples are `danoint`, `dg012412`, `opt1217`, `vpm1`), but on several other instances RS-1 is clearly weaker. This is well indicated by Table 4: RS-1 is always "inferior" and "superior" a large number of times. RS-η for $\eta > 1$ do not perform equally well as RS-1 on the few instances where RS-1 really dominates. On average $\eta > 1$ yields better results on the Easy and Medium instances in terms of nodes, but not in terms of CPU time, while being comparable on the Hard problems; the gap closed at the root node increases significantly on all problem classes.

The benefits for combining L&P and RS are much clearer when improving L&P cuts. Looking at Table 2, L&P-η with $\eta > 1$ is superior to L&P-1 in almost all respects: gap closed at the root (except on Hard instances for some values of η, for which we observe a slight decrease), number of nodes on the Easy and Medium instances, and gap closed after Branch-and-Bound on the Hard instances. On Medium instances, CPU times for L&P-2, L&P-3 and L&P-4 are better than for L&P-1. On Hard problems, Table 4 shows a clear trend when moving from $\eta = 1$ to $\eta = 6$: the cut generators are "superior" a larger number of times, and "inferior" a smaller number of times. The peak is reached by L&P-5: overall, this generator is the one which is "inferior" the smallest number of times, and is "superior" the largest number of times. Additionally, L&P-3 and L&P-4 are the only methods to solve one Hard instance (`aflow40b`) within the 2 hours time limit; L&P-4 requires 5240 seconds only for this task. To conclude, L&P-5 seems to be the best choice for difficult problems in our experiments.

These results suggest that combining the L&P and RS algorithms is indeed effective, and that alternating between them ≈ 5 times yields the best result; after 5 iterations, there is hardly any improvement. Moreover, applying L&P as the first GMI cut strengthening step seems a better choice than starting with RS: this is because RS cuts are not as consistently strong as L&P cuts, being very strong on some problems, but weak on others.

Finally, we observe that the average cut generation time increases by $\approx 50\%$ from $\eta = 1$ to $\eta = 2$: the second step is computationally expensive, but not as expensive as the first one. This is because at each step, we can reuse some of the data computed at previous iterations; in particular, we do not have to recompute the LP basis inverse from scratch. For each $\eta > 2$, the CPU time required for cut generation increases by less than 10%, since the number of cuts that are modified decreases. On Easy instances, GMI cuts are the best choice in terms of CPU time, because all other methods spend too much time for cut generation at the root – more than the time needed to solve the instance with GMI cuts only. On Medium instances, all methods improve the root gap closed, and reduce the total number of nodes without deteriorating the CPU time. On Hard instances cut generation is expensive, but it is rewarded by the final gap closed. To conclude, our cuts can be effective in reducing total solution time, provided that we have a mechanism to detect easy instances for which excessive cut generation time is detrimental. This issue is beyond the scope of this paper.

6.4 Comparison with default Lift-and-Project and Reduce-and-Split

We now compare the results obtained by our cut generator with the default versions of the L&P and RS generators described in [12, 19]. As mentioned in Section 6.2, our configuration uses very similar parameters to the default settings of the L&P generator, but the RS generator is different because in this paper we generate at most 50 cuts per round. In this section we provide results to back our choice of the parameters and experimental setup.

In Table 5 we report average values for the results obtained in the same Cut-and-Branch framework discussed in the previous sections. For the sake of brevity, we do not provide detailed results. We use the same format of Tables 2 and 3.

Table 5: Performance of the default version of the L&P and RS cut generators in a Cut-and-Branch framework.

	Default L&P						Default RS					
instances	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
Easy	68.38	2.16	230.50	100.00	58.29	2.55	52.27	8.71	1272.81	100.08	134.47	9.78
Medium	37.97	1.45	263.56	100.00	55589.05	37.56	27.44	1.70	1189.61	100.00	100385.71	60.88
Hard	31.89	20.69	369.84	64.75	565815.94	7200.03	26.60	110.31	4342.26	63.66	420426.50	7200.02

As expected, Default L&P has a very similar performance to the L&P-1 cut generator tested in this paper. Our combination of L&P and RS has a better average performance than Default L&P: this can be seen, for instance, by comparing Default L&P of Table 5 with L&P-4 in Table 2. Default RS does not perform similarly to RS-1: the reason is the huge difference in the number of generated cuts. If no upper limit is enforced, Default RS generates thousands of cuts per instance, whereas in this paper we generate at most 500 cuts. Despite the disparity in the number of cuts, Default RS seems to perform worse than RS-1 (except on Hard instances, where the performance is comparable). This can be explained by the fact that Default RS does not generate simple GMI cuts: if a GMI cut cannot be improved by the RS procedure, it discards the cut, whereas RS-1 generates the simple GMI cut. In [19], it was implicitly assumed that the RS cuts would be used in conjunction with GMI cuts, but this is not the case here.

In light of these results, our choice of parameters and experimental setup is justified: in [19], the default RS generator is tested in a different framework that would make comparisons impossible with our current setup, and in [12], the default L&P generator has very similar performance to our L&P-1.

6.5 RS cuts on non-optimal bases: cut violation

We provide here a brief analysis of the cut violation when RS cuts are generated from non-optimal bases of the LP relaxation. Recall from Section 4 that in this case, we could generate cuts which are not violated, which would have to be discarded. It is natural to ask how often does this happen; this is the question we try to answer in this section. Thus, we recorded the number of non violated cuts that are computed while applying 10 rounds of cuts at the root on our test set, with the L&P-4 generator (which turned out to be the strongest one, see Section 6.3). We gathered the same statistics for other generators as well, and obtained very similar results, therefore here we only present data for L&P-4.

As discussed in Section 4, if we generate large cut coefficients on the variables $j \in B^*$ which are basic in the optimal LP basis, but are nonbasic in the L&P basis from which RS cuts are generated, then the cut may not be violated. This explains why we modify the RS algorithm to try and reduce those cut coefficients as much as possible. How often do we generate non-violated cuts if we employ the RS algorithm unmodified? It turns out that, even if we do *not* consider the set B^* when applying the RS coefficient reduction algorithm (i.e. we apply the RS algorithm directly as described in [19]), only 11 cuts are discarded because they are not violated. This is an extremely small number: for comparison, the total number of generated cuts is 17085. We give two possible explanations for this behavior. First, the L&P cut given in input to the RS procedure cuts off the optimal basic solution $\bar{x}$ by a larger amount than the initial GMI cut; hence, changing the split disjunction to obtain a stronger cut is likely to still cut off $\bar{x}$. Second, sparsity plays in our favor: if the LP tableau on which RS is applied is sufficiently sparse, it is likely that computing a linear combination of its rows will not deteriorate the coefficients on the columns $j \in B^*$ by a large amount.

