Safe and Verified Gomory Mixed Integer Cuts in a Rational MIP Framework *

Leon Eifler[1] Ambros Gleixner[1,2]

[1]Zuse Institute Berlin
[2]HTW Berlin

July 25, 2023

Abstract

This paper is concerned with the exact solution of mixed-integer programs (MIPs) over the rational numbers, i.e., without any roundoff errors and error tolerances. Here, one computational bottleneck that should be avoided whenever possible is to employ large-scale symbolic computations. Instead it is often possible to use safe directed rounding methods, e.g., to generate provably correct dual bounds. In this work, we continue to leverage this paradigm and extend an exact branch-and-bound framework by separation routines for safe cutting planes, based on the approach first introduced by Cook, Dash, Fukasawa, and Goycoolea in 2009. Constraints are aggregated safely using approximate dual multipliers from an LP solve, followed by mixed-integer rounding to generate provably valid, although slightly weaker inequalities.

We generalize this approach to problem data that is not representable in floating-point arithmetic, add routines for controlling the encoding length of the resulting cutting planes, and show how these cutting planes can be verified according to the VIPR certificate standard. Furthermore, we analyze the performance impact of these cutting planes in the context of an exact MIP framework, showing that we can solve 21.5% more instances and reduce solving times by 26.8% on the MIPLIB 2017 benchmark test set.

*The work for this article has been conducted within the Research Campus Modal funded by the German Federal Ministry of Education and Research (BMBF grant numbers 05M14ZAM, 05M20ZBM).

1 Introduction

Even though the problem class of mixed-integer programs (MIPs) is $\mathcal{NP}$-hard [10], state-of-the-art MIP solvers manage to solve a large number of such problems with up to millions of variables and constraints [17]. While the core algorithm, LP-based branch-and-bound, is straightforward, it is a long list of sophisticated solving techniques such as presolving, cutting planes, primal heuristics, conflict analysis, and branching rules that make this remarkable performance possible.

However, virtually all established solvers that contain these techniques rely on fast floating-point arithmetic, combined with numerical error tolerances to achieve a high degree of numerical stability. For most applications, especially in industry, this is completely sufficient. Nevertheless, some problem cases require exact proofs of optimality or infeasibility without the slight numerical inaccuracies that result from rounding errors in floating-point arithmetic. This is the case when mixed-integer programs are used as a tool in mathematics, e.g., to computationally investigate open conjectures. Recent examples of such approaches include [5, 7, 14, 24, 27, 34]. Examples for industrial applications where the correctness of results is paramount are, e.g., hardware verification [1], compiler optimization [39], or more recently infeasibility analysis in hydro unit commitment [35].

To the best of our knowledge, the first fully general, exact MIP solver that can handle MIPs with rational input data is presented by Cook, Koch, Steffy, and Wolter [12]. Their hybrid-precision framework uses both symbolic as well as numeric computations and applies different dual bounding methods [16, 32, 37] to generate provably valid dual bounds. However, besides employing reliability pseudocost branching, this framework still lacks the previously mentioned advanced solving techniques since their application in the roundoff-error free setting is often not trivial.

While a direct translation of methods using symbolic computations is always possible, it is often prohibitively slow in practice. Safe methods that try to avoid symbolic computations in favour of safe rounding techniques can often provide better results. A first step in the direction of closing this algorithmic gap between exact and inexact MIP was established [13] by revising the approach of Cook et al. [12] and extending it by symbolic presolving routines, as well as a repair step that enables the usage of all existing primal heuristics [13].

