Math. Prog. Comp. (2013) 5:345–395
DOI 10.1007/s12532-013-0057-4

FULL LENGTH PAPER

On the safety of Gomory cut generators

Gérard Cornuéjols · François Margot · Giacomo Nannicini

Received: 17 May 2012 / Accepted: 3 July 2013 / Published online: 3 August 2013
© Springer-Verlag Berlin Heidelberg and Mathematical Optimization Society 2013

Abstract Gomory mixed-integer cuts are one of the key components in Branch-and-Cut solvers for mixed-integer linear programs. The textbook formula for generating these cuts is not used directly in open-source and commercial software that work in finite precision: Additional steps are performed to avoid the generation of invalid cuts due to the limited numerical precision of the computations. This paper studies the impact of some of these steps on the safety of Gomory mixed-integer cut generators. As the generation of invalid cuts is a relatively rare event, the experimental design for this study is particularly important. We propose an experimental setup that allows statistically significant comparisons of generators. We also propose a parameter optimization algorithm and use it to find a Gomory mixed-integer cut generator that is as safe as a benchmark cut generator from a commercial solver even though it generates many more cuts.

Mathematics Subject Classification 90C11 · 90C57

1 Introduction

Gomory mixed-integer (GMI) cuts [15] are one of the key components in Branch-and-Cut solvers for Mixed-Integer Linear Programs (MILPs) [5,6]. The textbook formula for generating these cuts is simple enough, but due to the limited numerical precision

G. Cornuéjols · F. Margot
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: gc0v@andrew.cmu.edu

F. Margot
e-mail: fmargot@andrew.cmu.edu

G. Nannicini (✉)
Engineering Systems and Design,
Singapore University of Technology and Design, Singapore, Singapore
e-mail: nannicini@sutd.edu.sg

of the computations, all open-source and commercial software that work in finite precision use additional steps to avoid the generation of invalid cuts. This paper studies the usefulness and practical impact of these safety-enhancing steps.

We perform statistical tests of several hypotheses related to these steps, in the context of a reasonable use of a GMI cut generator over a large enough and relevant set of instances. The use of the cut generator should be reasonable because we want to state properties that hold true in a practical Branch-and-Cut setting. The set of instances should be large enough so that we can draw statistically meaningful conclusions, and it should be relevant in the sense that it should contain the kind of instances that are routinely solved in real-world applications. The hypotheses that we want to test relate to the effectiveness of the safety-enhancing steps that are typically applied by existing cut generators. In particular, we would like to identify which steps are beneficial, irrelevant or harmful towards generating safe cuts.

In Sect. 2 we describe the cut safety-enhancing steps that we investigate. These steps are selected based on inspection of the open-source codes of COIN-OR [10] and `SCIP` [1], as well as discussion with developers of open-source and commercial solvers. They involve considering the fractionality of the basic integer variable used to generate the cut, the ratio between the largest and smallest absolute values of nonzero coefficients in the generated cut, the support of the cut, zeroing-out small coefficients, the violation of the cut by the current LP solution, and the relaxation of the right-hand side of the cut.

Numerical failures related to cut generation come in two flavors: generation of invalid cuts and difficulties in the LP reoptimization process. In Sect. 3 we propose a framework called DIVE-AND-CUT for the statistical analysis of such failures in cutting plane generators. Its basic idea is to generate a number of feasible solutions $S_{\mathcal{I}}$ for each test instance $\mathcal{I}$, and to perform the following experiment several times: randomly fix a number of integer variables in instance $\mathcal{I}$ to get an instance $\mathcal{I}_F$ such that solutions in a nonempty subset S_F of $S_{\mathcal{I}}$ are feasible for $\mathcal{I}_F$ (we call this a "dive"), generate several rounds of cuts for $\mathcal{I}_F$ and report whether a solution in S_F violates at least one of the generated cuts. A different type of failure occurs when the solver reports an infeasible LP but S_F is actually nonempty. Another milder kind of failure occurs when the computing time for solving the LP becomes prohibitively high due to numerical difficulties.

Even if feasible solutions to an integer program exist, they may not be representable or verifiable in finite precision arithmetics. Open-source and commercial software typically rely on absolute ϵ-feasibility tolerances. We propose a more robust definition of integer feasible solution that addresses many of the issues arising when employing an absolute ϵ-feasibility tolerance only. Our definition, introduced formally later in this section, allows a mild violation of the constraints, but guarantees that the solution lies very close to the feasible region. An algorithm to compute solutions that verify our definition is given in Sect. 3.

Our investigation focuses on two measures of performance of the cut generators, the *failure rate* (the fraction of dives where one of the numerical failures described in the previous paragraph occurred) and the *rejection rate* (the fraction of cuts that did not pass the safety-enhancing steps and were therefore discarded by the cut generator). We argue that a good generator should have a low failure rate for a given rejection rate,

while still generating strong cuts. Having a low failure rate is an obvious goal, while parameterizing according to the rejection rate allows to decouple cut generation from the cut selection process in Branch-and-Cut solvers. The ultimate goal is to select a family of cuts that minimizes the computing time of Branch-and-Cut algorithms. This is an extremely difficult question to answer for various reasons. One such reason is that aggressive unsafe cut generators may look more attractive than safe ones based on average computing time. An investigation of cut selection in the context of faster Branch-and-Cut solvers only makes sense if one compares cut generators that have similar levels of safety. The present paper focuses on the issue of cut safety. We plan to investigate cut selection for strength in future work.

Section 4 presents the instances used in the tests and studies the impact of some parameters of DIVE-AND-CUT on the number of failures (number of rounds of cutting, number of dives). We show that by increasing these two parameters we can increase the power of the statistical tests used to compare cut generators.

Section 5 reports on the variations in cut failure and cut rejection rates when modifying a single safety-enhancing parameter. We find that steps using the fractionality of the basic integer variable, the violation of the cut by the LP solution, or zeroing-out small coefficients have a significant impact on the safety of the generated cuts. We also find that the relaxation of the right-hand side should be done carefully. We observe a spike in invalid cuts when the right-hand side is relaxed by a constant close to the tolerance used to test if a value is integer.

This sets up the stage for Sect. 6, where we seek to "optimize" over all the parameters used in the safety-enhancing step. Our goal is to obtain a GMI cut generator with the following characteristics: its failure rate should be the same as or lower than that of the GMI cut generator of a commercial solver (we chose `Cplex` as our benchmark) and its rejection rate should be the lowest possible, subject to this constraint. We describe a black-box optimization algorithm to achieve this goal. Note that this algorithm does not consider the strength of the cuts when optimizing the safety-enhancing parameters. (We however verify in Sect. 7 that our best generators are not significantly weaker than generators typically used in practice.) Our philosophy is that solvers should pick good cuts in a second stage, among cuts that are deemed safe in a first stage.

In this paper, we focus on the first stage, which is to reject unsafe cuts. The cut generators we consider have twelve parameters and optimizing over all of them simultaneously would require an excessive amount of CPU time. We thus first use regression techniques to identify a set of six most influential parameters over which the optimization is performed. The remaining parameters are considered afterwards. We are able to find GMI cut generators that are as safe as the `Cplex` cut generator and that have a rejection rate around 40 %.

Section 7 validates the results of Sect. 6. We use a different set of test instances to compare five GMI cut generators obtained by our optimization algorithm to six cut generators from commercial or open-source solvers. The conclusions are that our generators are consistently safer than the cut generators (commercial or open-source) that have a similar rejection rate, and they accept many more cuts than the only generator (commercial) that has a similar safety. In addition, we observe that the gap closed at the root using any of our generators is comparable to the gap closed by usual generators. This gives hope that coupling an efficient cut selection procedure with the

safety-enhancing procedures described in this paper could yield safe and strong cut generators.

Finally, Sect. 8 concludes the paper summarizing our findings and providing a set of suggested parameter values for the generation of safe cuts.

1.1 Preliminaries

Consider a MILP in canonical form

$$\left.\begin{array}{rl} \min\ c^\top x & \\ Ax & \geq b \\ x & \in \mathbb{R}^n_+ \\ x_j & \in \mathbb{Z} \quad \text{for all } j \in N_I, \end{array}\right\} \tag{MILP}$$

where $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$, $A \in \mathbb{Q}^{m\times n}$ and $N_I \subset \{1,\ldots,n\}$. Lower (resp. upper) bounds on x are denoted by x^L (resp. x^U) and are included in $Ax \geq b$. Rows of A are denoted by a^i, $i = 1,\ldots,m$. For a positive integer k, we denote by $[k]$ the set $\{1,\ldots,k\}$ and by $\mathbf{0}_k$ the all-zero k-vector. The nearest integer to $z \in \mathbb{R}$ is denoted by $\lfloor z \rceil$. (MILP) can be expressed in standard form by defining $\hat{A} = (A,\ -I)$, $\hat{c}^\top = (c^\top,\ \mathbf{0}_m{}^\top)$ and appending m variables to the vector x. We assume that the first n components of x are the original variables. Variables numbered $n+1$ to $n+m$ are called *surplus* variables. We thus obtain

$$\left.\begin{array}{rl} \min\ \hat{c}^\top x & \\ \hat{A}x & = b \\ x & \in \mathbb{R}^{n+m}_+ \\ x_j & \in \mathbb{Z} \quad \text{for all } j \in N_I. \end{array}\right\} \tag{MILP$_s$}$$

The LP relaxation of (MILP$_s$) is the linear program obtained by dropping the integrality constraints, and is denoted by (LP). Let $B \subset [n+m]$ be an optimal basis of (LP), and let $J = [n+m] \setminus B$ be the set of nonbasic columns. Let B_I, J_I and J_C be the sets of integer basic variables, integer nonbasic variables, and continuous nonbasic variables respectively. The simplex tableau associated with B is given by

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

Choose $i \in B_I$ such that $x_i \notin \mathbb{Z}$. Define $f_0 := \bar{x}_i - \lfloor \bar{x}_i \rfloor$ and $f_j := \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$ for all $j \in J_I$. The GMI cut obtained from the row where x_i is basic is

$$\sum_{j\in J_I : f_j \leq f_0} f_j x_j + \sum_{j\in J_I : f_j > f_0} \frac{f_0(1-f_j)}{1-f_0} x_j + \sum_{j\in J_C : \bar{a}_{ij} \geq 0} \bar{a}_{ij} x_j - \sum_{j\in J_C : \bar{a}_{ij} < 0} \frac{f_0 \bar{a}_{ij}}{1-f_0} x_j \geq f_0. \tag{2}$$

The GMI cut (2) was introduced in Gomory's seminal paper [15] on mixed-integer linear programs. The cut (2) is generated from the problem in standard form (MILP$_s$), but virtually all Branch-and-Cut codes require the cut to be expressed in the space of the original variables before adding it to (LP). Therefore, surplus variables with nonzero cut coefficients must be substituted by their expression in terms of the original variables. When expressed in the space of original variables, the GMI cut is often called MIR inequality. In the following, for simplicity this cutting plane will be written as

$$\sum_{j\in[n]} \alpha_j x_j \geq \alpha_0, \tag{3}$$

or, if we need its expression in the $n+m$ space, as

$$\sum_{j\in[n+m]} \hat{\alpha}_j x_j \geq \hat{\alpha}_0. \tag{4}$$

Software using finite precision arithmetic works with tolerances for constraint violation. For MILPs, a natural choice is to use a tolerance for considering a number to be integer (ϵ_{int}), a tolerance for absolute violation of a constraint (ϵ_{abs}), and a tolerance for relative violation of a constraint (ϵ_{rel}). Available commercial solvers typically use $\epsilon_{\text{rel}} = \infty$. Given nonnegative values for ϵ_{int}, ϵ_{abs}, and ϵ_{rel}, we say that a point x^* is $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, \epsilon_{\text{int}}) - \mathit{feasible}$ for (MILP) if

(i) $\forall i \in N_I, x_i^* - \lfloor x_i^* \rceil \leq \epsilon_{\text{int}}$,
(ii) $\forall i \in [m], b_i - a^i x^* \leq \epsilon_{\text{abs}}$,
(iii) $\exists x' : Ax' \geq b, x' \geq 0, \|x^* - x'\| \leq \epsilon_{\text{rel}}$.

Point (iii) above is stronger than simply imposing a relative tolerance for individual constraints: $\forall i \in [m], (b_i - a^i x^*)/\|a^i\|_2 \leq \epsilon_{\text{rel}}$. The stronger condition is motivated by the following simple example. Suppose x^* satisfies at equality the relative feasibility tolerance for two rows of (MILP), say rows 1 and 2. Thus $(b_i - a^i x^*)/\|a^i\|_2 = \epsilon_{\text{rel}}$ for $i = 1, 2$. Now consider the constraint $(a^1 + a^2)x \geq b_1 + b_2$. It is redundant for (LP), but it is not satisfied by x^* according to the relative feasibility tolerance unless a^1 and a^2 are parallel. Indeed,

$$\frac{(b_1 + b_2) - (a^1 + a^2)x^*}{\|a^1 + a^2\|_2} = \frac{\epsilon_{\text{rel}}(\|a^1\|_2 + \|a^2\|_2)}{\|a^1 + a^2\|_2} \geq \epsilon_{\text{rel}}$$

and the inequality is strict unless a^1 and a^2 are parallel. In other words, even though the Euclidean distance between x^* and each violated constraint is at most ϵ_{rel}, the distance between x^* and the closest point in the feasible set of the LP could be much larger than ϵ_{rel}. In this situation, a cutting plane that cuts off x^* should not be marked as invalid. Therefore we do not want x^* in our set of feasible solutions. In the following, whenever we say *feasible solution* we refer to an $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, \epsilon_{\text{int}})$-feasible solution unless otherwise noted. We remark that these solutions could be infeasible if feasibility were checked in infinite precision. However, solutions returned by commercial solvers are typically accepted in practice despite using $\epsilon_{\text{rel}} = \infty$. In this paper we use the more

restrictive choice $\epsilon_{\text{rel}} < \infty$, which guarantees that the solutions are inside or close to the feasible region using any standard metric.

A *cut generator* is an algorithm whose input is a simplex tableau for the LP relaxation of an MILP and whose output is a set of cuts. Safety-enhancing steps are part of the cut generator. A cut generator can produce *failures* of the following types.

Type 1: A cutting plane that cuts off a known integral feasible solution is generated.
Type 2: The linear relaxation becomes infeasible after the addition of the generated cutting planes, but an integral feasible solution is known.
Type 3: A time limit for cut generation and LP resolve is hit.

Failures of Type 1 and 2 depend on the precision of the machine and of the computations. A Type 3 failure depends on the time limit that is set, and can be seen as less severe than failures of Type 1 and 2 because it may not be the result of unsafe computations. However, if the time limit is sufficiently large, a Type 3 failure renders the cut generator impractical and therefore we consider it a defect.

Ideally, no failure should occur when the cut generator is used in a Branch-and-Cut algorithm. However, practitioners in integer programming know that numerical problems are not uncommon, and if the choice of the cut generation parameters is not careful, failures of the three types listed above can happen [23]. To decrease the occurrence of failures, several cut safety-enhancing steps have been devised empirically. Some of these steps modify the cut, while others are numerical checks that result in the acceptance or rejection of the cut. Rejected cuts are simply discarded.

1.2 Related work

A step in the direction of testing safety of cutting plane generators is taken in [23]. The paper proposes a methodology for comparing the strength of cut generators. Safety is a fundamental issue in this context, since only cut generators with a similar safety can be compared on equal ground with respect to strength (otherwise, the comparison would favor "unsafe" but more aggressive cut generators). [23] provides experimental evidence that existing cut generators in COIN-OR `Cgl` [10] may run into numerical problems on seemingly innocuous instances.

A Branch-and-Bound solver that relies on rational arithmetic and thus is not affected by the numerical problems mentioned above, is described in [14]. A combination of the rational LP solver [4] with the Branch-and-Cut code `SCIP` [1] is in progress. The idea is to use finite precision arithmetic for the majority of the computations, and switch to (slower) rational arithmetic only for those operations that would invalidate optimality of the result if carried out in a non-exact fashion (such as pruning based on dual bounds). More details can be found in [13]. Note however that the implementations described in [13,14] do not include cutting planes.

The generation of numerically safe GMI cuts has been investigated in [24] when all the variables are bounded, and in [12] in general. These safe cuts are generated in floating-point arithmetic using a clever rounding scheme and they are guaranteed to be satisfied by all feasible solutions of the MILP. However, [12, (p. 641)] stresses that these cuts are guaranteed to be valid only "when computations to evaluate the

inequality are performed in infinite precision." They are even more explicit in their conclusion [12, (p. 648)]:

> This does not mean, however, that if we add these cuts instead of unsafe Gomory cuts in current floating-point-based MIP solvers that we are guaranteed (or even have a better chance) to obtain the correct optimal solution. For example, a solver could incorrectly assert that a solution to the MIP problem violates a safe Gomory cut if it evaluated the cut in finite precision arithmetic.