Thus, there is not a big margin of improvement for the modification of the RS algorithm proposed in Section 4: the number of non-violated cuts is already negligible. Indeed, it turns out that with the modified RS algorithm, we still generate 11 non-violated cuts (in total, we generate 17377 cuts in this case). However, an interesting side effect of the modification is that we close more integrality gap: the average integrality gap closed at the root over all instances after 10 rounds increases from 47.39% to 49.01%. Hence, the proposed modification seems to have a positive effect. Our intuition is that the modified RS algorithm is likely to increase the cut violation, yielding deeper cuts. This can be seen by looking at the expression for the distance cut off (first used as a measure of cut quality in [9]). Suppose the cut is $\alpha x \leq \alpha_0$; then the normal vector of the hyperplane represented by this cut is α. Therefore, the distance d of the basic solution $\bar{x}$ from the hyperplane $\alpha x = \alpha_0$ satisfies $\alpha(\bar{x} + d\alpha) = \alpha_0$. From this we get the expression:

$$d = (\alpha_0 - \alpha\bar{x})/\|\alpha\|_2^2. \tag{16}$$

By giving more priority to reducing cut coefficients on the columns $j \in B^*$ such that $\bar{x}_j > 0$, the modified RS algorithm acts on both the numerator and the denominator of (16), as opposed to only trying to reduce the denominator.

6.6 Cut density

We conclude our computational study with an analysis of the density of the cutting planes generated by the methods proposed in this paper. The density is recorded on all cutting planes generated during the 10 rounds applied at the root node of all instances in our test set, and for each cut it is computed as a percentage with respect to the maximum density allowed, i.e.: number of nonzeroes over the maximum number of nonzeroes allowed. The maximum number of nonzeroes allowed is equal to $\min\{n, 1000 + n/5\}$, where n is the number of columns; similar strategies to select the maximum density are used in the Branch-and-Cut solvers COIN-OR Cbc [15] and SCIP [1]. In Figure 4 we report the average density values for all cut generators L&P-η and RS-η with $\eta = 1, \ldots, 6$, for each round of cut generation applied at the root. For comparison, we additionally report the same curve for the traditional GMI cuts.

We can draw some conclusions from the graph. Surprisingly, GMI cuts are the densest cut on average, and they are also denser than most other cuts through the 10 rounds, with the exception of RS-1. RS-1 is close to GMI in most rounds; therefore, even if it aims at reducing cut coefficients (in the extended $(n+m)$-space, i.e. when the tableau is expressed with equality constraints), it does not reduce density (in the original n-space)

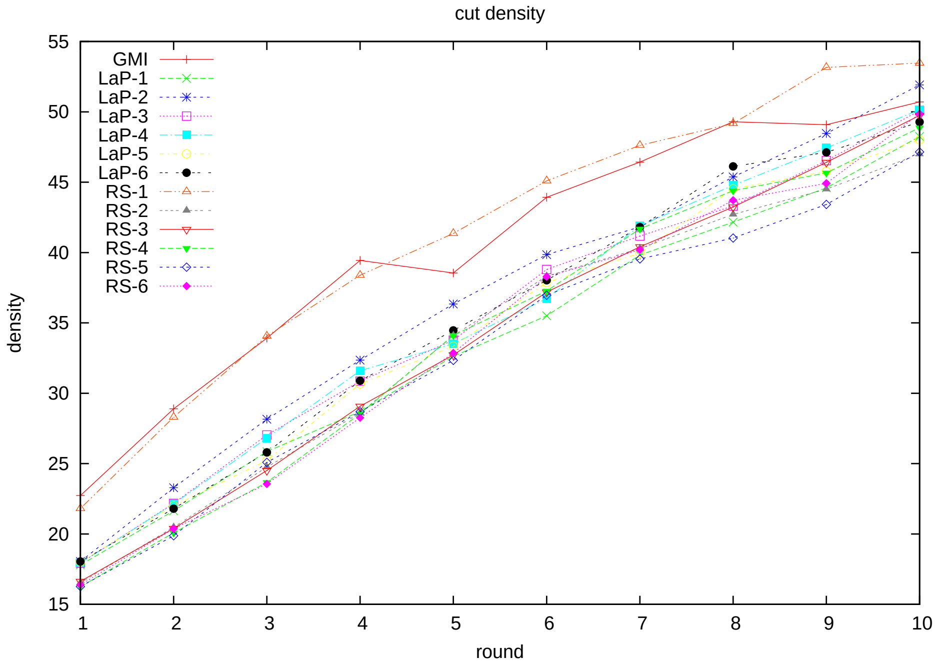

Figure 4: Average cut density for the first 10 rounds.

by a large amount: $\approx 2\%$ on average. L&P cuts, on the other hand, appear to be consistently sparser than GMI cuts through all 10 rounds. The same beneficial effect is observed when L&P and RS are combined. An important observation is that there does not seem to be an increase in cut density when η moves from 1 to 6: our combined L&P + RS cut generation algorithm is very stable in this respect, regardless of the number of iterations and whether we start with L&P or RS. Finally, density grows steadily with the number of applied rounds, and the distance between GMI and other cut generators becomes smaller: at the tenth round, all cut generators yield similarly dense cuts, and the density is more than double that of the first round.

7 Conclusion

In this paper we presented a combination of two existing algorithms for generating split cuts: Lift-and-Project and Reduce-and-Split. In doing so, we introduced an extension of the Lift-and-Project procedure on the original simplex tableau that can be employed on general split disjunctions (instead of elementary disjunctions), and we analyzed the application of Reduce-and-Split on non-optimal bases of the LP relaxation. We obtained a cut generation algorithm that iteratively modifies both the LP basis and the split disjunction from which a split cut is generated.

Computational experiments on a set of benchmark instances showed that this combination is effective on mixed-integer instances, solving problems in a smaller number of nodes and closing more integrality gap on the unsolved problems on average. In particular, iterating more than once between L&P and RS proved to be a good choice: in our experiments, applying L&P first and then iterating 4 times between the two algorithms yielded the best results. Our cut generation algorithm is not significantly slower than the original L&P and RS algorithms, but generates stronger cutting planes that should be useful in practice for the solution of difficult MILPs.

References

[1] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.

[3] 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.

[4] K. Andersen, G. Cornuéjols, and Y. Li. Split closure and intersection cuts. *Mathematical Programming A*, 102(3):457–493, 2005.

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

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

[7] E. Balas, and R. G. Jeroslow. Stengthening cuts for mixed integer programming *European Journal of Operational Research*, 4:224–234, 1980.

[8] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programming. *Mathematical Programming*, 58: 295–324, 1993.

[9] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

[10] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

[11] E. Balas and M. Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer gomory cuts for 0-1 programming. *Mathematical Programming*, 94(2-3):221–245, 2003.

[12] E. Balas and P. Bonami. Generating lift-and-project cuts from the lp simplex tableau: open source implementation and testing of new variants. *Mathematical Programming Computation*, 1:165–199, 2009.

[13] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[14] S. Ceria, G. Cornuéjols, and M. Dawande. Combining and strengthening Gomory cuts. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization*, volume 920 of *Lecture Notes in Computer Science*, pages 438–451. Springer Berlin / Heidelberg, 1995.

[15] COIN-OR Branch-and-Cut. `https://projects.coin-or.org/Cbc`

[16] COIN-OR Cut Generation Library. `https://projects.coin-or.org/Cgl`

[17] COIN-OR Linear Programming. `https://projects.coin-or.org/Clp`

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

[19] G. Cornuéjols and G. Nannicini. Reduce-and-split revisited: efficient generation of split cuts for mixed-integer linear programs. Technical report, Tepper School of Business, Carnegie Mellon University, April 2010.

[20] R. E. Gomory. An algorithm for the mixed-integer problem. Technical Report RM-2597, RAND Corporation, 1960.