One key feature missing to date is a separation routine for cutting planes, which is known to be among the most important components to make MIP solvers perform well in practice, as reported, e.g., by Achterberg and Wun-

derling [2]. Among different types of cutting planes, they found that especially mixed-integer rounding (MIR) cuts (48% speedup), as well as Gomory cuts (28% speedup), seem to provide the most benefit. However, designing an efficient roundoff-error-free separation procedure for Gomory cuts is nontrivial since a purely symbolic approach would require an exact LP tableau row. To compute the tableau, the LP relaxation needs to be solved exactly and LP rows need to be aggregated, both of which are computationally expensive operations. Instead, Cook, Dash, Fukasawa, and Goycoolea [11] introduced a technique that can be used to construct numerically safe cuts, using safe rounding techniques. These cuts are guaranteed to be exactly feasible, without any symbolic computations, but at the cost of slightly weaker efficacy. While the focus of Cook et al. was to provide safe cutting planes within a normal floating-point MIP solver, the technique can be generalized to the exact MIP setting.

This generalization is our first contribution. We show how to relax rational problem data to be usable for the safe rounding technique and generalize the approach of [11] to allow for negative variable bounds. Our second contribution is adapted scaling and a new post-processing technique that improves the performance of safe cutting planes in the exact MIP framework. Finally, our third major contribution is the independent verification of these cutting planes using the VIPR [9] certificate format.

To motivate the last point further, note that although the exact MIP framework guarantees an exactly optimal solution *in theory*, the implemented computer program is very complex and a user might not have full confidence that the result is correct *in practice*. Our goal is for the solving algorithm to provide a certificate of optimality for each instance it solves which can be verified independently. Such proof-logging features are standard in the world of satisfiability solvers [22, 38] and have also been adapted to pseudo-Boolean [15, 19] and maximum satisfiability solvers [4]. In the context of integer programming, certifaction has been employed in the past for the travelling salesman problem [3], to verify an optimal tour with 85,900 cities by a problem-specific branch-and-cut certificate. For general purpose exact MIP, the VIPR certificate standard [9] is a possibility to encode a proof for the optimality of a branch-and-bound tree, using only elementary branching logic and basic arithmetic. We introduce an explicit verification of MIR cuts using only the logic of VIPR, as well as an algorithm to account for the rounding errors introduced through the safe rounding procedure.

The overall goal of this research is twofold. First, we aim to further reduce the algorithmic gap between exact and floating-point MIP solvers to make exact MIP solving a real alternative for researchers going forward.

Second, we aim to better understand the practical differences between numerically safe cutting planes in exact MIP and normal ones in conventional floating-point MIP, especially the impact they have on LP solving behaviour. We conduct a detailed computational study, investigating both of these points.

The paper is structured as follows. In Section 2, we present the construction of numerically safe Gomory mixed-integer cuts in an exact MIP setting, discuss scaling, and introduce a new post-processing routine that controls the encoding length of cut coefficients. In Section 3, we present verification routines for MIR cuts and a completion algorithm that accounts for the rounding errors introduced during safe rounding. Section 4 constitutes our computational study, where the performance of all new features is evaluated and compared against an analogue experiment within a floating-point solver. We conclude with remarks on further research directions in Section 5.

All the code for completing and checking the certificates is freely available on GitHub [8].

2 Safe Gomory mixed-integer cuts for exact MIP

We consider the general rational mixed-integer program (MIP) of the form

$$\min\{\, c^T x : Ax \leq b,\ \ell \leq x \leq u,\ \text{and } x_j \in \mathbb{Z} \text{ for all } j \in I \,\}$$

where $A \in \mathbb{Q}^{m \times n}, c, \ell, u \in \mathbb{Q}^n, b \in \mathbb{Q}^m$ and $I \subseteq \{1, \ldots, n\}$. We denote the set of all feasible points for this MIP by P.

The variable bounds are allowed to be $\pm\infty$, but we assume throughout this paper that all variables appearing in a cut have either a finite lower or a finite upper bound:

Assumption 1. *For any valid inequality $a^T x \leq b$ encountered during the construction of a cut, if $a_i \neq 0$ then either $u_i < \infty$ or $\ell_i > -\infty$ holds. Thus we can choose sets U, L such that $u_i < \infty$ for all $i \in U$, $\ell_i > -\infty$ for all $i \in L$, and $a_i = 0$ for all $i \notin U \cup L$.*

Note that this assumption can always be fulfilled by splitting variables with infinite bounds into their positive and negative part; however, it is necessary to introduce new auxiliary variables explicitly for these parts. In practice, after presolving, Assumption 1 usually seems to hold and we did not encounter any problems where cuts needed to be discarded in our experiments.