The generation of safe GMI cuts in finite precision arithmetic is particularly useful when using a hybrid solver that performs most computations in finite precision for speed but switches to infinite precision once in a while to guarantee correctness. However, when using a finite precision state-of-the-art solver, we observed no significant difference in the safety of the GMI cuts when turning on or off the safe rounding scheme in the code of [12]. Specifically, out of 5,240 experiments using the method of [23] on 29 MIPLIB3 instances, we observed 1,773 overall failures with the safe generator of [12] (38 failures of Type 1, 148 failures of Type 2 and the remaining ones of Type 3, as discussed in the previous section) and 1,737 failures when safe rounding was turned off (19 failures of Type 1, 128 failures of Type 2 and the rest of Type 3). The differences are not statistically significant but, clearly, the safe rounding scheme has little impact when used with a finite precision solver. This is explained by the fact that the rounding scheme of [12] typically affects the 15th significant digit of the coefficients while the error in computing the left-hand-side of a GMI cut is often orders of magnitude greater, say 10^{-9} or larger.

2 Cut generation and safety-enhancing parameters

Generating GMI cuts using finite precision computations involves three basic nonnegative parameters.

(i) ZERO: Any number $z \in \mathbb{R}$ such that $|z| \leq$ ZERO is replaced by zero;
(ii) EPS, EPS_REL: Any two numbers $z, w \in \mathbb{R}$ such that $|z - w| \leq \max\{\text{EPS}, \text{EPS\_REL} \cdot \max\{|z|, |w|\}\}$ are considered to be equal.

The choice EPS_REL $= 0$ is common in practice.

Two broad classes of cut safety-enhancing procedures are *cut modifications* (modifying the coefficients or the right-hand side of the cut) and *numerical checks* (performing checks on the cut in order to either accept or reject it). We now describe the safety-enhancing procedures that we consider in this paper.

2.1 Cut modification

In a typical GMI cut generator, each cut computed by the Gomory formula (2) is modified by up to three procedures before being added to (LP).

(i) COEFFICIENT REMOVAL: First, very small cut coefficients for surplus variables are removed without adjusting the right-hand side. Then the cut is expressed in

the original space. Finally, small cut coefficients are removed, possibly adjusting the right-hand side of the cut to ensure its validity.

(ii) RIGHT- HAND SIDE RELAXATION: The right-hand side of the cutting plane is relaxed to generate a safer cut.

(iii) SCALING: The coefficients and right-hand side of the cut are scaled by a positive number.

COEFFICIENT REMOVAL is applied by all open-source cut generators in `Cgl` [10] and `SCIP` [1]. The purpose of removing cut coefficients for surplus variables before substituting their expression in terms of the original variables is to save computing time. RIGHT- HAND SIDE RELAXATION and SCALING procedures are not always employed. SCALING can be performed in various ways that may significantly differ. For example, one can scale to obtain the largest cut coefficient equal to 1 or scale to obtain integral cut coefficients. Note that SCALING affects the absolute violation of the cut at a point $\bar{x}$, i.e. the value of $\alpha_0 - \alpha\bar{x}$. Because there is no standard SCALING procedure in the cut generators that we examined, and because we have computational evidence that SCALING is not beneficial in our framework (remember that we use a relative feasibility tolerance), we postpone an analysis of SCALING until Sect. 6.6. For now we concentrate on the first two of the above modification procedures. They require the following parameters.

(i) `EPS_ELIM`: For $j \in \{n+1, \ldots, n+m\}$, cut coefficients $\hat{\alpha}_j$ such that $|\hat{\alpha}_j| \leq$ `EPS_ELIM` are set to zero, without substituting the corresponding surplus variable with its expression in terms of the original variables;

(ii) `LUB`: For $j \in [n]$, a variable x_j with $x_j^L = \beta$ or $x_j^U = \beta$ for some β such that `LUB` $\leq |\beta| < \infty$ is considered having a large bound. Define L as the set of such variables;

(iii) `EPS_COEFF`: For $j \in [n] \setminus L$, cut coefficients α_j such that $|\alpha_j| \leq$ `EPS_COEFF`, are set to zero, adjusting the right-hand side of the cut as follows. If $\alpha_j > 0$ (resp. $\alpha_j < 0$), the right-hand side α_0 becomes $\alpha_0 - \alpha_j x_j^U$ (resp. $\alpha_0 - \alpha_j x_j^L$) unless $x_j^U = \infty$ (resp. $x_j^L = -\infty$), in which case the cut is discarded;

(iv) `EPS_COEFF_LUB`: For $j \in [n] \cap L$, cut coefficients α_j such that $|\alpha_j| \leq$ `EPS_COEFF_LUB` are set to zero and no adjustment of the right-hand side occurs; typically `EPS_COEFF_LUB` is much smaller than `EPS_COEFF`;

(v) `EPS_RELAX_ABS`: The cut right-hand side α_0 is relaxed to $\alpha_0 -$ `RELAX_RHS_ABS`;

(vi) `EPS_RELAX_REL`: The cut right-hand side α_0 is relaxed to $\alpha_0 - |\alpha_0| \cdot$ `RELAX_RHS_REL`.

2.2 Numerical checks

All generated cutting planes undergo a sequence of checks aimed at deciding whether or not they should be added to (LP). These checks test the numerical properties of the cuts, as well as their effectiveness. The *support* of a cut is the set of all variables whose coefficient is nonzero. In a typical cut generator, the following checks are performed.

(i) FRACTIONALITY CHECK: A cut is discarded (rather, not generated), if the value of the corresponding integer basic variable is too close to an integer value;
(ii) VIOLATION CHECK: A cut is discarded if it does not cut off the optimal solution to (LP) by at least a given amount;
(iii) SUPPORT CHECK: A cut is discarded if the cardinality of its support is too large;
(iv) RATIO CHECK: A cut is discarded if the ratio between the largest and smallest absolute values of the nonzero coefficients is too large. In the literature, this ratio is often referred to as the DYNAMISM of the cut;
(v) SCALING CHECK: A cut is discarded if it is badly scaled.

The SCALING CHECK is not regulated by a single parameter. For example, a cut might be discarded if its ℓ_2-norm does not fall within a given range. A study of scaling is postponed until Sect. 6.6. The other four checks require the following parameters. Let $\bar{x}$ denote the current basic solution as defined in (1).

(i) `AWAY`: The cut generated from the tableau row where x_i is basic is discarded if $|\bar{x}_i - \lfloor \bar{x}_i \rceil| <$ `AWAY`;
(ii) `MIN_VIOL`: The cut is discarded if $\alpha_0 - \sum_{j\in[n]} \alpha_j \bar{x}_j < \max\{1, |\alpha_0|\} \cdot$ `MIN_VIOL`;
(iii) `MAX_SUPP_ABS`, `MAX_SUPP_REL`: The cut is discarded if the support of α is larger than `MAX_SUPP_ABS` $+ n \cdot$ `MAX_SUPP_REL`;
(iv) `MAX_DYN`: The cut is discarded if $\max\{|\alpha_j| : j \in [n]\} >$ `MAX_DYN` $\cdot \min\{|\alpha_j| : |\alpha_j| > 0, j \in [n]\}$ and $L \cap \{j : |\alpha_j| > 0\} = \emptyset$;
(v) `MAX_DYN_LUB`: The cut is discarded if $\max\{|\alpha_j| : j \in [n]\} >$ `MAX_DYN_LUB` $\cdot \min\{|\alpha_j| : |\alpha_j| > 0, j \in [n]\}$ and $L \cap \{j : |\alpha_j| > 0\} \neq \emptyset$.

Observe that any of the cut modification and numerical check procedures can be disabled by setting the corresponding parameter to an appropriate value. For example, using `MAX_SUPP_ABS` $= n$ implies that no SUPPORT CHECK is performed.

3 Dive-and-Cut

In this section we propose a method for testing the safety of cut generators. It assumes that a set of problem instances is available. For each instance a preliminary Solution-Generation step is applied. The goal of this step is the generation of many feasible (or almost feasible) solutions.

Once the Solution-Generation step has been completed, the testing phase for an instance amounts to diving randomly by fixing a number of integer variables, and then generating rounds of cuts. In the testing phase, standard features of the LP solver such as presolving are turned on. The main task is to solve a sequence of LPs, to generate the corresponding GMI cuts and to check their validity using the known feasible (or almost feasible) solutions.

This scheme is similar to the method RANDOMDIVES proposed in [23], in the sense that the work horse is a large number of random dives to be able to perform meaningful statistical tests. However, DIVE-AND-CUT improves over RANDOMDIVES on the three criteria mentioned in Sect. 1 (reasonable use of the generator, large and relevant set of instances). In addition, DIVE-AND-CUT is usually faster than RANDOMDIVES.

3.1 Solution-Generation phase

Testing invalid cuts requires the knowledge of feasible solutions. Branch-and-Cut solvers typically accept $(\epsilon_{\text{abs}}, \infty, \epsilon_{\text{int}})$-feasible solutions with positive and finite values for ϵ_{abs} and ϵ_{int}. Suppose that for problem (MILP) we have an $(\epsilon_{\text{abs}}, \infty, \epsilon_{\text{int}})$-feasible solution $\tilde{x}$, and there exists at least one row a^i such that $b_i - a^i\tilde{x} > 0$. Thus, we can find $\lambda \in \mathbb{R}^m_+$ such that $\lambda^\top(b - A\tilde{x}) > \epsilon'$ for arbitrary $\epsilon' > 0$. In other words, we can find a valid inequality $\alpha x \geq \alpha_0$ for the system $Ax \geq b$, with $\alpha = \lambda^\top A$ and $\alpha_0 = \lambda^\top b$, that is violated by $\tilde{x}$ by an arbitrary amount. It follows that we should be careful when choosing the solutions that are used for testing the validity of the cuts. On the one hand, using slightly infeasible solutions may lead to mislabeling cuts as invalid. On the other hand, commercial MILP solvers typically return slightly infeasible solutions that are often acceptable for practical purposes, therefore cutting off such a solution can reasonably be considered a failure of the cut generator.

We use algorithm GENERATESOLUTIONS given in Appendix A to generate $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, 0)$-feasible solutions for an instance, with positive and finite values for ϵ_{abs} and ϵ_{rel}. It applies a Branch-and-Cut solver and acts whenever the solver discovers an integer solution. First, integer variables are set to integer values and this updated solution is checked for $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, 0)$-feasibility. If it satisfies both the absolute violation tolerance ϵ_{abs} and the relative violation tolerance ϵ_{rel} for each constraint, a rational solver is used to find a feasible solution close to the updated solution, and this solution is used to check whether condition (iii) in the definition of $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, \epsilon_{\text{int}})$-feasibility is also satisfied. Details are in Appendix A.

3.2 Testing phase

In this section we assume that for each instance in our test set, a collection of feasible solutions is available. These solutions will be used to detect invalid cuts. A formal description of the method that we propose is given in Algorithm 1.

We call this algorithm DIVE-AND-CUT. It starts by diving towards a feasible solution x^* chosen at random among the available solutions. This is achieved by selecting uniformly at random a value t between 0 and T (we use $T = 80\,\%$ in our experiments) and fixing randomly chosen integer variables to their value in x^* until a fraction t of the initial gap is closed. The gap is computed with respect to a given upper bound U. In this paper we set U to the value of the best solution returned by GENERATESOLUTIONS (in the vast majority of cases, this is the same as the value of the best known solution for the instance, see Appendix A.2). This simulates the generation of a node of a hypothetical Branch-and-Cut tree. Once at this node, DIVE-AND-CUT solves the LP, generates a round of GMI cuts from the rows of the optimal tableau, adds these cuts to the formulation, resolves the LP, and repeats this process generating ρ rounds of cuts. If we hit the time limit during cut generation, or the LP is infeasible, or we cut off any feasible solution, the algorithm returns a failure. Otherwise, it returns that no failure occurred. Note that we only test against the feasible

Algorithm 1 DIVE-AND-CUT.

INPUT: Problem $P = (A, b, c)$, set of solutions S, maximum gap threshold T, upper bound U, number of rounds ρ, tolerances $\epsilon_{\text{abs}}, \epsilon_{\text{rel}} \geq 0$, time limit for a dive
OUTPUT: A failure flag: 1 for an invalid cut, 2 for an infeasible LP, 3 for exceeding the time limit, 0 for no failure.
Let $F \leftarrow \emptyset$
Randomly choose $x^* \in S$
Randomly choose $t \in [0, T]$
Compute $\bar{x} = \arg\min\{c^\top x | Ax \geq b, x \geq 0\}$
Initialize $\bar{x}' \leftarrow \bar{x}$
while $(c^\top(\bar{x}' - \bar{x}) < t \cdot (U - c^\top \bar{x}))$ **do**
  Randomly choose $j \in \{i \in N_I \backslash F\}$
  Append the constraint $x_j = x_j^*$ to (A, b)
  Let $F \leftarrow F \cup \{j\}$
  Compute $\bar{x}' = \arg\min\{c^\top x | Ax \geq b, x \geq 0\}$
Compute $S(x^*, F) = \{x \in S | x_i = x_i^* \ \forall i \in F\}$
for $1, \ldots, \rho$ **do**
  Generate cuts $\alpha^i x \geq \alpha_0^i, i = 1 \ldots, h$ and append to (A, b)
  Resolve $\min\{c^\top x | Ax \geq b, x \geq 0\}$
  if (time limit is hit) **then**
    return failure $\leftarrow 3$
  else if (LP is infeasible) **then**
    return failure $\leftarrow 2$
  else if $(\exists \tilde{x} \in S(x^*, F) : ((\max_{i \in [h]}\{\alpha_0^i - \alpha^i \tilde{x}\} > \epsilon_{\text{abs}}) \vee (\max_{i \in [h]}\{(\alpha_0^i - \alpha^i \tilde{x})/\|\alpha^i\|\} > \epsilon_{\text{rel}})))$ **then**
    return failure $\leftarrow 1$
  Perform cut management
return failure $\leftarrow 0$

solutions in the set S that have the same value as x^* on the variables that have been fixed.[1]

This method is designed to represent a reasonable use of a cut generator. In the majority of Branch-and-Cut solvers, cutting planes are mostly generated at the root node and cut management procedures are used. In DIVE-AND-CUT the node obtained after the dive mimics a root node. DIVE-AND-CUT involves several random decisions, therefore the algorithm can be applied as many times as required to obtain statistically significant results.

4 Empirical testing: preliminaries

In this section we describe the framework for the empirical testing of GMI cut generators conducted using DIVE-AND-CUT.

4.1 Parameters and implementation

We list here the parameters and implementation features used throughout the computational experiments. `ZERO` is set to 10^{-20}, `EPS` and `EPS_REL` are set to 10^{-12}.

[1] This is not necessary if the cuts being tested can be generated independently of the bounds on the variables, or are lifted to be globally valid.

A number α is considered integer valued if $|\alpha - \lfloor \alpha \rceil| \leq \max\{10^{-9}, 10^{-15} \cdot |\alpha|\}$. The absolute feasibility tolerance ϵ_{abs} is set to 10^{-9}, the relative feasibility tolerance ϵ_{rel} is set to 10^{-9}. The number ρ of rounds of cut generation is set to 30, unless otherwise stated (see discussion in Sect. 4.4).

Throughout our code except in the cut generator itself, the computation of all sums of a sequence of numbers (e.g., dot product, norm) is carried out with the compensated summation algorithm [20] to compute the left-hand side of inequalities. (Compensated summation ensures that the numerical error is independent of the number of additions.) In the GMI cut generator, compensated summation is not used, as it is not standard practice in commercial and open-source Branch-and-Cut solvers. The GMI cut generator recomputes the simplex tableau from scratch, using the basis information, instead of obtaining it directly from the LP solver. We experimented with using the tableau provided by `Cplex`, but in our framework we did not detect any difference in safety of the generated cuts.

The algorithms discussed in this paper are implemented in C++ within the COIN-OR framework. We use several functions available in COIN-OR `Cbc` 2.7 [8]. The GMI cut generator is implemented as a `CglCutGenerator`, following the guidelines of `Cgl` [10]. The LP solver of choice is IBM ILOG `Cplex` 12.2 [19].

A small part of the experiments required a manageable amount of time on a single machine: all experiments discussed in Sects. 4.4, 6.1, 6.6 and 7. These tests were executed on a machine equipped with an `AMD Opteron 4176 HE` processor clocked at 2.4 GHz and 48 GB RAM, running Linux. Because the required CPU time was manageable, we performed 300 dives with a time limit of 600 s per dive. In the rest of the paper, we refer to this setup as the *single-machine setup*.

Due to the huge amount of processing time required, most of the experiments were run in parallel on the Condor [22] grid at the University of Wisconsin-Madison: all experiments discussed in Sects. 4.2, 5, 6.3, 6.4, and 6.5. Compared to the single machine setup, these tests use only 150 dives with a time limit of 300 s per dive, to reduce computing time. In the rest of the paper we refer to this setup as the *Condor setup*. All machines running Linux in the Condor pool were candidates for executing the experiments. For this reason, our code is compiled to run on a generic x86 architecture. Since we use different machines, some variation on the results of the computations across machines should be expected. A preliminary computational evaluation revealed that this is not a major problem, as the differences recorded by running the same experiment several times were not statistically significant.

4.2 Instance selection and cut management

In order to have a large and diverse set of instances to test the cut generators, we built an initial test set containing all instances from MIPLIB3 [7], MIPLIB2003 [2], and the Benchmark set of MIPLIB2010 [21] beta (downloaded March 2011) for a total of 169 instances.