[21] I. ILOG. *IBM ILOG CPLEX 12.1 User's Manual*. IBM ILOG, Gentilly, France, 2009.

[22] F. Margot. Testing cut generators for mixed-integer linear programming. *Mathematical Programming Computation*, 1(1):69–95, 2009.

[23] G. Nemhauser and L. Wolsey. A recursive procedure for generating all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.

[24] M. Perregaard. A practical implementation of lift-and-project cuts. In *International Symposium on Mathematical Programming*, Copenhagen, 2003.

[25] F. Wesselmann. Strengthening Gomory mixed-integer cuts: a computational study. Technical report, University of Paderborn, 2009.

A Detailed tables of results from Section 6.3

Table 6: Detailed results for L&P-1 and RS-1 cuts.

	L&P-1						RS-1					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	16.54	252	100.00	161	21.08	100.00	8.25	4	100.00	161	12.79
blend2	32.56	0.18	129	100.00	917	0.50	29.08	0.15	114	100.00	929	0.42
dcmulti	86.52	2.25	398	100.00	94	2.46	72.64	0.77	348	100.00	209	1.09
dsbmip	0.00	4.56	384	100.00	13	4.74	0.00	4.12	364	100.00	13	4.30
egout	100.00	0.02	55	100.00	0	0.02	100.00	0.01	32	100.00	0	0.01
fiber	91.77	1.68	305	100.00	508	2.06	88.52	4.76	206	100.00	570	5.47
fixnet6	56.72	1.54	189	100.00	693	3.07	49.94	0.60	161	100.00	376	1.06
flugpl	16.36	0.02	73	100.00	290	0.03	96.88	0.01	50	100.00	64	0.01
gen	96.07	0.25	95	100.00	2	0.26	91.65	0.15	92	100.00	2	0.17
gesa3	81.04	5.85	500	100.00	94	6.11	44.11	2.91	500	100.00	240	3.20
gesa3_o	89.57	5.28	403	100.00	73	5.49	61.69	2.71	145	100.00	312	3.06
khb05250	99.13	0.46	106	100.00	13	0.52	96.68	0.31	109	100.00	22	0.37
misc06	96.99	0.44	96	100.00	12	0.50	80.30	0.29	105	100.00	32	0.36
qnet1	51.10	4.76	461	100.00	585	10.78	33.05	6.14	283	100.00	495	8.36
qnet1_o	69.26	3.07	405	100.00	237	4.19	58.67	4.74	311	100.00	527	7.91
rentacar	0.00	2.17	34	100.00	17	2.52	0.00	0.93	10	100.00	30	1.37
aflow30a	47.13	4.10	409	100.00	25987	62.30	29.50	1.43	313	100.00	54983	62.03
arki001	50.59	7.78	408	100.00	490040	1202.43	57.06	6.81	490	100.00	276275	833.40
bell3a	62.16	0.03	26	100.00	27922	1.74	62.16	0.02	29	100.00	15696	1.56
gesa2	94.74	5.53	443	100.00	2662	8.04	91.62	3.68	496	100.00	2274	6.25
gesa2_o	94.28	4.20	461	100.00	3822	8.19	91.17	3.42	417	100.00	1148	4.93
glass4	0.00	0.26	314	100.00	463683	114.80	0.00	0.24	251	100.00	2265847	515.80
mas74	9.21	0.23	188	100.00	2981710	319.32	8.11	0.06	87	100.00	2885266	268.69
mas76	9.35	0.22	168	100.00	545667	48.21	8.16	0.04	62	100.00	516026	40.76
misc07	2.43	0.92	246	100.00	39561	24.46	0.72	0.39	126	100.00	18140	10.45
mod011	12.22	8.45	144	100.00	6406	57.72	0.26	6.57	1	100.00	17246	81.09
modglob	62.63	1.14	340	100.00	92705	45.23	56.32	0.57	308	100.00	171204	80.24
noswot	-0.00	0.08	163	100.00	694040	104.47	0.00	0.07	181	100.00	694040	103.57
pk1	0.00	0.07	150	100.00	279380	35.56	0.00	0.07	149	100.00	279380	36.01
pp08aCUTS	87.04	1.78	385	100.00	1283	3.20	73.49	0.63	418	100.00	4596	4.42
pp08a	95.03	0.75	329	100.00	2106	2.13	94.93	0.33	378	100.00	2883	2.43
qiu	27.94	18.71	467	100.00	29448	563.77	11.44	5.04	469	100.00	12591	121.07
rgn	51.36	0.08	144	100.00	3580	0.42	70.66	0.06	132	100.00	2262	0.32
rout	31.68	2.40	454	100.00	128541	203.35	10.33	0.67	269	100.00	181745	135.69
vpm1	55.54	0.07	134	100.00	2665	0.47	100.00	0.03	79	100.00	1	0.04
vpm2	63.10	0.41	288	100.00	16907	5.50	51.80	0.22	294	100.00	18999	6.67
a1c1s1	27.21	27.27	499	63.82	799673	7200.00	23.41	6.18	478	61.43	1419778	7200.01
aflow40b	34.15	14.67	218	95.50	1063047	7200.00	19.52	3.37	81	91.87	1773851	7200.00
b1c1s1	16.80	41.81	500	69.01	397488	7200.00	17.06	9.09	486	69.27	599906	7200.01
b2c1s1	16.09	59.47	500	66.58	173045	7200.01	10.24	13.28	427	64.63	281404	7200.00
bg512142	3.00	13.79	500	49.61	774056	7200.00	2.61	6.92	492	49.38	745056	7200.00
dano3mip	0.02	178.92	6	1.04	2101	7200.04	0.03	61.26	7	1.06	2458	7200.04
danoint	1.16	3.50	348	61.35	576748	7200.01	1.07	1.44	358	79.97	750774	7200.00
dg012142	0.01	23.77	500	49.53	561717	7200.00	0.01	31.92	498	55.71	616579	7200.01
mkc	54.21	5.29	198	81.31	2832166	7200.00	32.28	21.89	216	80.97	2407618	7200.00
momentum1	64.54	312.14	230	65.35	521	7200.06	64.52	144.15	227	81.97	527	7200.12
momentum2	40.77	1074.65	229	68.77	477	7200.17	39.05	310.60	219	68.78	488	7200.24
nsrand-ipx	61.80	35.88	217	87.60	1434311	7200.00	49.78	9.75	108	86.34	1713539	7200.00
opt1217	20.96	1.22	268	22.83	4739969	7200.08	54.19	1.20	222	54.19	8413096	7200.01
roll3000	53.08	16.62	393	82.06	952937	7200.00	7.46	6.53	289	61.47	772180	7200.00
set1ch	67.41	2.40	500	93.70	10653874	7200.00	48.02	1.17	500	79.31	3831719	7200.09
swath	28.40	17.98	290	58.08	896689	7200.00	28.11	10.54	129	58.71	1014968	7200.00
timtab1	39.72	0.74	500	87.38	16443060	7199.99	42.21	0.36	500	91.82	16792769	7200.00
timtab2	27.08	1.32	500	56.41	10903845	7200.01	26.92	1.24	500	57.18	10039227	7200.00
tr12-30	45.36	4.57	500	55.86	5776355	7200.01	47.76	3.12	500	57.31	3431033	7200.00

Table 7: Detailed results for L&P-2 and RS-2 cuts.