To give an overview, our steps for generating safe MIR cuts in general and safe Gomory mixed-integer cuts in particular are as follows:

1. *Approximate* rows by a floating-point representable relaxation.
2. Generate a valid, floating-point representable inequality for P using *safe aggregation* of rows.
3. Construct the MIR cut:
   (a) Safely *transform* this inequality into non-negative variable space.
   (b) Apply the *safe MIR technique* to the transformed inequality and retransform to original variable space.
   (c) *Substitute* slack variables.
4. Post-process the cut:
   (a) *Scale* the resulting cut to improve numerical stability and possibly make coefficients integer.
   (b) Control the *encoding length* of the generated cut.

Steps 2 and 3 are in essence the procedure described in [11], although slight differences exist because we allow for variables with negative lower bounds in ℓ. In the following sections, we describe all steps in more detail.

2.1 Safe MIR cuts by directed rounding

Let $\alpha^T x \leq \beta$ be a valid inequality for P. If all integer variables are non-negative, then the mixed-integer rounding cut

$$\sum_{i \in I} \Big(\lfloor \alpha_i \rfloor + \frac{(f_i - f)^+}{1 - f} \Big) x_i + \sum_{i \notin I, a_i < 0} \frac{\alpha_i}{1 - f} x_i \leq \lfloor \beta \rfloor \tag{1}$$

with $f = \beta - \lfloor \beta \rfloor$ and $f_i = \alpha_i - \lfloor \alpha_i \rfloor, i \in I$, is valid for P [30]. If not all integer variables are non-negative but Assumption 1 holds, then we can transform all integer variables to a non-negative space via

$$x'_i := \begin{cases} u_i - x_i & \text{for all } i \in U \cap I, \\ x_i - \ell_i & \text{for all } i \in L \cap I. \end{cases} \tag{2}$$

Gomory mixed-integer (GMI) cuts are obtained by applying the MIR technique to one row of an optimal simplex tableau that corresponds to an integer variable with a fractional LP solution value.

The naïve approach to computing exact GMI cuts for P by roundoff-error-free rational arithmetic can become prohibitively expensive not only due to the operations involved in (1). Most of all, obtaining $\alpha^T x \leq \beta$ as an exact tableau row requires a rational LU factorization of the basis matrix. In recent approaches to solve MIPs exactly, however, only an approximate floating-point LP solution is at hand [13].

Instead of performing all computations in exact arithmetic, a better option is to construct numerically safe cuts as proposed by [11]. By usnoing safe directed rounding, valid inequalities can be generated that are guaranteed to be exactly feasible for P, without symbolic computations, though at the cost of obtaining slightly weaker cuts. Hence, let us first define the notation used for safe rounding that will be needed throughout this paper.

Let $\mathbb{F} \subseteq \mathbb{Q}$ denote the set of floating-point numbers. In practice, these will usually be standard IEEE double-precision numbers with 11 bits for the exponent and 52 bits for the mantissa. This means any number $f \in \mathbb{F}$ can be written as $f = s * m * 2^e$, where the sign is $s \in \{-1, 1\}$, the mantissa $m = (1, M)_2$ is a binary number between 1 and 2 with 52 bits, and the exponent e is a binary number with up to 11 bits, centered around 0, i.e., $e \in [-1022, 1023]$. For a detailed description of the format, see e.g., [33]. It is clear that not all rational numbers can be exactly represented in this format and that $\mathbb{F}$ is not closed under addition, subtraction, multiplication, or division. Hence, the result of these operations is rounded up or down to a nearest representable number; the rounding direction can be controlled in computer code.