For each instance in the set, we applied `Cplex`'s Branch-and-Cut and GENERATESOLUTIONS in order to generate the set of feasible solutions (see Appendix A).

Table 1 Parameters defining the cut generator CGBASE

Parameter	Value	Parameter	Value	Parameter	Value
AWAY	10^{-9}	MAX_DYN	∞	EPS_COEFF	0
MIN_VIOL	$-\infty$	MAX_DYN_LUB	∞	EPS_COEFF_LUB	0
MAX_SUPP_ABS	∞	EPS_ELIM	0	EPS_RELAX_ABS	0
MAX_SUPP_REL	∞	LUB	∞	EPS_RELAX_REL	0

Table 2 Number of failures on the full test set, minus instances on which no failures of Type 1 or 2 were recorded with CGBASE

k	Failures			
	T. 1	T. 2	T. 3	Tot.
1	300	63	150	513
2	436	262	331	1,029
3	391	279	353	1,023
∞	353	238	518	1,109

This test set comprises 74 instances. The value of k is the number of consecutive rounds of inactivity after which cuts are removed from the LP

As GENERATESOLUTIONS fails to generate any feasible solution for ten of the instances, we are left with 159 instances.

Since running experiments on instances that do not generate failures is useless in the present context, we keep only instances for which a crude GMI cut generator called CGBASE generates some failures. CGBASE is the most basic cut generator that can be designed given our parameters, as it accepts all cuts generated from rows whose basic variable has a fractionality exceeding the integrality tolerance. Its parameterization is given in Table 1.

The test runs consist in applying DIVE-AND-CUT in the Condor setup with the tolerances described in Sect. 4.1. Four cut management procedures are tested. We say that a generated cut is *inactive* in an optimal solution of (LP) if its dual variable has a value smaller than 10^{-5}. The four cut management procedures that we considered are to remove all cuts that are inactive for k consecutive rounds for $k = 1, 2, 3$ and $k = \infty$.

We also allow for an early stopping criterion: if more than 20 failures of Type 3 are detected on an instance with a given cut management procedure, the execution of DIVE-AND-CUT is stopped. Overall, the experiment required more than 3,000 h of CPU time. All instances on which no failures of Type 1 or 2 occurred are removed.

The test runs show that the number of failures decreases if we remove inactive cuts more aggressively. At the same time, CPU time decreases, which is expected. Experiments where cuts are never removed from the LP turn out to be very time-consuming, with a significant number of Type 3 failures. The number of recorded failures for the four cut management procedures are given in Table 2.

It can be seen from Table 2 that the number of failures significantly increases when we do not remove all inactive cuts immediately. On the other hand, there is little

difference between $k = 2, 3$ and ∞. Indeed, the total number of failures is relatively stable for these three values of k. The fact that the number of Type 1 and 2 failures decreases between $k = 3$ and $k = \infty$ can be explained by observing that some of the Type 3 failures might generate failures of Type 1 or 2, if given more time. Furthermore, more instances time out, hence we perform fewer dives overall because of the early stopping criterion. Since we are interested in producing a large number of failures as quickly as possible, we will use the cut management procedure with $k = 2$ in the remainder of the paper.

The final modification to the test set consists in removing the instances taking too much time with $k = 2$. We remove instances where more than 10 failures of Type 3 are recorded or such that 150 dives take more than 5 hours. We are left with a set of 51 instances that we call FAILURE SET, see Appendix A.2.

4.3 Statistical tests

In this section, we briefly cover the application of statistical tests to the analysis of results of $c \geq 2$ algorithms on r instances. In the usual presentation of these tests in statistics textbooks, algorithms are referred to as "treatments" and instances are referred to as "blocks" or "subjects". For a complete presentation of these tests, see [11,27] or any reference statistics book.

The Friedman test used in the analysis of our results is a non-parametric test, i.e., a test that does not assume any form for the distribution of the population data. The test assumes that a null hypothesis is true and gives the probability (the p value) of obtaining a test statistic at least as extreme as the one observed. The p value is then compared to a given α value (we use $\alpha = 0.05$ in this paper) for a test with significance $(1-\alpha)$. If the p value is smaller than α, the null hypothesis is rejected, as the observed results have a low probability of occurring if the null hypothesis were true.

Friedman test:

(i) Application: (1) each instance is solved by all c algorithms; (2) the outcome of using an algorithm on an instance is a real value called the performance of the algorithm on that instance.
(ii) Null hypothesis: The c algorithms have similar performances; Alternative hypothesis: There is a difference in performance between some of the algorithms.
(iii) Assumptions: (1) The set of r instances is a random sample of the population; (2) The r c-variate random variables are mutually independent; (3) The outcome is a continuous random variable.

We use the Iman-Davenport's variant of the Friedman test, known to be more accurate than the original version of the test [11]. Note that the Friedman test is based only on the ranking of the performance of the algorithms on each instance. It does not take into account the magnitude of the differences in performance.

When we say "we apply a Friedman test on the failure rate", the performance of a cut generator on an instance is the average failure rate over all dives on that instance. The null hypothesis is that the cut generators are indistinguishable in terms of their failure rate.

Similarly, when "we apply a Friedman test on the rejection rate" the performance of a cut generator on an instance is the average rejection rate over all dives on that instance. In this case, the null hypothesis is that the cut generators are indistinguishable in terms of their rejection rate.

When the Friedman test rejects the null hypothesis, an additional statistic can be used to test if algorithm A has a better performance than algorithm B, for each pairs A, B. The result of all pairwise comparisons is not always a total order, as transitivity is not guaranteed.

We use a 2-dimensional table for displaying the result of pairwise comparisons. Rows and columns of the table corresponds to the tested algorithms. Entry in cell in row i and column j is a "+" sign (resp. "−" sign) if the algorithm in row i has larger (resp. smaller) performance value than the one in column j at the given significance level. A "=" means that no difference could be detected at the given significance level. All comparisons in this paper are carried out at a significance level of 95 %.

4.4 Number of rounds and number of dives

The number ρ of rounds of cuts to be generated is one of the key decisions in cutting plane algorithms. In this section we show that this choice does not affect the conclusions that can be drawn by applying DIVE-AND-CUT, in the sense that increasing ρ simply increases the power of the statistical tests that we use but does not change the safety rankings of cut generators. The same is true for the number of dives. We provide some details on our study of failures as a function of ρ.

The data is obtained by applying DIVE-AND-CUT in the single-machine setup on the 51 instances of the FAILURE SET with the cut generator CGBASE (see Table 1). For each type of failure, we plot in Fig. 1 the points (r, f) where f is the number

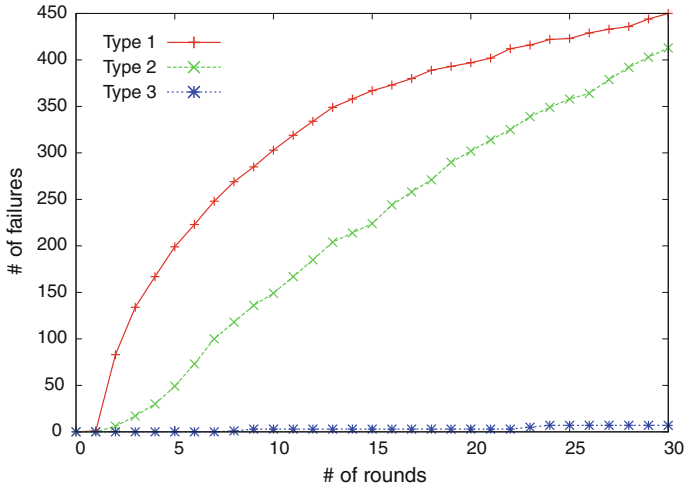

Fig. 1 Number of failures depending on the value of ρ (x axis)

of failures that occurred up to round r. Similar graphs are obtained with other cut generators, therefore we only report results for CGBASE.

An interesting fact that can be observed in Fig. 1 is that the number of Type 1 failures is a concave function of ρ and the number of failures of Type 2 and 3 increases almost linearly with ρ. This is surprising, as we expected very few failures in the first few rounds and a super-linear increase for larger values of ρ. The importance of this finding lies in the fact that we can increase the number of rounds to increase the number of failures without putting an unreasonable stress on the generators. This helps in detecting differences between otherwise indistinguishable cut generators, without severely affecting the ranking of the generators. We verified this claim by comparing the safety of a set of cut generators for $\rho = 5, 10, 20$, and 30 using a Friedman test on the failure rate. Some differences among the cut generators that are detected for $\rho \geq 20$ are not detected for $\rho = 5, 10$.

To summarize, the number of rounds of cut generation and the number of dives have a direct influence on the detection power of our tests. By increasing those two parameters, we can magnify the differences between cut generators, at the expense of requiring more computing time.

5 Empirical testing: parameter ranges

In this section we discuss one-at-a-time changes in the cut generation parameters. The base cut generator in this section is CGBASE described in Table 1. All experiments were run in the Condor setup.

For each parameter listed in Sects. 2.1 and 2.2, we perform several tests over the range of possible values, recording the number of failures and the average rejection rate as the parameter value changes. This information is used in Sect. 6.3 to determine the initial parameter ranges for the optimization.

For the sake of brevity, instead of reporting extensive statistical tests for each parameter value, we simply graph the number of failures and cut rejection rate for a range of possible values of the parameter.

5.1 Variables with large bounds

The `LUB` parameter theoretically takes value in $[0, \infty]$. There are only 5 instances in FAILURE SET for which `LUB` $\geq 10^4$ yields a different set L of variables with a large bound compared to `LUB` $= 10^3$: `bell4a` ($|L|$ decreases from 92 to 62), `blend2` ($|L|$ decreases from 90 to 88), `maxgasflow` ($|L|$ decreases from 4,920 to 4,912) `noswot` ($|L|$ decreases from 53 to 28), `roll3000` ($|L|$ decreases from 244 to 6 for `LUB` $= 10^4$, 2 for `LUB` $> 10^4$). For the remaining instances, any value of `LUB` $\geq 10^3$ yields the same set of variables with a large bound.

The value of the `LUB` parameter affects both `MAX_DYN_LUB` and `EPS_COEFF_LUB` and these parameters are hard to decouple. Therefore we do not analyze them in this section where the focus is on one-at-a-time changes.

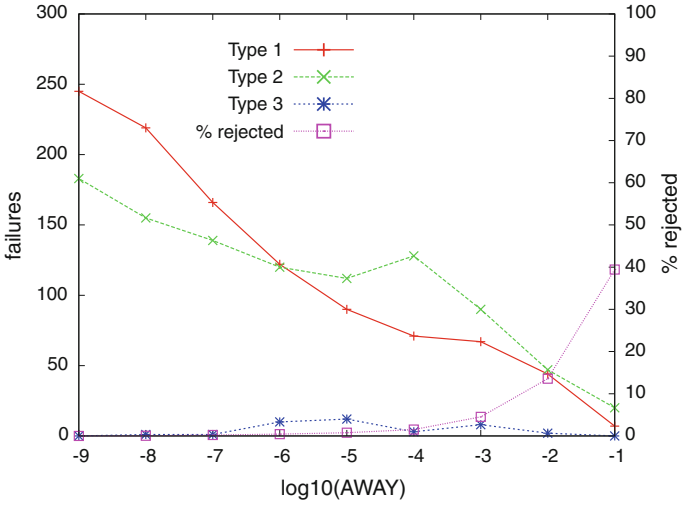

Fig. 2 Number of failures and cut rejection rate for cut generators with different values of AWAY. AWAY is set to 10^k, where k is the value on the x axis

5.2 Numerical check parameters

In this section we analyze the effect of varying separately each parameter given in Sect. 2.2. In each case, we plot the number of failures and the rejection rate.

5.2.1 Fractionality of the right-hand side

The AWAY parameter takes its value in [0, 0.5]. Since the integrality tolerance is 10^{-9}, we use a lower bound of 10^{-9} for AWAY. We tested the values AWAY $= 10^k$ for $k = -9, \ldots, -1$. Smaller values of AWAY lead to generating more cuts and more failures.

Figure 2 graphs the number of failures and percentage of rejected cuts as a function of k. It shows that generating cuts with small AWAY is extremely risky and leads to many failures. Since a small value of AWAY allows for the generation of cuts with large coefficient ratios, the generated unsafe cuts could possibly be discarded through RATIO CHECK, but such interactions will only be considered later through the optimization algorithm of Sect. 6.2. By increasing AWAY, a much safer cut generator can be obtained, while still rejecting very few cuts. The rejection rate starts increasing to non-negligible levels only for AWAY $> 10^{-5}$.

5.2.2 Ratio test

The MAX_DYN parameter used in the ratio test takes value in $[0, \infty]$. We tested the values MAX_DYN $= 10^k$ for $k = 2, 4, \ldots, 30$.

Figure 3 shows that the cut rejection rate decreases at a low rate while k increase from 2 to 16, decreases sharply when k increases from 16 to 26, and is almost 0 when

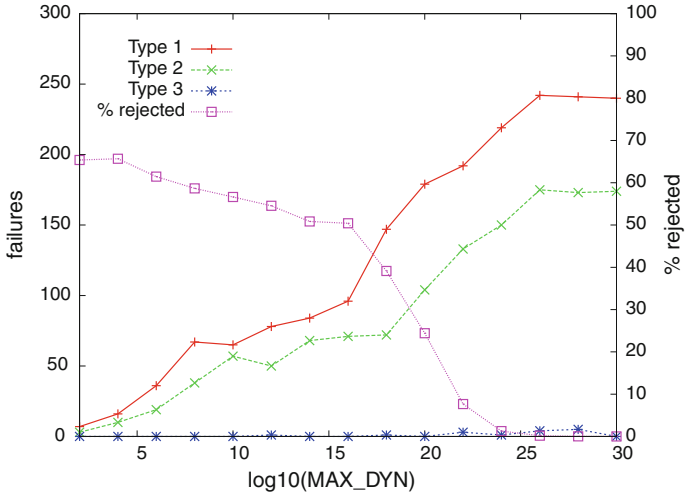

Fig. 3 Number of failures and cut rejection rate for cut generators with different values of MAX_DYN. MAX_DYN is set to 10^k, where k is the value on the x axis

k is larger than 26. This implies that most of the generated cuts when AWAY $= 10^{-9}$ have very poor numerical properties. We suspect that this happens especially in later rounds. By relying on RATIO CHECK only, halving the number of Type 1 and 2 failures comes at the cost of rejecting more than half of the generated cuts. If we only accept cuts with MAX_DYN $= 10^2$, just 10 failures are recorded overall, but almost two thirds of the cuts are rejected. Interestingly, for $k < 22$ essentially no Type 3 failures are recorded. This suggests that adding cuts with large coefficient ratios makes the LP solution process significantly more time-consuming.

5.2.3 Violation

The MIN_VIOL parameter theoretically takes its value in [0, 1], but due to errors in the finite precision computations, applying VIOLATION CHECK even with MIN_VIOL $=$ 0 could reject some cutting planes. We tested the values MIN_VIOL $= 10^k, k = -10, \ldots, -1$, and MIN_VIOL $= 0$ (which is reported as $k = -$inf in the figure for sake of simplicity).

Figure 4 shows that even small values of MIN_VIOL are surprisingly effective in reducing the number of failures. The number of Type 1 and 2 failures can be reduced by 50 % by rejecting < 2 % of the generated cuts. For $k > -6$, the fraction of rejected cuts begins to rise sharply.

5.2.4 Maximum support

The maximum allowed support for the cuts is kept under control by two parameters: MAX_SUPP_ABS that takes value in $[0, \infty]$, and MAX_SUPP_REL that takes value in [0, 1]. The largest instance in FAILURE SET has 10,724 columns, there-

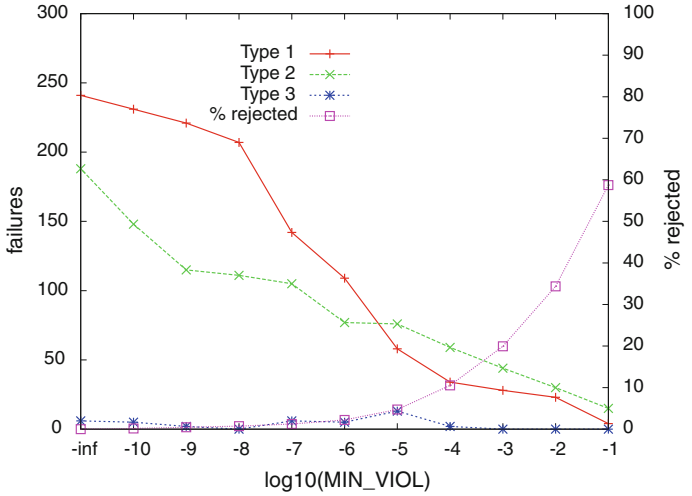

Fig. 4 Number of failures and cut rejection rate for cut generators with different values of `MIN_VIOL`. `MIN_VIOL` is set to 10^k, where k is the value on the x axis

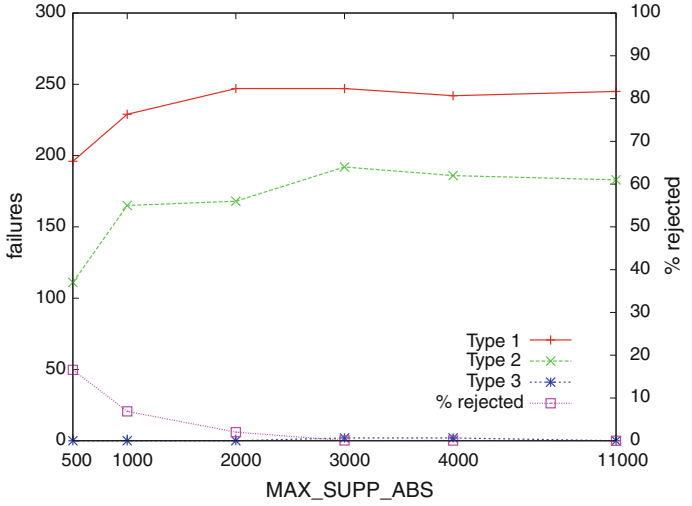

Fig. 5 Number of failures and cut rejection rate for cut generators with different values of `MAX_SUPP_ABS`

fore we can consider `MAX_SUPP_ABS` to take value in $[0, 10{,}724]$. We report results for the values `MAX_SUPP_ABS` $= 500, 1{,}000, 2{,}000, 3{,}000, 4{,}000, 11{,}000$. For `MAX_SUPP_REL`, we report results for the values 0.1, 0.2, 0.5, 0.8, 0.9, 1.0. Note that several cut generators in COIN-OR `Cgl` and `SCIP` have a nonzero value for both parameters, but for simplicity here we test the parameters one at a time.