	L&P-2						RS-2					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	35.83	213	100.00	161	40.37	100.00	7.36	4	100.00	161	11.89
blend2	35.83	0.47	147	100.00	917	0.75	32.53	0.38	146	100.00	908	0.70
dcmulti	87.44	4.00	440	100.00	107	4.35	82.87	2.63	399	100.00	210	3.03
dsbmip	0.00	6.03	350	100.00	13	6.22	0.00	6.95	375	100.00	13	7.14
egout	100.00	0.02	47	100.00	0	0.02	100.00	0.01	32	100.00	0	0.01
fiber	89.82	4.44	322	100.00	570	5.15	92.35	5.73	257	100.00	515	6.12
fixnet6	57.92	1.67	167	100.00	605	3.17	52.95	1.68	197	100.00	1053	4.25
flugpl	75.47	0.01	67	100.00	133	0.02	98.86	0.01	71	100.00	28	0.02
gen	96.94	0.35	85	100.00	0	0.37	94.31	0.35	98	100.00	0	0.37
gesa3	70.22	9.98	500	100.00	183	10.41	75.19	8.93	500	100.00	76	9.26
gesa3_o	90.06	9.25	404	100.00	34	9.43	93.05	7.19	346	100.00	43	7.29
khb05250	98.66	0.58	99	100.00	16	0.64	97.07	0.61	105	100.00	18	0.67
misc06	99.73	0.56	95	100.00	7	0.61	94.28	0.57	112	100.00	22	0.62
qnet1	50.15	12.44	468	100.00	449	14.84	49.93	11.21	463	100.00	598	16.86
qnet1_o	70.65	6.77	401	100.00	301	7.81	73.77	5.90	396	100.00	300	7.27
rentacar	0.00	2.46	34	100.00	17	2.80	0.00	1.75	11	100.00	30	2.18
aflow30a	47.17	5.79	398	100.00	32231	60.44	43.15	4.58	401	100.00	31300	62.41
arki001	55.89	14.21	463	100.00	195454	634.03	56.51	8.89	487	100.00	230492	552.87
bell3a	62.16	0.03	30	100.00	28082	1.71	62.16	0.03	26	100.00	24756	1.54
gesa2	97.58	9.24	478	100.00	236	9.58	96.98	8.10	483	100.00	157	8.36
gesa2_o	97.41	7.86	454	100.00	680	8.56	98.41	6.63	446	100.00	115	6.77
glass4	0.00	0.27	314	100.00	463683	113.37	0.00	0.32	305	100.00	2265847	520.24
mas74	9.02	0.29	178	100.00	3012675	315.93	8.69	0.20	132	100.00	3155390	354.51
mas76	9.08	0.23	146	100.00	494158	40.92	9.36	0.12	122	100.00	465009	40.84
misc07	1.61	1.22	278	100.00	16243	10.58	4.05	1.05	282	100.00	46806	28.24
mod011	5.19	13.09	131	100.00	13603	81.42	0.28	9.35	3	100.00	19128	97.00
modglob	65.22	1.38	312	100.00	17820	9.67	63.27	1.75	368	100.00	47297	29.91
noswot	-0.00	0.09	152	100.00	694040	104.12	0.00	0.11	192	100.00	694040	104.29
pk1	0.00	0.07	150	100.00	279380	35.65	0.00	0.07	149	100.00	279380	36.01
pp08aCUTS	90.09	1.94	344	100.00	1938	4.38	89.85	1.66	361	100.00	1646	3.84
pp08a	95.76	1.20	353	100.00	1679	2.74	96.51	0.77	327	100.00	1440	1.96
qiu	28.13	18.77	467	100.00	48408	796.39	27.71	22.07	469	100.00	58757	1079.17
rgn	57.61	0.09	138	100.00	4114	0.57	66.36	0.08	128	100.00	3589	0.44
rout	33.21	4.07	458	100.00	214771	356.08	31.60	2.62	442	100.00	123222	235.92
vpm1	78.19	0.05	88	100.00	112	0.08	92.99	0.04	79	100.00	39	0.06
vpm2	68.74	0.65	302	100.00	5125	2.67	58.98	0.53	310	100.00	5640	2.50
a1c1s1	23.71	42.54	497	60.44	814055	7200.00	26.28	29.11	457	62.35	954268	7200.01
aflow40b	34.61	18.72	255	94.92	1443596	7200.00	22.03	4.45	87	98.21	2025478	7200.00
b1c1s1	18.05	67.57	500	68.56	381824	7200.02	15.01	34.61	444	66.81	421127	7200.02
b2c1s1	16.98	99.04	500	67.26	156537	7200.01	14.61	52.50	404	63.13	185771	7200.01
bg512142	2.97	28.45	500	48.44	544550	7200.01	2.27	16.40	496	49.90	608810	7199.99
dano3mip	0.02	418.07	6	1.02	1881	7200.01	0.03	93.50	7	1.06	2373	7200.13
danoint	1.16	5.73	354	81.80	685112	7200.00	1.33	3.76	352	66.54	583194	7200.01
dg012142	0.01	73.97	500	52.31	555966	7200.01	0.01	46.56	494	52.74	530445	7200.01
mkc	41.92	15.53	222	80.41	2590007	7200.00	63.57	14.97	235	81.21	2490173	7200.00
momentum1	64.49	396.92	266	64.75	489	7200.15	62.06	344.01	220	64.67	514	7200.19
momentum2	40.92	1043.20	195	68.80	506	7200.28	39.57	1042.54	230	68.84	512	7200.15
nsrand-ipx	62.97	47.95	270	92.60	1772810	7200.00	50.21	11.47	105	84.04	1731898	7200.00
opt1217	17.60	1.38	267	24.41	11270792	7200.01	13.78	0.83	224	21.77	10414613	7200.00
roll3000	28.89	28.08	413	70.92	665477	7200.01	39.53	19.37	403	83.16	1014449	7200.00
set1ch	63.64	4.73	499	87.61	6815028	7200.00	56.50	3.76	500	87.57	6527500	7200.01
swath	28.43	21.18	273	58.93	1062400	7200.01	27.39	18.73	192	49.51	981648	7200.01
timtab1	48.84	1.82	497	92.65	16375094	7200.00	45.70	0.97	500	90.96	16663797	7200.00
timtab2	31.81	3.96	494	58.96	7914985	7200.00	27.71	1.99	500	58.30	9757182	7200.00
tr12-30	46.16	10.72	495	55.64	5626807	7200.00	38.78	7.59	500	49.43	6222719	7200.00

Table 8: Detailed results for L&P-3 and RS-3 cuts.

	L&P-3						RS-3					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	36.81	227	100.00	161	41.37	100.00	9.93	4	100.00	161	14.47
blend2	34.75	0.42	134	100.00	967	0.76	33.21	0.54	146	100.00	935	0.82
dcmulti	89.14	3.65	412	100.00	78	3.91	83.35	4.34	444	100.00	158	4.68
dsbmip	0.00	6.59	354	100.00	13	6.78	0.00	10.41	345	100.00	13	10.60
egout	100.00	0.01	44	100.00	0	0.02	100.00	0.01	20	100.00	0	0.01
fiber	89.89	2.51	212	100.00	540	2.89	90.64	10.34	286	100.00	495	10.86
fixnet6	59.11	2.16	182	100.00	628	3.26	56.05	1.93	166	100.00	1061	3.45
flugpl	97.54	0.01	57	100.00	18	0.01	99.80	0.01	66	100.00	3	0.02
gen	97.38	0.24	64	100.00	2	0.26	58.35	0.50	107	100.00	2	0.52
gesa3	78.52	10.01	500	100.00	110	10.32	81.32	13.53	500	100.00	51	13.78
gesa3_o	94.93	9.78	402	100.00	28	9.97	89.38	11.88	377	100.00	47	12.04
khb05250	98.79	0.55	90	100.00	15	0.60	98.10	0.97	114	100.00	20	1.05
misc06	99.48	0.50	89	100.00	10	0.55	98.40	0.85	125	100.00	12	0.90
qnet1	47.68	12.50	461	100.00	640	17.62	58.53	18.52	349	100.00	286	19.84
qnet1_o	71.34	8.19	414	100.00	429	9.71	74.90	11.64	415	100.00	237	12.76
rentacar	0.00	2.51	34	100.00	17	2.85	0.00	2.13	11	100.00	30	2.57
aflow30a	44.86	6.83	393	100.00	32341	73.04	43.78	6.74	385	100.00	36989	75.98
arki001	53.49	14.12	442	100.00	228453	880.62	63.45	12.66	472	100.00	168103	571.50
bell3a	62.16	0.03	29	100.00	33524	2.11	62.16	0.03	26	100.00	24756	1.54
gesa2	97.37	9.89	443	100.00	260	10.20	96.65	13.05	492	100.00	886	14.05
gesa2_o	97.66	9.25	447	100.00	578	10.03	96.28	12.18	471	100.00	439	12.66
glass4	0.00	0.27	314	100.00	463683	114.52	0.00	0.34	305	100.00	2265847	523.23
mas74	8.92	0.34	181	100.00	3013313	336.47	8.74	0.25	147	100.00	4350042	488.91
mas76	9.31	0.22	168	100.00	540241	44.29	9.80	0.16	120	100.00	496915	42.31
misc07	2.51	1.15	259	100.00	23080	14.97	2.62	1.34	262	100.00	30760	21.35
mod011	5.31	17.26	115	100.00	10769	89.92	0.28	9.45	3	100.00	19128	87.36
modglob	63.57	1.92	322	100.00	34601	17.67	68.94	1.71	330	100.00	40151	24.69
noswot	-0.00	0.10	175	100.00	694040	104.50	-0.00	0.13	179	100.00	694040	105.24
pk1	0.00	0.07	150	100.00	279380	35.78	0.00	0.07	149	100.00	279380	36.03
pp08aCUTS	85.40	2.66	358	100.00	1669	4.25	90.87	1.54	322	100.00	2090	3.82
pp08a	97.17	1.05	319	100.00	992	2.28	96.34	0.75	259	100.00	1180	1.71
qiu	28.62	23.10	467	100.00	37797	663.80	30.82	23.66	466	100.00	36000	662.47
rgn	52.66	0.17	165	100.00	4059	0.65	68.50	0.09	124	100.00	2956	0.46
rout	44.05	4.56	434	100.00	123978	243.45	40.58	4.09	434	100.00	248505	597.18
vpm1	92.95	0.05	71	100.00	54	0.06	92.99	0.06	79	100.00	39	0.07
vpm2	61.64	0.61	287	100.00	3179	1.67	66.79	0.58	306	100.00	3927	1.71
a1c1s1	28.00	53.86	499	66.52	903983	7200.01	24.03	38.14	430	61.11	875210	7200.00
aflow40b	35.13	22.23	211	100.00	1471422	6327.53	26.72	5.75	109	92.11	1554640	7200.00
b1c1s1	21.15	63.84	500	71.44	364426	7200.02	15.39	56.61	409	68.64	451821	7200.00
b2c1s1	17.31	119.59	500	66.67	178552	7200.00	17.99	77.88	459	62.61	166794	7200.01
bg512142	3.04	31.13	500	49.02	579253	7200.01	2.47	32.69	489	50.46	627360	7200.01
dano3mip	0.02	404.32	6	1.04	2115	7200.08	0.03	130.95	7	1.02	1905	7200.09
danoint	1.16	6.33	357	63.96	576663	7200.01	1.08	5.61	354	59.28	518942	7200.01
dg012142	0.01	79.07	500	41.48	592802	7200.00	0.01	104.81	489	45.92	579770	7200.00
mkc	50.56	22.73	257	81.00	2466335	7200.00	52.87	36.36	235	79.99	2669203	7200.00
momentum1	64.46	373.26	259	82.23	681	7200.11	64.52	560.36	234	82.36	1473	7200.09
momentum2	32.36	1135.28	235	68.82	499	7200.18	35.97	1217.84	165	68.81	487	7200.34
nsrand-ipx	65.75	53.89	258	94.08	1901215	7200.01	56.86	14.60	119	84.39	1667384	7200.00
opt1217	21.65	2.32	304	24.80	10641431	7200.00	17.86	2.37	303	21.73	8138058	7200.00
roll3000	35.99	26.13	414	83.28	1196977	7200.00	47.06	31.44	421	77.48	656312	7200.01
set1ch	56.82	5.58	496	85.01	6117535	7200.05	60.11	5.65	497	89.03	6954290	7200.01
swath	28.43	24.25	290	58.00	931899	7200.01	27.97	19.16	148	47.16	994738	7200.00
timtab1	43.36	2.24	492	92.07	15276320	7200.00	51.21	1.90	498	93.70	15326395	7200.01
timtab2	31.25	4.16	494	60.75	8343697	7200.00	33.56	5.15	500	61.64	8928425	7200.02
tr12-30	43.60	12.12	494	52.83	4437638	7200.00	42.36	13.57	498	52.57	4942071	7200.01

Table 9: Detailed results for L&P-4 and RS-4 cuts.