Figure 5 graphs the number of failures and percentage of rejected cut depending on the value of `MAX_SUPP_ABS`, whereas Fig. 6 reports the same information depending on the value of `MAX_SUPP_REL`.

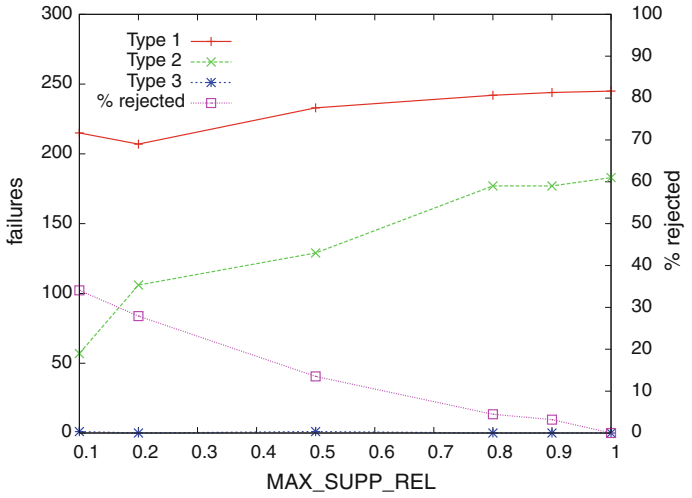

Fig. 6 Number of failures and cut rejection rate for cut generators with different values of MAX_SUPP_REL

The graphs show that limiting the maximum cut support has effect mostly on Type 2 failures, but little effect on Type 1 failures (there are too few Type 3 failures to detect any difference). A Friedman test to compare cut generators with MAX_SUPP_ABS $\geq 1{,}000$ does not reject the null hypothesis that they have the same number of failures of Type 1, with a p value of 0.6007. If we compare the number of dives that end with a failure of Type 2 instead, the null hypothesis is rejected with a p value of 0.0008. Similarly, for MAX_SUPP_REL ≥ 0.5 no difference in the number of failures of Type 1 is detected, but it is detected for Type 2 failures. If we limit the support even more, then the number of Type 1 failures decreases as well. This suggests that limiting the maximum cut support does not affect much the generation of invalid cuts unless we use a low threshold, however it can help in making the LPs easier to solve.

5.3 Cut modification parameters

We now turn our attention to the cut modification procedures by varying the corresponding parameters described in Sect. 2.1. As MIN_VIOL is set to 0, cuts can only be rejected if, after modification, they are no longer violated by the current LP solution.

5.3.1 Elimination of coefficients on surplus variables

The EPS_ELIM parameter takes value in $(0, \infty]$, but it is reasonable to assume that it should have a relatively small positive value. Indeed, eliminating large cut coefficients on surplus variables before their substitution in terms of original variables is likely to yield invalid cuts. We test the values EPS_ELIM $= 10^k$ for $k = -20, -18, \ldots, -2$ to get a sense of the impact of the parameter. The number of recorded failures and the cut rejection rate are reported in Fig. 7.

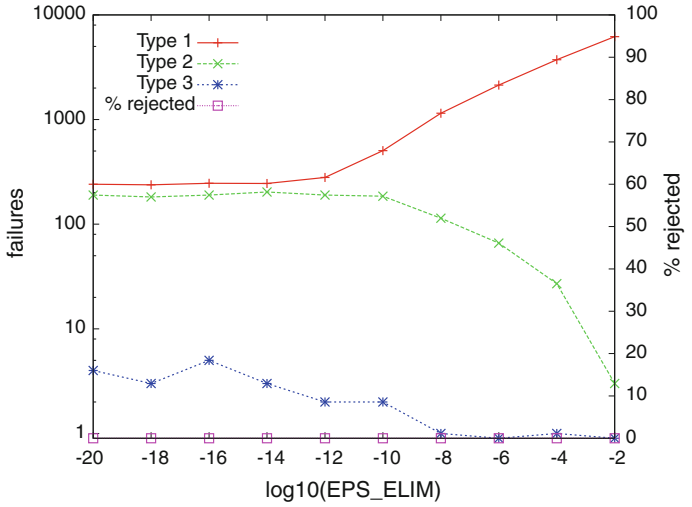

Fig. 7 Number of failures and cut rejection rate for cut generators with different values of `EPS_ELIM`. `EPS_ELIM` is set to 10^k, where k is the value on the x axis. The *left* y-axis has a logarithmic scale

In our experiments, the number of failures of Type 1 grows exponentially for `EPS_ELIM` $> 10^{-12}$. As expected, only small values for `EPS_ELIM` make sense in practice. For all values of `EPS_ELIM` $< 10^{-12}$, we observe very similar performance in terms of number of failures and rejection rate.

5.3.2 Elimination of small cut coefficients

`EPS_COEFF` takes value in $(0, \infty]$. When cut coefficients smaller than `EPS_COEFF` are set to zero, the right-hand side of the cut is adjusted accordingly to preserve validity. We tested the values `EPS_COEFF` $= 10^k$ for $k = -20, -18, \ldots, -2$.

We can see from Fig. 8 that `EPS_COEFF` seems to have an impact on the number of failures of all three types, especially Type 1 failures. The rejection rate increases quickly for `EPS_COEFF` $\geq 10^{-6}$ as the cut is no longer violated by the current LP solution.

5.3.3 Relaxation of the right-hand side value

Relaxation of the cut right-hand side value is controlled by two parameters: An absolute relaxation `EPS_RELAX_ABS` and a relative relaxation `EPS_RELAX_REL`. They both take value in $[0, \infty]$, but large values are likely to lead to an inequality not violated by the LP solution. For both parameters, we test the values 10^k for $k = -20, -19, \ldots, -1$.

Figure 9 plots the results for `EPS_RELAX_ABS` and Fig. 10 those for `EPS_RELAX_REL`.

For both parameters, values larger than 10^{-6} increase the rejection rate while decreasing the number of failures. This is not surprising as the cut relaxation is

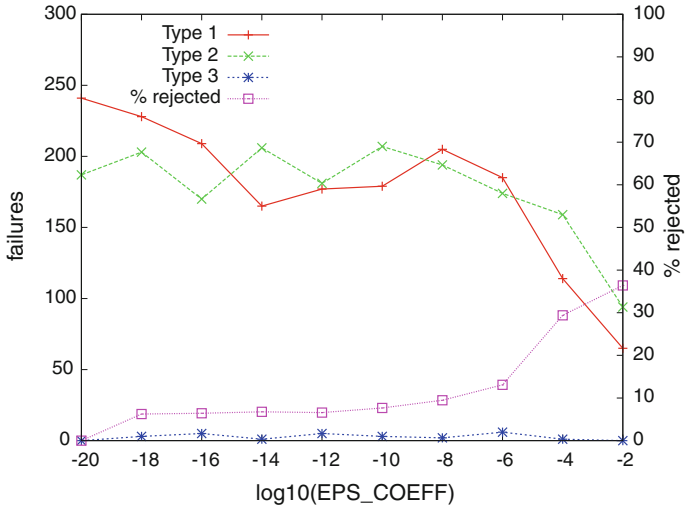

Fig. 8 Number of failures and cut rejection rate for cut generators with different values of `EPS_COEFF`. `EPS_COEFF` is set to 10^k, where k is the value on the x axis

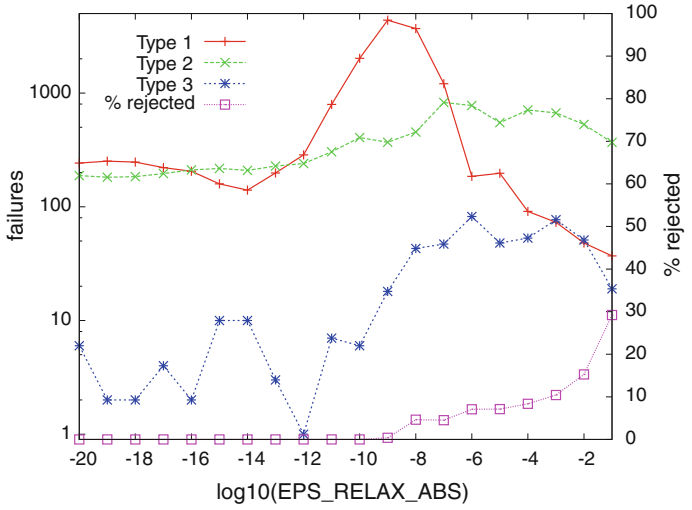

Fig. 9 Number of failures and cut rejection rate for cut generators with different values of `EPS_RELAX_ABS`. `EPS_RELAX_ABS` is set to 10^k, where k is the value on the x axis. The *left* y-axis has a logarithmic scale

significant. For values smaller than 10^{-12}, the cut rejection rate is small and the number of failures is fairly stable. However for values of the parameters in the range $[10^{-12}, 10^{-6}]$, the number of Type 1 failures increases significantly (up to a factor of 25 in the case of `EPS_RELAX_ABS`). We investigated this behavior and found that the amount by which the right-hand side of the cut is relaxed directly affects the

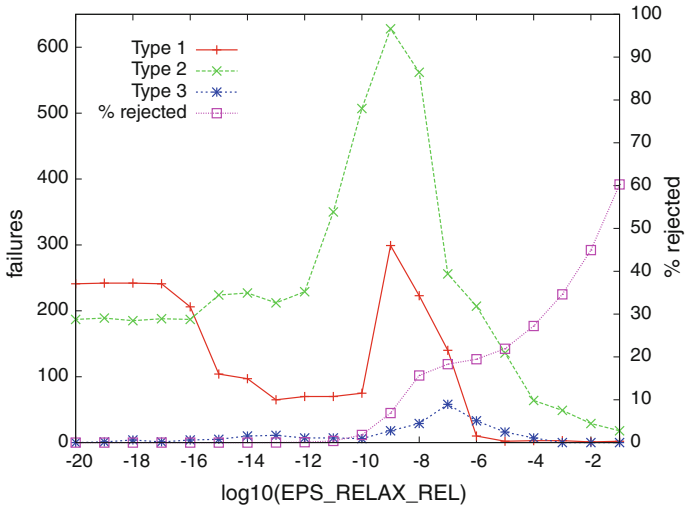

Fig. 10 Number of failures and cut rejection rate for cut generators with different values of EPS_RELAX_REL. EPS_RELAX_REL is set to 10^k, where k is the value on the x axis

fractionality of the basic integer variables at later rounds. We provide data to support this claim.

For each value of EPS_RELAX_ABS tested above and for each dive, we record the fractionality of the basic integer variables in all rounds, regardless of whether or not a GMI cut is derived from the corresponding row. For each instance in FAILURE SET, we compute over all dives and over all rounds the percentage p_k of basic integer variables whose fractionality falls in each of the ranges $[10^k, 10^{k+1})$, $k = -9, \ldots, -1$. Then, we compute the average $E[p_k]$ over all the instances, for all values of k. The heat map in Fig. 11 shows that $E[p_k]$ is maximum when 10^k is close to the value of EPS_RELAX_ABS. Experiments with different cut generators yield the same conclusions.

In light of these results, we can explain why relaxing the right-hand side by values in the range $[10^{-12}, 10^{-6}]$ yields an increase in the number of failures: the number of basic integer variables with small fractionality ($\leq 10^{-6}$) increases, which leads to potentially dangerous cuts since AWAY is set to 10^{-9} in this experiment, as shown in Sect. 5.2.1.

We also note that 10^{-9} is the primal feasibility tolerance. Therefore, cut relaxations of that order could increase the degeneracy of the LP bases and potentially lead to numerical troubles. However, we were unable to confirm whether or not primal degeneracy plays any role in the observed behavior of the cut generators.

6 Empirical testing: parameter optimization

To investigate the question of finding the optimal values for the cut generation parameters, we must first specify what we mean by "optimal".

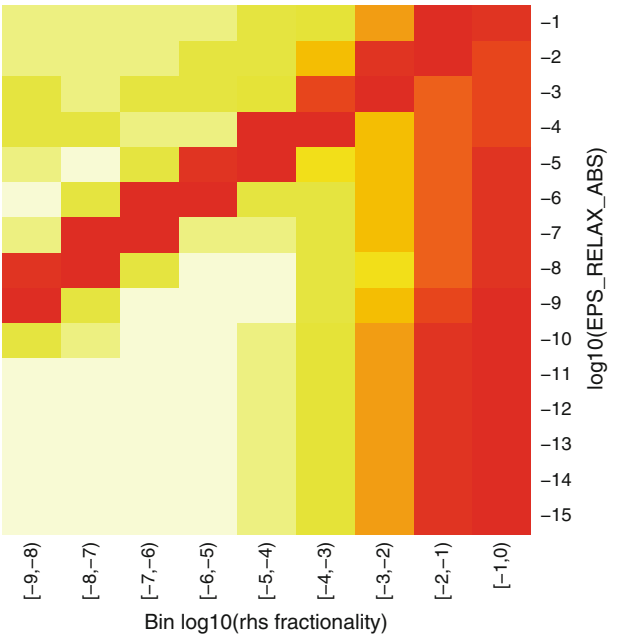

Fig. 11 Heat map of the matrix relating the fractionality of the basic integer variables and the value of EPS_RELAX_ABS. Darker colors correspond to larger values in the matrix. To enhance the picture, each column is rescaled to have its maximum equal to 1, i.e. the darkest color

In this paper, we are concerned with the safety of the cuts, as opposed to their strength. Our assumption is that cut generators should be compared in terms of strength only when they are comparable in terms of safety. We think of the cut modification and numerical check procedures as a filter. The cuts are modified and then checked, and they can be either accepted or rejected depending on whether or not they are judged safe. Strength does not play a role here. We would like the filter to be as loose as possible, while maintaining a given level of safety. In other words, we want to minimize the rejection rate of the cut generator, while achieving at least a given level of safety. Note that completely ignoring cut strength in this optimization step might be seen as dangerous, as we could end up with a generator with low rejection and failure rates but very weak. It turns out that this is not the case in our experiments. We verify in Sect. 7 that our best generators are comparable to standard generators in term of gap closed at the root.

6.1 Choice of the safety level

The optimization algorithm treats safety of the cut generator as a constraint. This implies that we should first define our measure of safety, and decide what is an acceptable level.

We measure safety of a cut generator on a set of instances by computing its *failure rate* with the DIVE-AND-CUT procedure. The failure rate is defined as the fraction of

dives that result in a failure of Type 1, 2 or 3. In our experiments we have 51 instances and we typically perform 300 dives on each for a total of 15,300 dives. Note that each dive may itself involve the generation of hundreds or even thousands of cuts. The validity of each cut is tested against a set of known feasible solutions.

To compute what is an acceptable failure rate over FAILURE SET, we test the commercial MILP solver IBM ILOG `Cplex` 12.2. We use `Cplex` as a black-box GMI cut generator within the DIVE-AND-CUT framework. The generator we call CPXGMI starts the MIP solution process on a copy of the original problem, disabling all presolving routines and cutting planes except GMI cuts, then reads generated cuts directly from the LP before branching at the root node. This disabling is necessary, as the cuts are intercepted while `Cplex` is solving the problem. Indeed, if presolve is used, the cuts are expressed using the variables of the presolved problem, while the feasible solutions used to check validity of the cuts are expressed using the variables of the original problem. As `Cplex` does not provide a correspondence between the two sets of variables, we would not be able to check validity of the cuts. We let `Cplex` generate GMI cuts with its default settings (number of candidate variables, number of rounds). As a result, we do not know precisely how many rounds are applied. Furthermore, we do not know if some cuts are generated but discarded for any reason. Since we cannot control the cut generation loop in `Cplex`, the only possibility is to read from the LP formulation the cuts that are still in the LP when `Cplex` decides to branch at the root.