	L&P-4						RS-4					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	36.33	227	100.00	161	40.85	100.00	7.49	4	100.00	161	12.00
blend2	38.55	0.52	141	100.00	1055	0.85	35.43	0.54	154	100.00	1020	0.87
dcmulti	89.92	3.98	422	100.00	90	4.22	85.17	4.02	405	100.00	123	4.36
dsbmip	0.00	6.81	367	100.00	13	7.00	0.00	9.19	351	100.00	13	9.38
egout	100.00	0.01	44	100.00	0	0.02	100.00	0.01	20	100.00	0	0.01
fiber	88.45	2.49	185	100.00	524	2.96	92.26	8.96	249	100.00	492	9.39
fixnet6	59.28	1.75	160	100.00	610	2.86	56.68	2.11	179	100.00	702	4.03
flugpl	97.55	0.01	57	100.00	16	0.01	99.93	0.02	70	100.00	4	0.02
gen	97.50	0.24	64	100.00	2	0.25	60.76	0.49	107	100.00	0	0.51
gesa3	75.12	11.05	500	100.00	71	11.30	79.46	13.18	500	100.00	94	13.52
gesa3_o	90.22	8.37	374	100.00	32	8.50	94.49	10.56	349	100.00	29	10.67
khb05250	98.79	0.55	90	100.00	14	0.59	98.92	0.82	104	100.00	15	0.89
misc06	98.90	0.55	88	100.00	9	0.60	98.48	0.82	123	100.00	13	0.87
qnet1	51.20	13.71	453	100.00	371	15.23	50.20	20.52	412	100.00	368	22.91
qnet1_o	75.19	10.33	398	100.00	235	11.60	73.57	10.62	416	100.00	525	14.42
rentacar	0.00	2.48	34	100.00	17	2.83	0.00	1.95	11	100.00	30	2.38
aflow30a	48.87	6.66	382	100.00	26691	64.72	45.55	7.05	405	100.00	29912	72.94
arki001	63.33	12.23	424	100.00	105960	337.49	61.47	12.51	475	100.00	258489	670.37
bell3a	62.16	0.03	29	100.00	33524	2.11	62.16	0.03	26	100.00	24756	1.55
gesa2	97.02	10.44	445	100.00	297	10.80	97.34	12.66	490	100.00	263	13.02
gesa2_o	97.73	9.62	454	100.00	345	9.94	96.65	11.39	444	100.00	412	11.80
glass4	0.00	0.28	314	100.00	463683	115.96	0.00	0.35	305	100.00	2265847	518.50
mas74	8.84	0.35	176	100.00	3916510	448.68	8.81	0.26	148	100.00	3159428	341.59
mas76	8.99	0.28	143	100.00	502255	43.56	9.47	0.13	101	100.00	475130	40.81
misc07	2.51	1.19	245	100.00	43040	24.12	1.97	1.38	285	100.00	24364	15.53
mod011	5.31	19.17	114	100.00	10823	93.39	0.28	9.43	3	100.00	19128	87.64
modglob	67.85	2.14	327	100.00	53342	30.80	74.86	1.63	314	100.00	22947	12.78
noswot	-0.00	0.09	179	100.00	694040	104.54	0.00	0.12	168	100.00	694040	105.29
pk1	0.00	0.07	150	100.00	279380	35.92	0.00	0.07	149	100.00	279380	36.17
pp08aCUTS	90.89	2.54	343	100.00	1186	3.82	90.58	1.72	317	100.00	1361	3.28
pp08a	96.91	1.23	332	100.00	1074	2.46	96.33	1.16	326	100.00	1111	2.18
qiu	27.14	21.90	467	100.00	41264	592.33	28.57	23.33	468	100.00	39959	765.87
rgn	62.12	0.16	142	100.00	4044	0.65	74.72	0.10	123	100.00	3325	0.60
rout	39.61	4.32	434	100.00	115575	214.39	32.40	4.74	447	100.00	349576	651.47
vpm1	92.99	0.05	65	100.00	6	0.06	92.99	0.05	79	100.00	39	0.06
vpm2	67.45	0.61	282	100.00	2804	1.59	68.45	0.58	283	100.00	1503	1.28
a1c1s1	26.68	52.95	500	65.31	1301327	7200.00	25.68	48.29	419	66.25	1070359	7200.04
aflow40b	36.49	20.12	224	100.00	1225113	5238.47	27.88	6.92	118	92.85	1468403	7200.01
b1c1s1	17.62	96.32	500	69.71	271930	7200.01	17.10	59.59	340	69.86	435420	7200.00
b2c1s1	18.65	103.46	500	66.72	144884	7200.00	18.51	83.38	451	65.91	189048	7200.00
bg512142	2.55	38.92	500	48.92	558244	7200.00	2.55	34.53	489	46.30	624781	7200.01
dano3mip	0.02	439.15	6	1.04	2107	7200.08	0.03	135.57	7	1.09	2681	7200.14
danoint	1.46	6.34	336	68.06	540115	7200.00	1.36	6.37	355	67.94	582558	7200.01
dg012142	0.01	84.45	499	54.55	524882	7200.01	0.01	96.07	487	51.21	557727	7200.00
mkc	51.04	20.40	256	78.68	2781088	7200.00	53.00	56.82	237	73.72	2268773	7200.00
momentum1	63.31	373.60	266	70.68	536	7200.12	61.68	479.82	243	69.03	525	7200.16
momentum2	40.96	914.25	238	68.85	483	7200.11	38.19	1193.88	196	68.78	474	7200.42
nsrand-ipx	60.46	56.03	227	89.68	1749585	7200.00	57.78	14.89	127	84.38	1570748	7200.00
opt1217	29.73	2.27	293	30.93	8048994	7200.09	20.13	2.08	274	24.34	7342698	7200.00
roll3000	32.30	32.38	415	77.18	879156	7200.01	46.63	34.70	413	77.58	734294	7200.01
set1ch	61.32	6.32	493	89.30	7397715	7200.00	56.60	5.84	500	85.02	7234043	7200.00
swath	28.44	22.31	266	57.96	994262	7200.00	27.57	19.54	163	48.14	959010	7200.00
timtab1	46.05	2.16	494	91.43	16065016	7200.00	46.25	1.92	493	94.06	16386369	7200.00
timtab2	32.10	4.94	495	61.76	9149961	7200.00	29.94	5.27	499	58.26	9108168	7200.00
tr12-30	45.85	13.88	494	55.36	6131751	7200.00	44.89	15.68	499	55.03	3613179	7200.00

Table 10: Detailed results for L&P-5 and RS-5 cuts.