We apply DIVE-AND-CUT on FAILURE SET in the single-machine setup with the cutting planes generated by `Cplex` as described above. Five failures of Type 1 were observed (one in each of `arki001`, `gt2`, `opt1217`, `p0033`, `p2756`) and none of Type 2 or 3, out of the 51×300 trials. Therefore the total failure rate is 0.03 %.

Even though we plan to generate significantly more cuts than CPXGMI, we want to achieve a similar or better level of safety.

6.2 The optimization algorithm

Optimizing the cut generation parameters is a black-box optimization problem. The objective function (cut rejection rate) and the constraints (failure rate) are unknown functions of the decision variables. Moreover, evaluating these unknown functions is computationally expensive, since this is done by running DIVE-AND-CUT on all instances of the test set.

Several methods for optimizing expensive black-box functions can be found in the literature. One possible approach is to use a response surface method (see e.g. [16,26]), where the constraint violations can be embedded into the objective function as penalty terms. Black-box optimization methods are typically tailored for continuous problems, avoiding the difficulty of dealing with discrete variables. More recently, some attempts at solving problems with integer variables have been made [17,18].

Instead of using an existing method from the literature, we decided to develop an ad hoc optimization algorithm for three reasons. First, we want to use a multidimensional objective function. That is, instead of considering the average cut rejection rate over all the instances and compare generators based on this single value, we consider the

cut rejection rate on each instance. Second, for assessing the safety of the generator, the failure rate must be below the threshold and, in addition, a Friedman test on the failure rate must show that the generator is comparable to or better than a reference generator. Third, we have the possibility of evaluating several points in the parameter space in parallel, using the Condor grid. Traditional response surface methods evaluate only one point at a time in a sequential fashion.

By using a statistical test for comparing points, we avoid the pitfall of aggregating results on several instances into a single measure of quality.

An important observation from the single-parameter experiments in Sect. 5 is that the cut rejection rate is monotone along each axis of the parameter space, and has a convex or almost-convex shape. Thus, an optimization algorithm that performs some kind of local search can reasonably be expected to find a good solution. Note however that since we use a vector-valued objective function and use a Friedman test on failure rates for comparing points, even convexity of the objective function would not guarantee convergence.

We discretize the set of possible values for each cut generation parameter. We use evenly spaced points in a reference interval, sometimes using a logarithmic scale for the parameter, using our best judgment for each parameter.

For the optimization algorithm, we assume that there are h parameters to optimize, and the ith parameter can take values in the ordered set $P_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,\text{len}(i)}\}$ where $\text{len}(i) \in \mathbb{N}$, and $p_{i,j} < p_{i,j+1}$ for $j = 1, \ldots, \text{len}(i) - 1$. A cut generator g is completely characterized by a point in $P_1 \times P_2 \times \cdots \times P_h$, and we denote by $g(i)$ the value of the ith parameter that defines g. For all $i, j \in \mathbb{Z}$, we define the function:

$$\text{midpoint}(i, j) = \begin{cases} i & \text{if} |i - j| \leq 1, \\ \lceil (i + j)/2 \rceil & \text{otherwise.} \end{cases}$$

For all $S_i \subseteq \{1, \ldots, \text{len}(i)\}$, we use the notation $P_i(S_i) = \{p_{i,j} : j \in S_i\}$. Algorithm 2 describes the main loop of the optimization algorithm. We label this algorithm OPTIMIZEPARAMETERS. The algorithm is a simple grid refinement algorithm. It repeatedly selects up to three values for each parameter, evaluates all generators with parameters on the grid defined by these values, selects the best generators (using the subroutine select_best(G) whose description is given in Algorithm 3), and computes the smallest box containing all generators in that set. OPTIMIZEPARAMETERS employs simple mechanisms to ensure that the search does not collapse too quickly towards a single point of the grid.

Given two cut generators $g, g' \in G$, we write $g <_R g'$ if a Friedman test on the cut rejection rate prefers g over g'. Similarly, we write $g <_F g'$ if a Friedman test on the failure rate yields that g is better than g'. Note that the $<_R$ and $<_F$ relations depend on the set of cut generators included in the statistical test. In Algorithm 3, $<_R$ or $<_F$ always refer to the test just performed.

For a set G of generators and a reference generator $\tilde{g}$, Algorithm 3 first selects in G' all generators in G that satisfy the upper bound on the failure rate and are not dominated by $\tilde{g}$ according to a Friedman test on the failure rate. In this paper, we use the CPXGMI cut generator discussed in Sect. 6.1 as the reference cut generator. It then

Algorithm 2 OPTIMIZEPARAMETERS

```
for i = 1, ..., h do
  ℓ_i ← 1
  u_i ← len(i)
  S_i ← {ℓ_i, midpoint(ℓ_i, u_i), u_i}
G' ← P_1(S_1) × P_2(S_2) × ··· × P_h(S_h)
repeat
  G ← G'
  Evaluate cut generators at grid points g ∈ G
  B ← select_best(G)
  for i = 1, ..., h do
    ℓ'_i ← arg min_j {p_{i,j} : (∃g ∈ B : g(i) = p_{i,j})}
    u'_i ← arg max_j {p_{i,j} : (∃g ∈ B : g(i) = p_{i,j})}
    if ℓ'_i = u'_i then
      center ← midpoint(ℓ_i, u_i)
      if ℓ'_i = ℓ_i then
        ℓ'_i ← 2ℓ_i − midpoint(ℓ_i, center)
        u'_i ← midpoint(ℓ_i, center)
      else if u'_i = u_i then
        ℓ'_i ← midpoint(u_i, center)
        u'_i ← 2u_i − midpoint(u_i, center)
      else
        ℓ'_i ← midpoint(ℓ_i, center)
        u'_i ← midpoint(u_i, center)
    ℓ_i ← max(1, ℓ'_i)
    u_i ← min(len(i), u'_i)
    S_i ← {ℓ_i, midpoint(ℓ_i, u_i), u_i}
  G' ← P_1(S_1) × P_2(S_2) × ··· × P_h(S_h)
until G = G'
```

applies a Friedman test on the cut rejection rate on generators in G' and selects in B all generators that are not rated as worse than any other generator in G'. As mentioned in Sect. 4.3, pairwise comparisons based on a Friedman test are not transitive, implying that it is possible to have three generators g_1, g_2 and g_3 with pairwise comparisons $g_1 <_F g_2$, $g_2 <_F g_3$, and $g_3 <_F g_1$. When this happens, none of the three generators are included in B, even if as a group they dominate all other generators in G'. To mitigate this unfortunate situation, the algorithm has a last loop that can increase the set B. That loop computes the set C of all generators in $G' \setminus B$ such that adding any one of the generators $c \in C$ results in c dominating a generator in B according to a Friedman test on $B \cup c$. One of the generators in C is then selected and added to B and this is repeated while the computed set C is nonempty. Note that more sophisticated selection of dominant subsets from inconsistent pairwise comparisons can be found in the literature [3,28]. For our purposes, the simple approach above seems to work well enough.

The selection of the generator in C requires a distance function, which we define next. Observe that in OPTIMIZEPARAMETERS a grid G has up to 3 possible values for each parameter. Given two generators $g, g' \in G$ and parameter i, the distance between g and g' along parameter i is 0 if $g(i) = g'(i)$, it is 2 if $|S_i| = 3$ and one of $g(i)$ or $g'(i)$ is the minimum value in S_i and the other is the maximum value, and it is 1 in all

other cases. The distance $d(g, g')$ is then defined as the sum over all parameters i of the distance between g and g' along parameter i.

Algorithm 3 select_best()

INPUT: Set of cut generators G, maximum failure rate γ, reference generator $\tilde{g}$
OUTPUT: Set of best cut generators B
Apply a Friedman test on $G \cup \{\tilde{g}\}$ on the failure rate
$G' \leftarrow \{g \in G : (\text{failure\_rate}(g) \leq \gamma) \wedge (g <_F \tilde{g})\}$
Apply on G' a Friedman test on the cut rejection rate
$B \leftarrow \{g \in G' : (\nexists g' \in G : g' <_R g)\}$
repeat
  $C \leftarrow \emptyset$
  for all $g \in G' \setminus B$ **do**
    Apply on $B \cup \{g\}$ a Friedman test on the cut rejection rate
    if $\exists g' \in B : g <_R g'$ **then**
      $C \leftarrow C \cup \{g\}$
  if $C \neq \emptyset$ **then**
    Select $c \in C$ such that $\min_{g \in B}\{d(c, g)\}$ is minimum; break ties selecting c such that $\text{conv}(B \cup \{c\})$ contains the largest number of elements in C; break further ties arbitrarily
    $B \leftarrow B \cup \{c\}$
until $C = \emptyset$

Observe that OPTIMIZEPARAMETERS terminates if we were not able to refine the grid during the previous iteration. Refining the grid depends on the detection power of the statistical test performed in the subroutine select_best() (Algorithm 3). When the grid cannot be refined, we could follow several strategies, such as increasing the number of dives or rounds to increase the detection power of the statistical tests, branching on the parameter space, or focusing only on one area of the parameter space. However, in our experiments, we were always able to refine the grid until it was sufficiently small, and therefore we did not need to resort to such strategies.

OPTIMIZEPARAMETERS is fully determined once the discretized values for each parameter are given. At the end of algorithm, we obtain a set of cut generators that yield lower cut rejection rates than the remaining generators tested during the course of the optimization, and such that a Friedman test on the rejection rate does not detect differences within the set.

6.3 Most influential parameters and initial grid

OPTIMIZEPARAMETERS evaluates 3^h points at each iteration. Since we have 12 parameters, this would require evaluating an overwhelming $3^{12} = 531{,}441$ generators at each iteration. Therefore, we first identify the most useful parameters, optimize over this smaller set and then find good values for the remaining ones.

To select the most useful parameters, we sample 500 points uniformly at random in the discretized parameter space. We run DIVE-AND-CUT on these 500 points in the Condor setup and fit a quadratic model for the total number of failures and for the rejection rate. We then use classical regression techniques to identify the most relevant parameters. Details are presented in the remainder of the section.

Table 3 Discretized parameter space. By convention, $10^{-\infty}$ means 0

AWAY	10^i	$i = -9, \ldots, -1$
EPS_COEFF	10^i	$i = -\infty, -20, \ldots, -1$
EPS_RELAX_ABS	10^i	$i = -\infty, -20, \ldots, -1$
EPS_RELAX_REL	10^i	$i = -\infty, -20, \ldots, -1$
MAX_DYN	10^i	$i = 6, \ldots, 30$
MIN_VIOL	10^i	$i = -\infty, -20, \ldots, -1$
MAX_SUPP_ABS	$250i$	$i = 1, \ldots, 16$
MAX_SUPP_REL	$i/10$	$i = 1, \ldots, 10$
EPS_ELIM	10^i	$i = -\infty, -20, \ldots, -12$
LUB	10^i	$i = 2, 3, 4, 50$
MAX_DYN_LUB	10^i	$i = 6, \ldots, 30$
EPS_COEFF_LUB	10^i	$i = -\infty, -20, \ldots, -1$

To choose the initial parameter ranges, we start with the ranges considered in Sect. 5, and reduce them based on the results of the experiments reported in that section. In particular, let P_i be the ordered set of values tested in Sect. 5 for the ith parameter. We set the lower (resp. upper) bound to the smallest (resp. largest) value such that the failure rate is at most 6 % and the rejection rate is at most 66 %.

The 6 % value for maximum failure rate allowed is chosen to exclude cut generators that yield more failures than CGBASE. The value 66 % for the maximum rejection rate allowed is chosen after testing a small number of "good" cut generators with typical parameter values. The smallest cut rejection rate recorded was 65.78 % and the corresponding failure rate was 0.04 %. Since we already know a cut generator with a rejection rate of 65.78 % and low failure rate, we exclude parameter ranges that are not likely to contain a better generator. Unfortunately this range reduction technique was not very effective: we could only reduce the range for MAX_DYN (lower bound increased to 10^6) and for EPS_ELIM (upper bound decreased to 10^{-12}). The resulting discretized parameter space P, from which the parameter values are randomly sampled is reported in Table 3. We sample 500 points $p_1, \ldots, p_{500}$ from P uniformly at random.

We evaluate the performance of cut generators parameterized with $p_1, \ldots, p_{500}$ on a subset of 25 instances of FAILURE SET, chosen randomly. Let $f : P \to \mathbb{R}$ and $r : P \to \mathbb{R}$ be the function returning respectively the failure rate and rejection rate.

We use the 12 parameters and their 66 first-order interaction terms and compute the best (smallest ℓ_2-norm of the vector of residuals) linear model fitting the points $(p_i, f(p_i))$ for $i = 1, 2, \ldots, 500$. We do this with the additional restriction that the linear model must use exactly s of the terms, for $s = 1, 2, \ldots, 12$. Results are reported in Table 4. For brevity, we use l() to indicate log, and only the initials of each parameter (e.g., MDL instead of MAX_DYN_LUB). Some parameters can assume the value 0; we substitute $\log(0) = -50$ for regression. Computations are performed with the open-source software R [25], using the packages biglm and leaps.

We repeat the same process for the cut rejection rate, using the points $(p_i, r(p_i))$ for $i = 1, 2, \ldots, 500$. Results are reported in Table 5.

Table 4 Independent variables defining the best subset of parameters for fitting a linear model to the failure rate function f

Size	l(A)	l(ERR)	l(MV)	l(A) l(ERA)	l(A) l(ERR)	l(A) l(MV)	l(EC) MSA	l(EC) l(MV)	l(EC) l(MDL)	l(ERA) l(MD)	l(ERR) l(MV)	MSA l(ECL)	l(MDL) l(ECL)	BIC
1	*													−55.99319
2					*	*								−75.88324
3		*			*						*			−87.27452
4		*	*		*						*			−89.88623
5		*	*		*	*					*			−98.61016
6		*	*	*	*	*					*			−99.13274
7		*	*	*	*	*				*	*			−96.73726
8		*	*		*	*	*	*			*	*		−100.72865
9		*	*		*	*	*		*		*	*	*	−105.34726
10		*	*	*	*	*	*		*		*	*	*	−105.75085
11		*	*	*	*	*	*		*	*	*	*	*	−105.89380
12		*	*	*	*	*	*	*	*	*	*	*	*	−105.57976

Column "size" specifies the size of the subset. Terms in a subset are identified with a "*". If a column label contains two parameters, it indicates an interaction term. The last column reports the Bayesian Information Criterion (BIC) value for each model

Table 5 Independent variables defining the best subset of parameters for fitting a linear model to the cut rejection rate function r

Size	l(EC)	l(ERR)	l(MDL)	l(A) l(ERA)	l(A) l(ERR)	l(A) l(MD)	l(A) l(MDL)	l(EC) l(MD)	l(ERA) l(ERR)	l(ERR) l(MD)	l(ERR) l(MV)	l(MD) l(L)	l(MV) l(MDL)	l(L) l(MDL)	l(L) l(ECL)	BIC
1										*						−136.9948
2						*				*						−227.1880
3	*					*				*						−254.1869
4	*			*				*		*						−280.0600
5	*					*		*	*		*					−309.9264
6	*					*		*	*	*			*			−317.0011
7	*					*		*	*	*			*	*		−317.0788
8	*					*		*	*	*		*	*	*		−319.7481
9	*					*	*	*	*	*		*	*	*		−318.6268
10	*	*			*		*	*	*	*	*	*		*		−318.9703
11	*	*	*		*	*		*	*	*	*	*		*		−317.5127
12	*	*	*		*	*		*	*	*	*	*		*	*	−315.7235

Column "size" specifies the size of the subset. Terms in a subset are identified with a "*". If a column label contains two parameters, it indicates an interaction term. The last column reports the Bayesian Information Criterion (BIC) value for each model

We consider that the optimization can be performed in reasonable time on up to six parameters. From Tables 4 and 5, we select the parameters `AWAY`, `EPS_COEFF`, `EPS_RELAX_ABS`, `EPS_RELAX_REL`, `MAX_DYN`, and `MIN_VIOL`. These six parameters (and their interaction terms) are sufficient to form the best subsets of parameters of size up to 7 for the model of the failure rate f, and up to 5 for the model of the cut rejection rate r. The Bayesian Information Criterion (BIC) values for the model using 7 terms for f and for the model using 5 terms for r are within 10 % of the minimum reported values, suggesting that these models are not over-fitting the data and predict the dependent variable well compared to the other subsets.

The grid over which the parameters are optimized is therefore the one reported in Table 3, limited to the six chosen parameters.

6.4 Results of the optimization algorithm

We ran OptimizeParameters as described in Sect. 6.2 for 5 iterations in the Condor setup. This required a massive amount of computing power on the Condor grid. Testing each cut generator requires typically between 20 and 40 hours of CPU time, and we tested thousands of cut generators.

Our target failure rate is $\gamma = 0.05\,\%$, close to `Cplex`'s 0.03 %. However, we start with $\gamma = 0.2\,\%$ at the first iteration, and lower this value by 0.05 % at each iteration until we reach the desired level. This prevents the failure rate constraint to eliminate a large portion of the parameter space in the first iterations, while the parameter grid is still very coarse. Later, the average and the standard deviation of the failure rate of the tested cut generator decrease and we can be more strict with the maximum failure rate constraint. In the end, the cut generators must be at least as safe as a reference generator (CpxGMI here) according to a Friedman test on the failure rate.

In Table 6 we provide a summary of the first 5 iterations of OptimizeParameters. We report the bounds of the parameter ranges at each iteration, the maximum allowed failure rate γ, the fraction of tested cut generators that satisfy the constraint on the failure rate, the average and standard deviation of the failure rate of the tested cut generators, and the average and standard deviation of the cut rejection rate.

We note that there is a trade-off between cut rejection rate and failure rate, hence minimizers of the rejection rate have a failure rate close to the allowed maximum. Since we lower the maximum failure rate γ in the first four iterations, the fraction of feasible cut generator drops. This can be seen for instance in Iteration 4, where only a few of the tested cut generators are feasible. Note that in the following iteration a larger fraction of cut generators is feasible, as γ is not changed. The average failure rate and cut rejection rate clearly show that OptimizeParameters is successful in identifying promising areas of the parameter space. By Iteration 5, both failure rate and rejection rate are very low and very stable across the tested generators (small standard deviation).

After Iteration 5 we are left with three cut generators. The parameters for one of them are reported in Table 7. The remaining two achieve the same performance, and they differ from BestGen only for the value of `MIN_VIOL`: this parameter is set to 10^{-16} and 10^{-11} instead of 0. For comparison, in Fig. 12 we report a histogram of

Table 6 Summary of the results of the OPTIMIZEPARAMETERS

	Iteration				
	1	2	3	4	5
AWAY	$[10^{-9}, 10^{-1}]$	$[10^{-5}, 10^{-1}]$	$[10^{-2}, 10^{-1}]$	$[10^{-2}, 10^{-2}]$	$[10^{-2}, 10^{-2}]$
EPS_COEFF	$[0, 10^{-1}]$	$[0, 10^{-11}]$	$[10^{-16}, 10^{-11}]$	$[10^{-12}, 10^{-10}]$	$[10^{-11}, 10^{-11}]$
EPS_RELAX_ABS	$[0, 10^{-1}]$	$[0, 10^{-1}]$	$[0, 10^{-11}]$	$[10^{-13}, 10^{-9}]$	$[10^{-12}, 10^{-10}]$
EPS_RELAX_REL	$[0, 10^{-1}]$	$[0, 10^{-1}]$	$[0, 10^{-11}]$	$[10^{-18}, 10^{-13}]$	$[10^{-14}, 10^{-12}]$
MAX_DYN	$[10^{6}, 10^{30}]$	$[10^{6}, 10^{30}]$	$[10^{6}, 10^{18}]$	$[10^{6}, 10^{9}]$	$[10^{6}, 10^{6}]$
MIN_VIOL	$[0, 10^{-1}]$	$[0, 10^{-11}]$	$[0, 10^{-11}]$	$[0, 10^{-11}]$	$[0, 10^{-11}]$
γ	0.20 %	0.15 %	0.10 %	0.05 %	0.05 %
% feasible	61.72 %	35.80 %	16.26 %	1.23 %	11.11 %
Avg fail rate	0.74 %	0.55 %	0.31 %	0.21 %	0.12 %
Std dev fail rate	1.65 %	0.66 %	0.19 %	0.13 %	0.05 %
Avg rej rate	71.72 %	57.12 %	47.96 %	40.03 %	41.88 %
Std dev rej rate	21.40 %	22.73 %	12.52 %	5.53 %	1.27 %

"γ" indicates the maximum failure rate allowed at each iteration. "% feasible" indicates the fraction of tested cut generators that satisfy the constraints (maximum failure rate and, from Iteration 4 on, at least as safe as reference generator). We then report, for each iteration, the average and standard deviation of the failure rate of the cut generators, and the average and standard deviation of the average cut rejection rate per instance

Table 7 Best cut generator BESTGEN returned by OPTIMIZEPARAMETERS, with failure rate 0.04 % and rejection rate 41.27 %

Cut generator BESTGEN					
AWAY	10^{-2}	EPS_RELAX_ABS	10^{-11}	MAX_DYN	10^{6}
EPS_COEFF	10^{-11}	EPS_RELAX_REL	10^{-13}	MIN_VIOL	0

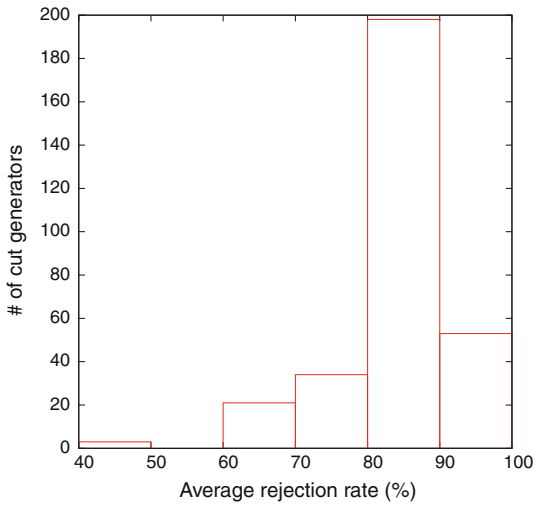

Fig. 12 Histogram of the average rejection rate for the cut generators that satisfy maximum failure rate of 0.05 %

the average cut rejection rate of all cut generators analyzed by OPTIMIZEPARAMETERS that satisfy the maximum failure rate constraint (0.05 %). This is a total of 309 cut generators. The cut generator in Table 7 is among the best 1 % generators encountered by OPTIMIZEPARAMETERS in terms of average cut rejection rate. It is interesting to note that no cut generator falls in the 50–60 % bin for the average rejection rate. In hindsight, we can explain this gap. OPTIMIZEPARAMETERS explores areas of the parameter space with small cut rejection rate but failure rate slightly above the allowed threshold 0.05 %, converging towards the only remaining feasible cut generators in the area that was identified as having the lowest rejection rate.

6.5 Parameter sensitivity

We now proceed to analyze the sensitivity of the failure and cut rejection rates with respect to the cut generation parameters in the neighborhood of one of the best generators found in the previous section. Our reference cut generator is BESTGEN from Table 7. Note that since the three generators in Table 7 are very similar and differ in one parameter value only, it seems likely that the results in this section are valid for the other two generators also. Details of our methodology and results are given in Appendix B.

Results in Appendix B suggest that for some of the parameters (e.g. `AWAY`, `EPS_RELAX_ABS`, `EPS_RELAX_REL`, `MAX_DYN`) even small changes have a visible effect. For other parameters, there is more freedom in choosing the parameter value. In our experiments, the parameters controlling the maximum support of the cutting planes have almost no effect on the number of failures. This is probably due to the data set and the large time limit before a Type 3 failure is reported (5 min). In practice, it may be desirable to set some limit for the cut support to speed up LP resolves, but in this paper we focus on safety and we did not find evidence to support the claim that dense cuts are less safe than sparse cuts, provided that the most important cut generation parameters are well chosen. Setting to zero small coefficients on surplus variables does not show any positive effect in our experiments: any nonzero value for `EPS_ELIM` yields a small (but statistically significant) increase in the number of failures, and for larger values many invalid cuts are generated (see Sect. 5.3.1). Using a positive value for `EPS_ELIM` may yield some CPU time savings, but in terms of safety it does not seem advantageous. The experiments with `EPS_RELAX_ABS`, `EPS_RELAX_REL` showed the behavior already observed in Sect. 5.3.3: a value of the parameter approximately in the range $[10^{-9}, 10^{-6}]$ yields an increase in the number of failures, although here the increase is not as large as in Sect. 5.3.3 because the remaining cut generation parameters mitigate the effect. It does not seem a good idea to choose `EPS_RELAX_ABS` or `EPS_RELAX_REL` close to 10^{-9}. The ranges for `MAX_DYN_LUB` are very similar to those of `MAX_DYN`, and the differences could be explained by the fact that `MAX_DYN_LUB` is less influential than `MAX_DYN` as it acts on fewer variables. In light of these results, there is not much evidence to support using for `MAX_DYN_LUB` a value different than the one used for `MAX_DYN`. In our experiments, using any value for `EPS_COEFF_LUB` other than the starting value 10^{-13} yielded a small but statistically

significant increase in the number of failures. We do not have an explanation for this behavior.

We also analyzed sensitivity with respect to the number of rounds ρ of cut generation. Performing a local reoptimization for $\rho = 15$, we found that the parameters did not change significantly. In particular the optimal values $\texttt{AWAY} = 10^{-2}$ and $\texttt{MAX\_DYN} = 10^{6}$ stayed the same.

6.6 Scaling

For a set of four cut generators (BESTGEN, ITER1, ITER2, ITER3; see Sect. 7 for their descriptions), we tested six variants of cut scaling in the single-machine setup. We chose these four generators because they are safe and belong to different regions of the parameter space. The six scaling variants are based on the following three commonly used procedures:

(i) Scale the largest cut coefficient to 1;
(ii) Scale the cut right-hand side to 1;
(iii) Scale all the coefficients of integral variables to integer.

For each procedure, we either enforce the scaling (cuts which cannot be scaled properly because the scaling factor is too large or too small are discarded), or do not enforce the scaling (if scaling fails, we keep the original cut). Hence the six combinations. We applied SCALING before the remaining cut modification procedures.

The results are the following. For generators ITER1 and ITER2 there is no difference in safety among the scaling variants detected by the usual significance tests. This is expected because ITER1 and ITER2 are very safe and conservative generators. For generators ITER3 and BESTGEN, using (ii) decreases safety noticeably and is detected by a Friedman test at the 95 % level. All other scaling variants to do not yield significant differences. There are some differences in the failure rates in many cases, but not enough to be significant.

Using (iii) yields very safe generators: we recorded only 1 failure in this entire set of experiments. However the rejection rate goes up to around 90 %, far from the rejection rate of the other generators. Moreover, as all four generators have low failure rates, a Friedman test does not detect a significant improvement when using (iii).

Summarizing, we did not find evidence that SCALING is beneficial, provided the remaining cut parameters are properly chosen. This is not surprising in light of the fact that we use a relative feasibility tolerance, and that LP solvers typically rescale cuts before using them.

7 Validation of the results

The goal of this section is to show that the conclusions drawn in the preceding sections have useful practical implications. All experiments in this section were executed in the single-machine setup, which is a different architecture (x86_64) than the Condor grid (generic i386). Therefore, we can verify if our conclusions carry over to different architectures.

There are several points that we want to investigate. First, we want to check that the optimal generators obtained at the end of OPTIMIZEPARAMETERS are safe and reject fewer cuts than other generators. We would also like to confirm that a small cut rejection rate translates into a cutting plane generator producing stronger cuts as a whole. A thorough analysis of the strength of cut generators is beyond the scope of this paper. However, here, we would like to investigate whether, when two cut generators have a comparable level of safety, the generator rejecting fewer cuts is stronger. We use the percent of integrality gap closed as a measure of strength. This not a very accurate measure of strength, but is a widely accepted approximation. Note that the results on the strength of the generators presented in this section are not meant to conclude that one of the tested generators should be used in practice "as is". The tests done here ignore many factors impacting the practical efficiency of cut generators when used in a Branch-and-Cut algorithm, such as the impact of cut support on LP resolve times.

We compare 11 cut generators over 300 dives of DIVE-AND-CUT on FAILURE SET. We use a different random seed than in previous experiments. Thus, we are not testing on the same instances that were used for OPTIMIZEPARAMETERS, i.e. the integer variables are fixed in a different way. The cut generators that we test are the following.

- BESTGEN: generator #1 from Table 7.
- BESTGENAWAY: generator #1 from Table 7 with `AWAY` set to $5 \cdot 10^{-3}$ as may be suggested by the parameter sensitivity reported in Table 14.
- ITER1: the cut generator with lowest failure rate in the set of best cut generators explored at Iteration 1 of OPTIMIZEPARAMETERS.
- ITER2: the cut generator with lowest failure rate in the set of best cut generators explored at Iteration 2 of OPTIMIZEPARAMETERS.
- ITER3: the cut generator with lowest failure rate in the set of best cut generators explored at Iteration 3 of OPTIMIZEPARAMETERS.
- CGLGOMORY: the Gomory cut generator from `Cgl`.
- CGLGOMORYMOD: our GMI cut generator parameterized in a similar way to CGLGOMORY.
- CGLLANDP: the Lift&Project cut generator from `Cgl`, parameterized with `pivotLimit = 0` so that it generates only GMI cuts.
- CGLLANDPMOD: our GMI cut generator parameterized in a similar way to CGLLANDP.
- CPXGMI: `Cplex`'s GMI cut generator with default parameters (i.e. `Cplex` decides the number of cuts and the number of rounds of cutting planes).
- CPX: `Cplex`'s cut generators with default parameters (i.e. `Cplex` decides which cutting plane families should be applied, the number of cuts and the number of rounds).

We note that the implementation of CGLGOMORY uses many more tolerances than our GMI cut generator, therefore CGLGOMORYMOD will yield different results. On the other hand, our implementation of CGLLANDPMOD is very similar to CGLLANDP, but a few important differences remain. In particular, CGLLANDP is tied to COIN-OR `Clp` [9] as the LP solver, whereas we use `Cplex`. Furthermore, CGLLANDP generates

at most 50 cuts per round and uses the optimal simplex tableau returned by the LP solver, while our GMI cut generator has no limit on the number of generated cuts and internally recomputes the optimal simplex tableau from scratch. For these reasons, comparisons with CglLandP are difficult to interpret and should be taken with a grain of salt.

We consider CglGomoryMod and CglLandPMod as "reasonable" parameterizations of the GMI cut generator that are interesting to compare to our reference BestGen. Comparing BestGen with Iter1, Iter2 and Iter3 allows us to verify that Algorithm 2 made progress and found better cut generators in later iterations. Note that Iter3 differs from BestGen only in the value of `EPS_RELAX_REL` (10^{-16} instead of 10^{-13}), hence they should have very similar results. We do not report results with Iter4 as it is identical to BestGen. BestGenAway is parameterized similarly to BestGen, but uses a smaller value of the `AWAY` parameter. This should yield a smaller cut rejection rate with a comparable safety level, according to the experiments in Appendix B. While this paper focuses on GMI cuts, in this section we include Cpx in the comparison to provide another reference point and to show that GMI cuts are not the only cuts that can be unsafe. We remark that in Dive-and-Cut, standard features of the LP solver such as presolve are turned on with default values. This does not conflict with the cut generators Cpx and CpxGMI, as all cut generators receive as input a copy of the original problem and the optimal basis. Note that in our framework the problem passed to the solver of choice is a pure LP, therefore we do not use MIP presolve.

We first compare the safety of the cut generators. We apply a Friedman test on the failure rate. The null hypothesis that all generators have the same failure rate is rejected with a p value of 0.0000. We perform post-hoc analysis to identify which generators are safer by determining, for each pair of cut generators, if the difference in the failure rates is significant. Results are reported in Table 8.

The results show that the failure rate has increased slightly with respect to the maximum allowed failure rate during the optimization run. This is explained by the fact that we are now running dives on different instances and on a different machine, therefore it is not surprising that the computations give different results. However, according to the statistical tests reported in Table 8, the safety of BestGen, BestGenAway, Iter1, Iter2, and Iter3 is still not significantly different from that of CpxGMI, which is our target. Our five cut generators are safer than CglGomory, CglLandPMod and Cpx. Notice that CglLandP has a much larger failure rate than CpxGMI, but no difference is detected by the Friedman test, for two reasons. First, the test does not take into account the magnitude of the differences. Second, the large average is mostly due to a single instance with very large failure rate (`arki001` with 70.33 % failure rate). In Fig. 13 we report the failure rate per instance. Figure 13 shows that among the cut generators deemed as safe as CpxGMI by a Friedman test, CglLandP is the least consistent, exhibiting the largest median and a large failure rate on a few instances. CpxGMI, BestGen and BestGenAway are the most consistent, with very low failure rates on every instance.