	L&P-5						RS-5					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	35.35	227	100.00	161	39.85	100.00	8.46	4	100.00	161	13.00
blend2	35.58	0.48	133	100.00	832	0.74	27.50	0.40	130	100.00	822	0.66
dcmulti	89.82	4.12	400	100.00	157	4.51	87.97	3.81	404	100.00	123	4.15
dsbmip	0.00	6.78	343	100.00	13	6.97	0.00	9.71	369	100.00	13	9.90
egout	100.00	0.01	44	100.00	0	0.02	100.00	0.01	20	100.00	0	0.01
fiber	90.32	5.54	329	100.00	547	6.10	90.43	9.66	274	100.00	509	10.22
fixnet6	61.46	1.98	171	100.00	597	3.40	54.90	2.57	191	100.00	842	4.61
flugpl	97.55	0.01	57	100.00	16	0.02	99.93	0.02	70	100.00	4	0.02
gen	98.15	0.26	67	100.00	0	0.27	89.88	0.39	83	100.00	0	0.41
gesa3	67.84	9.70	500	100.00	138	10.04	75.64	13.63	500	100.00	65	13.99
gesa3_o	90.76	9.10	390	100.00	44	9.25	94.49	11.25	349	100.00	29	11.37
khb05250	98.79	0.57	90	100.00	15	0.62	98.92	0.79	104	100.00	15	0.86
misc06	99.73	0.56	89	100.00	7	0.62	98.51	0.84	123	100.00	9	0.89
qnet1	56.57	19.02	464	100.00	504	22.18	47.22	20.17	463	100.00	608	26.60
qnet1_o	72.82	10.69	414	100.00	542	15.68	74.10	14.15	421	100.00	337	15.64
rentacar	0.00	2.64	34	100.00	30	3.07	0.00	2.08	11	100.00	30	2.52
aflow30a	48.79	6.72	409	100.00	21922	49.22	42.46	6.86	396	100.00	28898	59.40
arki001	61.78	16.07	482	100.00	673970	1784.63	64.31	11.53	477	100.00	177593	516.17
bell3a	62.16	0.03	29	100.00	33524	2.09	62.16	0.02	26	100.00	24756	1.55
gesa2	97.15	10.66	475	100.00	361	11.07	97.66	14.39	489	100.00	421	14.80
gesa2_o	97.59	9.72	446	100.00	288	10.00	97.76	12.74	455	100.00	327	13.09
glass4	0.00	0.29	314	100.00	2265847	532.52	0.00	0.34	305	100.00	2265847	517.79
mas74	9.09	0.37	182	100.00	3742849	435.87	8.71	0.23	128	100.00	2904274	304.35
mas76	9.36	0.27	143	100.00	296498	26.33	8.89	0.12	91	100.00	462411	39.71
misc07	2.51	1.08	246	100.00	38482	21.59	4.66	1.36	264	100.00	41917	25.75
mod011	5.21	17.70	117	100.00	10719	81.54	0.28	8.84	3	100.00	19128	84.76
modglob	66.68	2.08	325	100.00	50149	25.55	71.19	1.56	297	100.00	16719	10.28
noswot	-0.00	0.10	179	100.00	694040	104.97	0.00	0.15	194	100.00	694040	104.48
pk1	0.00	0.08	150	100.00	279380	35.95	0.00	0.07	149	100.00	279380	36.09
pp08aCUTS	88.52	2.54	339	100.00	2212	4.59	90.18	2.42	364	100.00	2310	5.21
pp08a	96.36	1.26	329	100.00	1406	2.43	96.63	1.29	346	100.00	1048	2.22
qiu	27.89	23.71	467	100.00	24090	499.77	28.58	23.90	468	100.00	36651	661.02
rgn	64.17	0.12	136	100.00	1299	0.25	73.95	0.10	122	100.00	3013	0.46
rout	31.83	5.13	453	100.00	182866	308.11	34.97	4.56	435	100.00	124676	257.86
vpm1	85.63	0.05	66	100.00	60	0.07	92.99	0.05	79	100.00	39	0.06
vpm2	64.99	0.77	297	100.00	4726	2.41	65.84	0.64	285	100.00	5073	2.42
a1c1s1	30.87	61.36	500	68.13	1045248	7200.00	25.76	47.33	450	62.96	1090123	7200.00
aflow40b	35.31	19.76	217	98.35	1543564	7200.00	27.88	6.87	118	93.14	1534693	7200.00
b1c1s1	18.77	80.81	500	70.20	387707	7200.00	14.71	67.37	404	68.85	426192	7200.01
b2c1s1	16.79	125.31	499	66.42	132228	7200.01	18.80	87.63	447	64.69	160327	7200.01
bg512142	3.26	44.49	500	49.50	564810	7200.01	2.71	36.36	489	48.39	570106	7200.00
dano3mip	0.02	386.52	6	1.05	2133	7200.10	0.03	156.65	7	1.05	2266	7200.11
danoint	1.45	7.61	350	70.03	582800	7200.00	1.42	6.37	354	76.44	634243	7200.00
dg012142	0.01	85.08	500	52.44	605850	7200.00	0.01	103.72	490	42.91	587047	7200.01
mkc	45.74	37.54	256	81.17	2462049	7200.00	47.22	32.79	264	78.68	2502733	7200.00
momentum1	52.81	390.39	237	64.72	513	7200.15	61.68	540.23	243	67.49	534	7200.16
momentum2	41.57	1208.50	104	68.81	517	7200.22	32.37	1204.54	143	68.80	478	7200.38
nsrand-ipx	65.38	58.13	241	93.91	1752996	7200.00	56.77	13.83	139	82.88	1697580	7200.00
opt1217	20.90	2.11	292	24.54	13017661	7200.00	17.70	1.73	266	22.87	16381688	7199.99
roll3000	37.85	35.77	435	76.57	675800	7200.01	44.86	35.66	418	84.01	992031	7200.00
set1ch	63.97	6.60	495	91.12	7147307	7200.03	56.92	7.06	497	84.88	6892564	7200.00
swath	28.44	29.17	273	57.70	963299	7200.00	27.40	19.88	154	50.90	1020300	7200.01
timtab1	48.98	2.48	493	98.43	16991540	7200.00	48.66	2.08	498	93.76	16073424	7200.00
timtab2	32.96	5.51	496	62.60	9715792	7200.00	31.98	5.32	497	61.11	9078882	7200.01
tr12-30	46.16	14.91	492	55.54	4977278	7200.00	41.73	17.50	495	52.27	4105357	7199.99

Table 11: Detailed results for L&P-6 and RS-6 cuts.

	L&P-6						RS-6					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	36.23	227	100.00	161	40.78	100.00	7.34	4	100.00	161	11.86
blend2	34.71	0.56	142	100.00	928	0.87	30.57	0.39	121	100.00	848	0.65
dcmulti	90.70	4.04	409	100.00	76	4.31	87.60	3.96	368	100.00	150	4.26
dsbmip	0.00	7.21	338	100.00	13	7.40	0.00	9.77	384	100.00	13	9.96
egout	100.00	0.02	44	100.00	0	0.02	100.00	0.01	20	100.00	0	0.01
fiber	91.91	5.75	323	100.00	564	6.29	90.33	7.68	297	100.00	500	8.23
fixnet6	62.21	1.98	174	100.00	451	2.52	56.95	2.38	184	100.00	666	4.42
flugpl	97.55	0.01	57	100.00	16	0.02	99.93	0.01	70	100.00	4	0.02
gen	98.15	0.26	67	100.00	0	0.27	95.69	0.38	85	100.00	0	0.39
gesa3	74.28	9.76	485	100.00	117	10.03	79.25	12.68	499	100.00	103	13.00
gesa3_o	90.83	9.40	390	100.00	43	9.55	94.82	10.24	353	100.00	26	10.32
khb05250	98.79	0.58	90	100.00	15	0.63	98.92	0.78	104	100.00	15	0.84
misc06	99.73	0.58	89	100.00	7	0.64	98.48	0.81	121	100.00	10	0.87
qnet1	54.04	18.51	464	100.00	411	20.90	45.98	18.70	467	100.00	339	20.90
qnet1_o	72.25	11.57	415	100.00	360	13.23	74.46	13.68	418	100.00	492	18.68
rentacar	0.00	2.62	34	100.00	30	3.05	0.00	2.01	11	100.00	30	2.46
aflow30a	45.41	7.52	401	100.00	25406	64.44	44.87	7.14	392	100.00	24630	52.77
arki001	67.86	13.63	464	100.00	538882	1634.39	63.25	13.68	481	100.00	184224	482.32
bell3a	62.16	0.03	29	100.00	33524	2.12	62.16	0.02	26	100.00	24756	1.54
gesa2	97.30	10.42	449	100.00	264	10.77	96.92	13.26	476	100.00	692	13.98
gesa2_o	91.78	10.09	451	100.00	716	11.63	97.47	11.03	452	100.00	636	11.87
glass4	0.00	0.28	314	100.00	2265847	529.79	0.00	0.34	305	100.00	2265847	521.99
mas74	9.09	0.38	182	100.00	3742849	446.98	8.81	0.20	121	100.00	3763241	406.77
mas76	9.32	0.28	162	100.00	297407	25.27	8.89	0.12	91	100.00	462411	39.81
misc07	2.51	1.13	246	100.00	38482	22.14	2.15	1.47	284	100.00	55238	37.27
mod011	5.21	16.53	117	100.00	10719	80.07	0.28	9.76	3	100.00	19128	89.14
modglob	59.46	2.26	342	100.00	52224	25.63	73.33	1.65	304	100.00	11299	6.41
noswot	-0.00	0.10	179	100.00	694040	104.97	0.00	0.15	194	100.00	694040	104.87
pk1	0.00	0.07	150	100.00	279380	35.69	0.00	0.07	149	100.00	279380	36.06
pp08aCUTS	88.15	2.87	354	100.00	2187	5.54	91.29	2.27	361	100.00	1482	4.37
pp08a	96.28	1.36	339	100.00	1284	2.69	96.31	1.22	337	100.00	1705	2.94
qiu	27.79	20.48	467	100.00	25465	404.77	28.67	25.66	468	100.00	30836	648.53
rgn	56.68	0.15	145	100.00	2893	0.54	72.84	0.10	119	100.00	2900	0.43
rout	29.73	5.83	463	100.00	187844	372.87	33.06	4.63	445	100.00	129373	241.08
vpm1	85.63	0.05	66	100.00	60	0.07	92.99	0.05	79	100.00	39	0.06
vpm2	66.48	0.70	283	100.00	4001	2.08	64.36	0.64	285	100.00	3302	1.77
a1c1s1	26.09	63.17	499	66.63	1154669	7200.00	21.47	48.15	408	60.90	1144082	7200.01
aflow40b	35.31	19.48	217	98.68	1609454	7200.00	27.88	7.27	118	93.29	1573342	7200.00
b1c1s1	18.41	95.71	500	71.09	356632	7200.01	16.11	66.66	386	69.15	383922	7200.00
b2c1s1	16.77	113.15	499	65.55	126223	7200.02	12.60	74.06	377	59.55	132862	7200.02
bg512142	3.33	43.77	500	48.35	569857	7200.01	2.68	38.90	491	49.42	635674	7200.00
dano3mip	0.02	359.22	6	1.05	2126	7200.10	0.03	129.35	7	1.07	2448	7200.07
danoint	1.31	7.57	353	67.71	562057	7200.01	1.52	6.68	349	63.13	518736	7200.01
dg012142	0.01	93.88	500	47.23	464062	7200.00	0.01	110.38	496	49.68	574702	7200.00
mkc	52.95	19.28	248	81.11	2422768	7200.01	54.09	29.23	243	79.44	2689472	7200.00
momentum1	64.51	338.10	230	70.98	513	7200.30	61.68	466.82	243	69.05	539	7200.21
momentum2	29.43	1251.96	198	68.87	519	7200.17	38.86	1224.55	201	68.81	501	7200.25
nsrand-ipx	62.71	58.81	250	88.98	1398748	7200.00	57.26	14.70	117	86.60	1510243	7200.01
opt1217	25.56	3.54	348	28.26	4670112	7200.01	23.24	2.12	274	26.22	7265291	7200.00
roll3000	56.36	34.27	408	72.82	678609	7200.01	44.00	38.01	429	76.80	661057	7200.02
set1ch	62.63	7.17	495	91.45	6952143	7200.07	59.18	6.76	494	85.17	7119449	7200.00
swath	28.43	28.81	287	58.34	1047207	7200.01	27.90	17.89	132	49.60	1025380	7200.00
timtab1	48.74	2.28	497	96.27	16722968	7200.00	45.19	2.23	495	94.21	16694199	7200.00
timtab2	30.36	5.57	497	61.32	9836809	7200.00	34.21	5.99	498	59.25	7902097	7200.00
tr12-30	46.12	16.42	493	55.74	6162441	7200.00	41.97	17.53	498	52.62	4431694	7200.00

Table 12: Detailed results for GMI cuts.

	GMI					
instance	root gap %	cut time	#cuts	final gap %	#nodes	total time
10teams	100.00	2.51	256	100.00	112	5.59
blend2	31.83	0.04	104	100.00	852	0.27
dcmulti	70.02	0.18	342	100.00	253	0.42
dsbmip	0.00	1.35	342	100.00	13	1.54
egout	98.97	0.01	100	100.00	4	0.01
fiber	89.78	0.20	239	100.00	632	0.71
fixnet6	48.38	0.26	191	100.00	690	1.12
flugpl	15.71	0.00	74	100.00	223	0.01
gen	21.91	0.05	106	100.00	0	0.08
gesa3	43.45	0.20	500	100.00	264	0.52
gesa3_o	60.88	0.13	162	100.00	317	0.43
khb05250	94.84	0.10	107	100.00	33	0.18
misc06	72.66	0.10	92	100.00	42	0.18
qnet1	33.24	0.52	264	100.00	664	2.56
qnet1_o	56.44	0.49	270	100.00	520	3.15
rentacar	0.00	0.20	16	100.00	30	0.62
aflow30a	32.50	0.59	332	100.00	65531	91.48
arki001	55.68	3.88	466	100.00	190622	505.08
bell3a	62.13	0.01	40	100.00	15844	1.57
gesa2	72.79	0.22	458	100.00	3371	4.63
gesa2_o	81.75	0.24	406	100.00	5497	5.60
glass4	0.00	0.04	258	100.00	2265847	518.47
mas74	8.12	0.02	56	100.00	2795555	289.31
mas76	7.04	0.02	47	100.00	397116	31.54
misc07	0.72	0.14	123	100.00	7423	4.66
mod011	3.49	0.54	115	100.00	18757	88.43
modglob	62.30	0.20	353	100.00	174053	102.61
noswot	0.00	0.02	149	100.00	694040	104.30
pk1	0.00	0.05	150	100.00	279380	36.12
pp08aCUTS	66.01	0.21	432	100.00	23005	14.61
pp08a	85.81	0.14	423	100.00	13323	7.89
qiu	15.55	2.63	468	100.00	17795	187.19
rgn	42.57	0.02	145	100.00	2632	0.36
rout	12.53	0.29	333	100.00	89627	87.84
vpm1	71.96	0.02	105	100.00	8163	1.18
vpm2	49.27	0.04	271	100.00	7642	1.88
a1c1s1	22.40	1.01	500	60.32	1463987	7200.00
aflow40b	17.19	0.62	99	92.09	1728792	7200.00
b1c1s1	12.52	1.96	500	70.35	655691	7200.01
b2c1s1	8.27	3.57	500	61.88	295712	7200.01
bg512142	1.60	1.90	500	49.50	904100	7200.01
dano3mip	0.02	7.19	5	1.04	2381	7200.06
danoint	1.01	0.40	348	82.06	860402	7200.00
dg012142	0.01	2.87	500	54.55	618284	7200.00
mkc	36.46	1.03	181	81.17	2580451	7200.00
momentum1	64.52	13.34	246	65.89	511	7200.30
momentum2	39.14	57.54	161	69.01	551	7200.08
nsrand-ipx	45.22	2.63	117	81.77	1551147	7200.00
opt1217	50.27	0.45	218	50.27	11579593	7200.01
roll3000	4.62	1.72	278	64.39	757635	7200.00
set1ch	61.11	0.20	500	89.79	6850274	7200.01
swath	28.02	0.81	91	41.24	915721	7200.00
timtab1	31.23	0.11	500	89.77	16470738	7200.00
timtab2	26.43	0.16	500	55.80	10265277	7200.00
tr12-30	52.52	0.19	500	61.45	4074819	7200.00