As we cannot compute the cut rejection rate for cut generators that are not based on our implementation (i.e. `Cplex` and the two `Cgl` generators), we use the total number of cuts instead of the cut rejection rate. This is of course different than com-

Table 8 Comparison of the failure rate per instance

	Fail. rate	BestGen	BestGenAway	Iter1	Iter2	Iter3	CglGomory	CglGomoryMod	CglLandP	CglLandPMod	CpxGMI	Cpx
BestGen	0.065		=	=	=	=	−	=	=	−	=	−
BestGenAway	0.072	=		=	=	=	−	=	=	−	=	−
Iter1	0.084	=	=		=	=	−	=	=	−	=	−
Iter2	0.098	=	=	=		=	−	=	=	−	=	−
Iter3	0.065	=	=	=	=		−	=	=	−	=	−
CglGomory	3.555	+	+	+	+	+		+	+	+	+	=
CglGomoryMod	0.229	=	=	=	=	=	−		=	=	=	−
CglLandP	1.680	=	=	=	=	=	−	=		−	=	−
CglLandPMod	1.379	+	+	+	+	+	−	=	+		+	−
CpxGMI	0.039	=	=	=	=	=	−	=	=	−		−
Cpx	1.634	+	+	+	+	+	=	+	+	+	+	

Column "Fail. rate" gives the average failure rate (%). A + (resp. −) in row i and column j means that the failure rate of the cut generator in row i is significantly larger (smaller resp.) than the failure rate of the cut generator in column j. The significance level is 95 %

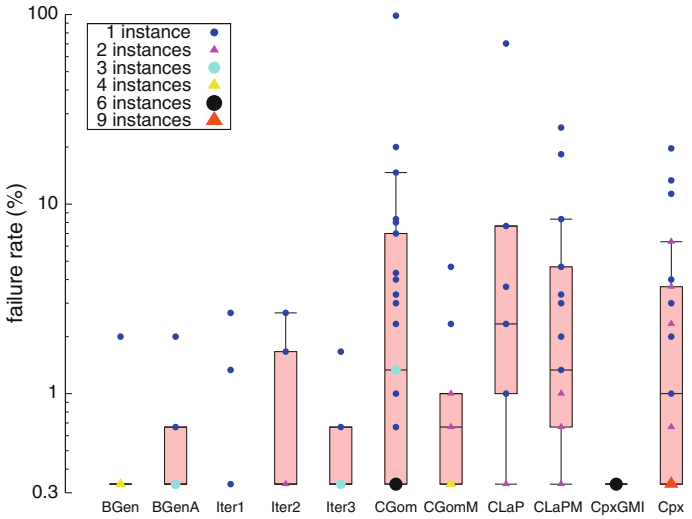

Fig. 13 Failure rate on all instances with nonzero failure rate. When two or more instances have the same failure rate for a cut generator, we group them together and increase the size of the point as indicated in the legend. For each cut generator, we additionally plot a box-and-whisker graph of the failure rate: The box extends from the 25 to the 75 % quantile, the middle bar indicates the median. The whiskers extend from the end of the box to the most distant point whose value lies within 1.5 times the difference between the 75 and the 25 % quantile

paring the cut rejection rate, as a cut generator may have a larger cut rejection rate while generating more cuts in the 30 rounds of cut generation. The comparisons are based on a Friedman test, where the performance measure is the total number of generated cuts. Results are reported in Table 9. We see that, according to the Friedman test, BestGenAway generates significantly more cuts than all remaining cut generators except CglLandP and CglLandPMod. In turn BestGen generates significantly more cuts than CpxGMI and Cpx. In fact BestGenAway generates roughly 50 times as many cuts as CpxGMI while still yielding a comparable number of failures. However, we have no way of computing the actual rejection rate of `Cplex`, as explained in Sect. 6.1. Table 10 compares the cut rejection rate for the six cut generators where this number can be computed, using a Friedman test. It is interesting to note that BestGenAway has a significantly smaller rejection rate than BestGen.

Finally, we compare the integrality gap closed by the cut generators. We use a Friedman test where the performance measure is the gap closed after applying cutting planes after each dive. The integrality gap for each dive is computed using the objective value of the best solution against which validity of the cuts is tested. This is an upper bound on the optimum value of the instance obtained after fixing random variables in Dive-and-Cut. The null hypothesis that all cut generators close the same amount of gap is rejected with a p value of 0.0000. Pairwise comparisons are reported in Table 11. Note that with this test we do not have any quantitative information in the improvement on the gap closed: The Friedman test only tells us whether some

Table 9 Comparison of the total number of cuts

	Gen. cuts	BestGen	BestGenAway	Iter1	Iter2	Iter3	CglGomory	CglGomoryMod	CglLandP	CglLandPMod	CpxGMI	Cpx
BestGen	1,222.47		−	+	+	=	=	+	−	−	+	+
BestGenAway	1,246.43	+		+	+	+	+	+	=	−	+	+
Iter1	991.82	−	−		=	−	−	−	−	−	+	+
Iter2	992.28	−	−	=		−	−	−	−	−	+	+
Iter3	1,227.72	=	−	+	+		=	+	=	−	+	+
CglGomory	822.48	=	−	+	+	=		+	=	−	+	+
CglGomoryMod	1,106.29	−	−	+	+	−	−		−	−	+	+
CglLandP	1,034.41	+	=	+	+	=	=	+		−	+	+
CglLandPMod	1,599.87	+	+	+	+	+	+	+	+		+	+
CpxGMI	22.53	−	−	−	−	−	−	−	−	−		−
Cpx	126.95	−	−	−	−	−	−	−	−	−	+	

Column "Gen. cuts" gives the average number of generated cuts per dive

Table 10 Comparison of the rejection rate

	Rej. rate	BestGen	BestGenAway	Iter1	Iter2	Iter3	CglGomoryMod	CglLandPMod
BestGen	37.21		+	−	−	=	−	=
BestGenAway	35.11	−		−	−	−	−	−
Iter1	57.40	+	+		=	+	=	+
Iter2	57.37	+	+	=		+	−	+
Iter3	37.26	=	+	−	−		−	=
CglGomoryMod	63.30	+	+	=	+	+		+
CglLandPMod	36.31	=	+	−	−	=	−	

Column "Rej. rate" gives the average rejection rate (%)

algorithms consistently rank better than others. The performance difference could be negligible and could depend on other factors than the strength of the cuts. To eliminate this undesired effect, we perform a related experiment where we require a minimum difference of 1 % integrality gap for a cut generator to be considered better than another. A Friedman test for this experiment yields exactly the same pairwise comparisons as in Table 11.

We observe that BestGenAway closes a similar amount of gap as BestGen and Iter3, and all three close more gap than all other generators except CglGomory, CglLandP and CglLandPMod. Recall, however, that the failure rate of CglGomory and CglLandPMod is significantly larger than those of BestGen, BestGenAway and Iter3. BestGenAway is comparable to CglLandPMod in terms of gap closed, thus it could be argued that it is stronger than BestGen and Iter3 (recall that the pairwise comparisons are not transitive).

Our final experiment consists in testing the cut generators BestGen, BestGenAway, CglLandP, CpxGMI, and Cpx on the set of instances labeled Extended Set, see Appendix A.2. These 69 instances were not used for OptimizeParameters as they yield fewer failures than instances in Failure Set. We perform 300 dives per instance. The number of recorded failures is reported in Table 12.

BestGen, BestGenAway and CglLandP do not generate Type 1 failures, performing better than CpxGMI and Cpx in this regard. However, they generate more Type 2 and Type 3 failures. We remark that 29 of the 33 failures of Type 2 for BestGen and 19 of the 24 failures of Type 2 for BestGenAway are recorded on the same instance (`rococoC10-001000`). CglLandP is safe on most instances but shows again poor performance on a few problems. 41 out of the 49 Type 2 failures occur on three instances only (`lectsched-4-obj`, `momentum1`, `momentum2`). On four instances (`30n20b8`, `momentum2`, `net12`, `rocII-4-11`), the 300 dives are not carried out to completion because of the early stopping criterion (more than 30 Type 3 failures). It is clear that our BestGen generators are very effective in rejecting potentially invalid cuts, whereas CpxGMI and Cpx are more effective in rejecting cuts that may slow down or create troubles in the LP solution process. This is not surprising as these aspects are not the primary focus of our investigation, while this is obviously an important practical consideration.

Table 11 Comparison of the gap closed

	BestGen	BestGenAway	Iter1	Iter2	Iter3	CglGomory	CglGomoryMod	CglLandP	CglLandPMod	CpxGMI	Cpx
BestGen		=	+	+	=	=	+	−	−	+	+
BestGenAway	=		+	+	=	=	+	−	=	+	+
Iter1	−	−		=	−	−	=	−	−	+	=
Iter2	−	−	=		−	−	=	−	−	+	=
Iter3	=	=	+	+		=	+	−	−	+	+
CglGomory	=	=	+	+	=		+	−	−	+	+
CglGomoryMod	−	−	=	=	−	−		−	−	+	=
CglLandP	+	+	+	+	+	+	+		=	+	+
CglLandPMod	+	=	+	+	+	+	+	=		+	+
CpxGMI	−	−	−	−	−	−	−	−	−		−
Cpx	−	−	=	=	−	−	=	−	−	+	

Table 12 Number of failures recorded on EXTENDED SET and number of distinct instances with at least one failure

	Failures				Distinct instances			
	T. 1	T. 2	T. 3	Tot.	T. 1	T. 2	T. 3	Tot.
BESTGEN	0	33	17	50	0	4	5	9
BESTGENAWAY	0	24	24	48	0	5	5	10
CGLLANDP	0	49	170	219	0	10	7	17
CPXGMI	35	2	0	37	6	2	0	8
CPX	400	2	0	402	21	2	0	23

8 Conclusions

Practitioners are well aware that MILP solvers sometimes report incorrect optimal solutions. This is due to the fact that computations are performed in finite precision and numerical difficulties do occur. A major source of numerical problems arises from the cutting planes used in Branch-and-Cut solvers, even though these solvers take various steps to avoid the generation of invalid cuts. This paper studies the effectiveness of these safety-enhancing steps on GMI cuts. We show that, among the dozen or so parameters that are widely used for safety, only five seem to really matter. The two most important are `AWAY` and `MAX_DYN`:

- A cut should not be generated if the value of the corresponding integer basic variable is within `AWAY` of an integer value; we found that the best setting for `AWAY` is somewhere between 0.005 and 0.01.
- A cut is discarded if the ratio between the largest and smallest absolute values of the nonzero coefficients is larger than `MAX_DYN`. Our optimization algorithm found that setting `MAX_DYN` to 10^6 works well.

In addition, cut coefficients that are smaller than some small value `EPS_COEFF` are set to zero, adjusting the right-hand side accordingly, and the cut right-hand side itself is relaxed using an absolute amount `RELAX_RHS_ABS` and a relative amount `EPS_RELAX_REL`. Our optimization algorithm recommends `EPS_COEFF` = `RELAX_RHS_ABS` = 10^{-11} and `EPS_RELAX_REL` = 10^{-13}.

Using these parameters, less than 40 % of the GMI cuts generated in 30 rounds were rejected as being potentially unsafe. We found that, compared to generators of existing solvers, our generators generate many more cuts while being at least as safe.

Selecting the best cuts among those that are deemed to be safe, with the goal of improving the performance of Branch-and-Cut solvers, is an important topic for future research.

Acknowledgments We thank the referees for their constructive comments. We are extremely grateful to Jeff Linderoth and the entire Condor team at UW-Madison for giving us the opportunity of running experiments on the grid. G. Cornuéjols was supported in part by NSF Grant CMMI 1263239 and ONR Grant N00014-12-10032. F. Margot was supported by ONR Grant N00014-12-10032. G. Nannicini was supported in part by SUTD Grant SRES11012 and IDC Grants IDSF1200108, IDG21300102.

Appendix A: DIVE-AND-CUT Solution-Generation phase

In this section, we describe the algorithm GENERATESOLUTIONS (see Algorithm 4) used to generate a set S of $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, 0)$-feasible solutions for a problem instance P. GENERATESOLUTIONS applies a Branch-and-Cut solver $\mathcal{B}$ to the instance at hand P and acts whenever the solver discovers an integer solution. In `Cplex`, this can be achieved by a call to the function `incumbentcallback()`.

Whenever a candidate solution $\tilde{x}$ is discovered, GENERATESOLUTIONS rounds each integer variable to the nearest integer, and checks the feasibility of this rounded solution. If it fails to satisfy the constraints within an absolute feasibility tolerance ϵ_{abs} and a relative feasibility tolerance ϵ_{rel}, it is rejected.

If $\tilde{x}$ is not rejected, we use a rational LP solver to generate provably feasible solutions to the problem P that have value $\tilde{x}_i$ for all integer variables x_i. First, we seek a provably feasible solution x^* that minimizes the original objective function. If we are able to compute x^*, it is added to the set S of feasible solutions. If its Euclidean distance to $\tilde{x}$ is less than ϵ_{rel}, we are done and $\tilde{x}$ is accepted. Otherwise, we seek the point x' in the feasible region of the LP that is closest to $\tilde{x}$ in ℓ_1-norm. If $\|\tilde{x} - x'\|_2 \leq$ ZERO, these two points are essentially the same and x' is added to S. If $\|\tilde{x} - x'\|_2 \leq \epsilon_{\text{rel}}$, $\tilde{x}$ is "feasible enough" according to our criteria and is added to S. In both cases, $\tilde{x}$ is accepted. Otherwise, it is rejected.

Algorithm 4 GENERATESOLUTIONS. The solution of an LP is the symbol NIL if the LP is infeasible.

INPUT: problem $P = (A, b, c)$, Branch-and-Cut solver $\mathcal{B}$, rational LP solver $\mathcal{R}$, tolerances $\epsilon_{\text{abs}}, \epsilon_{\text{rel}}$
OUTPUT: set S of $(\epsilon_{\text{abs}}, \epsilon_{\text{rel}}, 0)$-feasible solutions
Apply $\mathcal{B}$ to P
for (every candidate solution $\tilde{x}$ discovered) **do**
  Set feasible $\leftarrow$ `false`
  Set $\tilde{x}_i \leftarrow \lfloor \tilde{x}_i \rceil \ \forall i \in N_I$
  if $(\max_{i\in[m]}\{b_i - a^i\tilde{x}\} \leq \epsilon_{\text{abs}})$ **and** $(\max_{i\in[m]}\{(b_i - a^i\tilde{x})/\|a^i\|_2\} \leq \epsilon_{\text{rel}})$ **then**
    Compute $x^* = \arg\min\{c^\top x \mid Ax \geq b, x \geq 0, x_i = \tilde{x}_i \ \forall i \in N_I\}$ with $\mathcal{R}$
    if $(x^* \neq \text{NIL})$ **then**
      Set $S \leftarrow S \cup \{x^*\}$
      if $(\|x^* - \tilde{x}\|_2 \leq \epsilon_{\text{rel}})$ **then**
        Set feasible $\leftarrow$ `true`
      else
        Compute $x' = \arg\min\{\|\tilde{x} - x\|_1 \mid Ax \geq b, x \geq 0, x_i = \tilde{x}_i \ \forall i \in N_I\}$ with $\mathcal{R}$
        if $(\|\tilde{x} - x'\|_2 \leq$ ZERO$)$ **then**
          Set $S \leftarrow S \cup \{x'\}$
          Set feasible $\leftarrow$ `true`
        else if $(\|\tilde{x} - x'\|_2 \leq \epsilon_{\text{rel}})$ **then**
          Set $S \leftarrow S \cup \{\tilde{x}\}$
          Set feasible $\leftarrow$ `true`
  if (feasible $=$ `true`) **then**
    Report to $\mathcal{B}$ that $\tilde{x}$ is accepted
  else
    Report to $\mathcal{B}$ that $\tilde{x}$ is rejected

Note that GENERATESOLUTIONS uses the ℓ_1-norm for the problem of finding x' close to $\tilde{x}$ since this can be formulated as an LP using the usual reformulation: The objective

function $\min_{x\in\mathbb{R}^n} \|\tilde{x} - x\|_1$ can be expressed in linear form by adding n extra variables and $2n$ constraints: $\min_{w,x\in\mathbb{R}^n}\{\mathbf{1}_n w \mid \tilde{x} - x \leq w, x - \tilde{x} \leq w\}$. Since the ℓ_1-norm overestimates the ℓ_2-norm this guarantees that x' is also at most ϵ_{rel} away from $\tilde{x}$ in ℓ_2-norm. As a consequence, if such an x' exists, the relative violation of any nonnegative linear combinations of rows of $Ax \geq b$ by $\tilde{x}$ is at most ϵ_{rel}.

GENERATESOLUTIONS accomplishes several desirable goals. First, it does not modify the original instances. Second, given enough time, it finds an optimal solution (feasible according to our criteria) and adds it to the set S. Third, it potentially returns a large and diverse set of feasible solutions, with different values for the integer variables. This is valuable for our purposes, as this allows testing the validity of cutting planes with respect to solutions in different parts of the feasible region.

However, GENERATESOLUTIONS has also some drawbacks. First, the generated solutions are the nearest floating point representation (component-wise) of the feasible solutions computed in rational numbers. Therefore, they could be infeasible when the left-hand side of the constraints is computed in finite precision. Second, the generated solutions might be cut off in terms of absolute violation by linear combinations of the problem constraints. Using only a relative violation tolerance would address the second issue, as explained above. However, using also an absolute feasibility tolerance makes sense, because all available Branch-and-Cut codes have an absolute feasibility tolerance $\epsilon_{\text{abs}} > 0$.

A.1: Implementation

GENERATESOLUTIONS's implementation is tied to `Cplex` 12.2 because it uses advanced Branch-and-Cut functions. In particular, we use the `incumbentcallback()` function to intercept the discovery of feasible solutions and run our implementation of Algorithm 4. The feasibility and integrality tolerances for `Cplex` were set to 10^{-9}, the smallest feasibility tolerance allowed by `Cplex`.[2] The rational LP solver used by GENERATESOLUTIONS is `QSopt_ex` [4].

We parameterize the Branch-and-Cut algorithm of `Cplex` as follows. The integrality gap allowed for optimality is set to 0 (both relative and absolute). The time limit is set to 6 h, the number of parallel threads to 2, and `SolutionPoolIntensity` is set to 2. We disable rescaling of the LP matrix, in order to avoid discovery of solutions that are infeasible for the unscaled problem (`Cplex` error code: `OptimalInfeas`). If the instance is solved to optimality before the 6 hours time limit, we call the `CPXpopulate()` procedure to generate more solutions until the time limit is hit or an additional 100 solutions are found. The strategy for replacing solutions in the pool when it is full is set to `Diversify`.[3]

The experiments in this section are run on a machine equipped with an `AMD Opteron 4176 HE` processor clocked at 2.4GHz and 48GB RAM, running Linux.

[2] The smallest integrality tolerance is 0.0, but the user manual warns that such a small tolerance may not always be attainable.

[3] This setting should have no effect on the outcome since we intercept each solution discovered and store it in a separate pool; we mention it only for completeness.

A.2: Computational experiments on MIPLIB instances

Our initial test set contains all instances from MIPLIB3 [7], MIPLIB2003 [2], and the Benchmark set of MIPLIB2010 [21] beta (downloaded March 2011) for a total of 169 instances. Ten instances are eliminated, as GENERATESOLUTIONS does not find any feasible solution for them. We are left with the 159 instances listed in Table 13. On the `satellites1-25` instance, more than 10^6 solutions are found during the initial Branch-and-Cut; we keep the best solution found and an additional 1,499 solutions selected at random.

The vast majority of the solutions found by GENERATESOLUTIONS are feasible with very small tolerances. In Fig. 14 we report a histogram of the maximum absolute violation for the generated solutions, computed as $\max_{i\in[m]}\{b_i - a^i x^*\}$. The maximum violation for 62.99 % of the solution is 0, i.e., they are feasible even when checked using finite precision and compensated summation [20]. We note that the use of compensated summation here has a small effect, increasing the fraction of solutions with a 0 violation by 0.27 %.

For 98.8 % of the solutions the maximum violation is smaller than 10^{-11}. We eliminate the solutions for which the maximum violation is 10^{-9} or more.

For 152 instances out of 159 the optimal solution is known, therefore we can analyze the objective value of the solutions found by GENERATESOLUTIONS on these instances. We compute the relative distance from the known optimum f^* of the best solution value $\bar{f}$ among the solutions accepted by GENERATESOLUTIONS, using the formula: $(\bar{f} - f^*)/|f^*|$ if $f^* \neq 0$, or $(\bar{f} - f^*)$ otherwise. In 138 cases out of 152, this value is < 0.5 %, hence we found a solution that satisfies our criteria and can be considered optimal. For the remaining 14 cases, the average relative distance is 231.02 %. This large value is due to a small number of instances where GENERATESOLUTIONS returns solutions of poor quality: `markshare1` (200 % away from the optimum), `markshare2` (2,100 % away from the optimum), `satellites1-25` (700 % away from the optimum).

The 51 instances in bold face in Table 13 form the FAILURE SET. The selection is discussed in Sect. 4.2. The 69 instances in italics in Table 13 form the EXTENDED SET used to validate our results on the FAILURE SET. The EXTENDED SET contains all instances of Table 13 that are not part of FAILURE SET and for which 300 dives require $<$ 24 h of CPU time and generate $<$ 30 failures of Type 3.

The instances, solutions and code used in this paper are available on the website of the authors.

Appendix B: Parameter sensitivity

This section contains a sensitivity analysis of the failure and rejection rates when small changes are applied to the parameters of the cut generator BESTGEN from Table 7, denoted by g^* in this section. Experiments were run in the Condor setup.

We start with the parameterization p^* of g^* given in Table 7 and we investigate changes in a single parameter at a time. For each parameter i, we want to find a range R_i such that changing the value in p^* of p_i^* to any value in R_i yields a generator such

Table 13 Number of feasible solutions found by GENERATESOLUTIONS on each instance

10teams	103	*flugpl*	105	*momentum1*	27	*pp08aCUTS*	120
30n20b8	17	**gen**	44	*momentum2*	34	*pp08a*	117
a1c1s1	65	**gesa2**	97	momentum3	1	protfold	6
acc-tight5	52	**gesa2_o**	101	msc98-ip	11	*pw-myciel4*	52
aflow30a	107	**gesa3**	112	mspp16	56	**qiu**	125
aflow40b	113	**gesa3_o**	101	mzzv11	104	*qnet1*	110
air03	103	**glass4**	179	mzzv42z	104	*qnet1_o*	104
air04	109	**gmu-35-40**	71	n3div36	75	rail507	58
air05	105	**gt2**	2	n3seq24	7	*ran16x16*	69
app1-2	8	**harp2**	129	*n4-3*	95	*rd-rplusc-21*	2
arki001	58	*iis-100-0-cov*	53	*neos-1109824*	58	*reblock67*	27
atlanta-ip	32	*iis-bupa-cov*	56	neos-1337307	43	**rgn**	102
bab5	67	*iis-pima-cov*	56	*neos-1396125*	83	*rmatr100-p10*	56
beasleyC3	69	*khb05250*	105	**neos13**	67	*rmatr100-p5*	60
bell3a	102	*l152lav*	106	neos-1601936	67	*rmine6*	72
bell5	102	*lectsched-4-obj*	59	**neos18**	57	*rocII-4-11*	19
biella1	65	*liu*	315	neos-476283	33	*rococoC10-001000*	74
bienst2	86	*lseu*	105	*neos-686190*	60	**roll3000**	156
binkar10_1	55	*m100n500k4r1*	5	neos-849702	304	**rout**	125
blend2	115	**macrophage**	66	*neos-916792*	73	satellites1-25	1500
bley_xl1	56	*manna81*	101	neos-934278	8	**set1ch**	127
cap6000	111	map18	128	*net12*	106	*seymour*	113
core2536-691	11	map20	127	netdiversion	54	sp97ar	122
cov1075	57	*markshare1*	117	**newdano**	73	*sp98ic*	68
csched010	98	*markshare2*	117	**noswot**	104	*sp98ir*	64
dano3mip	10	*mas74*	159	**ns1688347**	6	**stein27**	102
danoint	2	**mas76**	137	ns1758913	65	*stein45*	103
dcmulti	117	**maxgasflow**	1236	*ns1830653*	189	*swath*	124
dfn-gwin-UUM	72	mcsched	70	*nsrand-ipx*	114	t1717	16
dln	52	**mik.250-1-100.1**	54	nw04	105	tanglegram1	11
ds	23	*mine-166-5*	70	opm2-z7-s2	65	*tanglegram2*	56
egout	3	*mine-90-10*	71	**opt1217**	2	**timtab1**	132
eil33.2	66	*misc03*	102	**p0033**	104	*timtab2*	170
eilB101	95	**misc06**	107	**p0201**	103	**tr12-30**	217
enigma	3	*misc07*	105	*p0282*	106	unitcal_7	133
enlight13	22	**mitre**	82	**p0548**	2	**vpm1**	106
ex9	23	**mkc**	123	**p2756**	109	**vpm2**	113
fast0507	108	*mod008*	102	*pg5_34*	62	vpphard	19
fiber	106	*mod010*	4	**pigeon-10**	2	*zib54-UUE*	70
fixnet6	111	**modglob**	203	*pk1*	108		

Instances in bold face are in the FAILURE SET. Instances in italics are in the EXTENDED SET

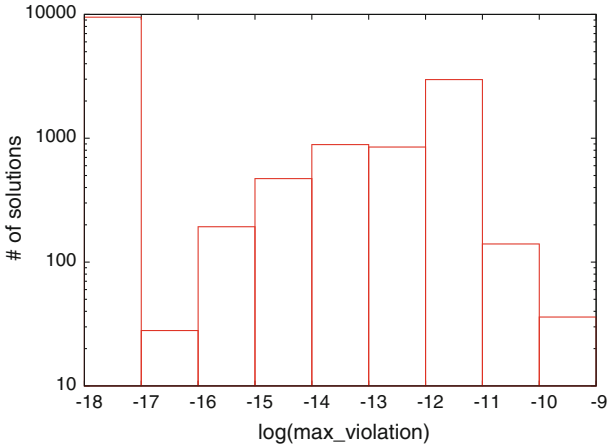

Fig. 14 Maximum absolute violation of the solutions found by GENERATESOLUTIONS

that both a Friedman test on the rejection rate and a Friedman test on the failure rate deem the generator comparable to g^*. We then determine the variation of the failure rate and cut rejection rate for values slightly outside of R_i.

The algorithm we use is a simple sampling algorithm. It maintains an interval $\underline{R_i}$ containing the convex hull of all tested values of p_i for which the modified generator is deemed similar to g^*. The initial interval $\underline{R_i}$ is the single point p^*. The algorithm also maintains the smallest interval $\overline{R_i}$ containing $\underline{R_i}$ such that one tested point on each side of $\underline{R_i}$ is deemed different between g^* and the modified generator. The initial interval $\overline{R_i}$ is chosen as the initial range of parameter i used in Sect. 6.4. However, if we already know from previous experiments one value p_i smaller (resp. larger) than p_i^* such that the corresponding cut generator is not equivalent to g^*, we remove all values smaller (resp. larger) than p_i from the initial interval $\overline{R_i}$.

At each iteration, we select k parameter values $p_1, \ldots, p_k \in \overline{R_i} \setminus \underline{R_i}$, compare the corresponding cut generators with g^* and update both $\underline{R_i}$ and $\overline{R_i}$. The algorithm continues until we are satisfied with the gap between $\underline{R_i}$ and $\overline{R_i}$. We use $k = 10$, except when the gap between $\underline{R_i}$ and $\overline{R_i}$ is very small. In that case, we use a smaller value for k. The tested values $p_1, \ldots, p_k$ are equally spaced in $\overline{R_i} \setminus \underline{R_i}$, with $k/2$ points on each side of $\underline{R_i}$. We use a logarithmic scale for all parameters that have a logarithmic scale in Table 3.

A similar algorithm is used to compute intervals $\underline{F_i}$ and $\overline{F_i}$ such that all generators in $\underline{F_i}$ have a safety level similar to g^* according a Friedman test on the failure rate, without regard to the rejection rate. The interval $\overline{F_i}$ contains $\underline{F_i}$ and one tested point on each side of $\underline{F_i}$ for which the modified generator is deemed different to g^* in term of safety.

Results are reported in Table 14. The columns are as follows. "p^*" contains the optimal value of the parameter in cut generator #1 of Table 7. For the parameters that are not reported in Table 7, we use the value corresponding to disabling the cut modification or numerical check that employs the parameter. For the `_LUB` parameters,

Table 14 Sensitivity analysis of the parameters of the reference cut generator

Parameter	p^*	$\underline{R}$	$\overline{R}$	Increase	$\underline{F}$	$\overline{F}$
AWAY	10^{-2}	$[9.9 \cdot 10^{-3}, 1.1 \cdot 10^{-2}]$	$[9.8 \cdot 10^{-3}, 1.2 \cdot 10^{-2}]$	$\Longrightarrow$	$[5.0 \cdot 10^{-3}, 10^{-1}]$	$[2.0 \cdot 10^{-3}, 1]$
EPS_COEFF	10^{-11}	$[10^{-14}, 10^{-8}]$	$[10^{-15}, 10^{-7}]$	$\Longleftarrow$	$[10^{-18}, 10^{-6}]$	$[0, \infty]$
EPS_RELAX_ABS	10^{-11}	$[10^{-12}, 2.0 \cdot 10^{-11}]$	$[10^{-13}, 4.0 \cdot 10^{-11}]$	$\Longrightarrow$	$[10^{-14}, 10^{-9}]$	$[10^{-15}, 10^{-8}]$
EPS_RELAX_REL	10^{-13}	$[10^{-14}, 10^{-12}]$	$[10^{-15}, 10^{-11}]$	$\Longrightarrow$	$[10^{-18}, 10^{-10}]$	$[0, 10^{-9}]$
MAX_DYN	10^6	$[9.2 \cdot 10^5, 1.1 \cdot 10^6]$	$[8.4 \cdot 10^5, 1.2 \cdot 10^6]$	$\Longleftarrow$	$[10^5, 10^7]$	$[0, 5.0 \cdot 10^7]$
MIN_VIOL	0	$[0, 10^{-9}]$	$[0, 10^{-7}]$	$\Longrightarrow$	$[0, 10^{-3}]$	$[0, \infty]$
MAX_SUPP_ABS	∞	$[2.0 \cdot 10^3, 5.0 \cdot 10^3]$	$[1.5 \cdot 10^3, \infty]$	$\Longleftarrow$	$[1.0 \cdot 10^3, 5.0 \cdot 10^3]$	$[0, \infty]$
MAX_SUPP_REL	1	$[9.2 \cdot 10^{-1}, 1.0]$	$[8.8 \cdot 10^{-1}, 1.0]$	$\Longleftarrow$	$[0, 1.0]$	$[0, 1.0]$
EPS_ELIM	0	$[0, 0]$	$[0, 10^{-20}]$	—	$[0, 0]$	$[0, 10^{-20}]$
MAX_DYN_LUB	10^6	$[5.0 \cdot 10^5, 10^7]$	$[10^5, 5.0 \cdot 10^7]$	$\Longleftarrow$	$[5.0 \cdot 10^3, 5.0 \cdot 10^8]$	$[0, \infty]$
EPS_COEFF_LUB	10^{-11}	$[10^{-11}, 10^{-11}]$	$[10^{-12}, 10^{-10}]$	—	$[10^{-11}, 10^{-11}]$	$[10^{-12}, 10^{-10}]$

we set the value to the corresponding non-LUB parameter. In this table, LUB was set to 10^3 (see Sect. 5.1). Columns "$\underline{R}$", "$\overline{R}$", "$\underline{F}$", and "$\overline{F}$" contain the four intervals of interest for each parameter. "Increase" indicates the direction along which the cut rejection rate increases; the column contains "–" if our experiments did not provide enough information for computing the direction.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)
2. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. **34**(4), 361–372 (2006)
3. Ailon, N.: Aggregation of partial rankings, p-ratings and top-m lists. Algorithmica **57**, 284–300 (2010)
4. Applegate, D.L., Cook, W., Dash, S., Espinoza, D.G.: Exact solutions to linear programming problems. Oper. Res. Lett. **35**(6), 693–699 (2007)
5. Balas, E., Ceria, S., Cornuéjols, G., Natraj, N.: Gomory cuts revisited. Oper. Res. Lett. **19**(1), 1–9 (1996)
6. Bixby, R., Rothberg, E.: Progress in computational mixed integer programming—a look back from the other side of the tipping point. Ann. Oper. Res. **149**(1), 37–41 (2007)
7. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Optima **58**, 12–15 (1998)
8. COIN-OR Branch-and-Cut 2.7 stable r1676. https://projects.coin-or.org/Cbc/
9. COIN-OR Linear Programming 1.14 stable r1753. https://projects.coin-or.org/Clp/
10. COIN-OR Cut Generation Library 0.57 stable r1033. https://projects.coin-or.org/Cgl/
11. Conover, W.J.: Practical Nonparametric Statistics, 3rd edn. Wiley, New York (1999)
12. Cook, W., Dash, S., Fukasawa, R., Goycoolea, M.: Numerically safe Gomory mixed-integer cuts. INFORMS J. Comput. **21**(4), 641–649 (2009)
13. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: An exact rational mixed-integer programming solver. In: Günlük, O., Woeginger, G.J. (eds.) Proceedings of IPCO 2011. Lecture Notes in Computer Science, vol. 6655, pp. 104–116. Springer, Berlin (2011)
14. Espinoza, D.G.: On linear programming, integer programming and cutting planes. PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology (2006)
15. Gomory, R.E.: An algorithm for the mixed-integer problem. Technical Report RM-2597, RAND Corporation (1960)
16. Gutmann, H.-M.: A radial basis function method for global optimization. J. Glob. Optim. **19**, 201–227 (2001). doi:10.1023/A:1011255519438
17. Hemker, T.: Derivative free surrogate optimization for mixed-integer nonlinear black-box problems in engineering. Master's thesis, Technischen Universität Darmstadt (2008)
18. Holmström, K., Quttineh, N.H., Edvall, M.M.: An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. Optim. Eng. **9**(4), 311–339 (2008)
19. Ibm, ILOG CPLEX 12.2 User's Manual. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
20. Kahan, W.: Pracniques: further remarks on reducing truncation errors. Commun. ACM **8**(1), 40 (1965)
21. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Math. Program. Comput. **3**(2), 103–163 (2011)
22. Litzkow, M., Livny, M., Mutka, M.: Condor—a hunter of idle workstations. In: Proceedings of the 8th International Conference on Distributed Computing System, pp. 104–111 (1998)
23. Margot, F.: Testing cut generators for mixed-integer linear programming. Math. Program. Comput. **1**(1), 69–95 (2009)
24. Neumaier, A., Shscherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Program. A **99**, 283–296 (2004)
25. R Development Core Team: R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN: 3-900051-07-0 (2008)
26. Regis, R., Shoemaker, C.: Improved strategies for radial basis function methods for global optimization. J. Glob. Optim. **37**, 113–135 (2007). doi:10.1007/s10898-006-9040-1

27. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman & Hall/CRC, London (2007)
28. van Zuylen, A., Williamson, D.P.: Deterministic algorithms for rank aggregation and other ranking and clustering problems. Math. Oper. Res. **34**, 594–620 (2009)