Definition 1. *Let $x \in \mathbb{Q}$ be a rational number. We denote the closest upper and lower floating-point representable approximations of x in $\mathbb{F}$ by*

$$\overline{x} := \min\{y \in \mathbb{F} : y \geq x\} \qquad \text{and} \qquad \underline{x} := \max\{y \in \mathbb{F} : y \leq x\},$$

respectively. We call x $\mathbb{F}$-representable *if $x \in \mathbb{F}$. We call an inequality $a^T x \leq b$* $\mathbb{F}$-representable *if $a \in \mathbb{F}^n$ and $b \in \mathbb{F}$. Finally, for $n \geq 2$ and $\lambda_1, \ldots, \lambda_n \in \mathbb{F}$, we define recursively*

$$\overline{\sum_{i=1}^{n} \lambda_i} := \overline{\lambda_1 + \overline{\sum_{i=2}^{n} \lambda_i}} \qquad \text{and} \qquad \underline{\sum_{i=1}^{n} \lambda_i} := \underline{\lambda_1 + \underline{\sum_{i=2}^{n} \lambda_i}}.$$

We handle other arithmetic operations, i.e., subtraction, multiplication, and division, as well as combinations thereof, analogously. We note that this is consistent with how modern computers handle floating-point numbers. Specifically, computers have the following rounding modes: down, up,

towards zero, and to nearest. These can be changed by the algorithm to ensure the desired over- or underestimation.

Note that this definition means that no rational computations are necessary to compute $\overline{\sum_{i=1}^{n} \lambda_i}$. On the other hand, the order of summation changes the result since we round after each pairwise addition.

2.2 Approximating problem data

Some coefficients in rows of the original problem formulation may not be in $\mathbb{F}$. Affected inequalities can be made $\mathbb{F}$-representable by relaxing them slightly according to the following formula, which relies on Assumption 1.

Lemma 1. *Let $a^T x \leq b$, with $a \in \mathbb{Q}^n, b \in \mathbb{Q}$ be valid for P. Then the $\mathbb{F}$-representable inequality*

$$\sum_{i \in U} \overline{a_i} x_i + \sum_{i \in L} \underline{a_i} x_i \leq \overline{b + \sum_{i \in U} (\overline{a_i} - \underline{a_i}) \overline{u_i} + \sum_{i \in L} (\underline{a_i} - \overline{a_i}) \underline{\ell_i}}$$

is also valid for P.

Proof. The first step is to transform all variables into a non-negative form. Define x'_i according to (2) for all $i \in I$. Substituting x for x' yields

$$\sum_{i \in U} a_i (u_i - x'_i) + \sum_{i \in L} a_i (\ell_i + x'_i) \leq b.$$

Moving all constants to the right-hand side

$$\sum_{i \in U} -a_i x'_i + \sum_{i \in L} a_i x'_i \leq b - \sum_{i \in U} a_i u_i - \sum_{i \in L} a_i \ell_i.$$

Since $x'_i \geq 0$, we can round down all coefficients on the left-hand side. With $\underline{-a_i} = -\overline{a_i}$, we get

$$\sum_{i \in U} -\overline{a_i} x'_i + \sum_{i \in L} \underline{a_i} x'_i \leq b - \sum_{i \in U} a_i u_i - \sum_{i \in L} a_i \ell_i.$$

Substituting x for x' yields

$$\sum_{i \in U} \overline{a_i} x_i + \sum_{i \in L} \underline{a_i} x_i \leq b + \sum_{i \in U} (\overline{a_i} - a_i) u_i + \sum_{i \in L} (\underline{a_i} - a_i) \ell_i.$$

Finally, rounding the whole right-hand side upwards, we get the final inequality. □

This procedure of implicitly transforming to non-negative variable space, rounding in the correct direction, retransforming, and finally rounding again is the essential technique from [11] that is used for all the safe operations in this manuscript. Since the correction factor on the right-hand side of the inequalities always has non-negative coefficients for $i \in U$ and non-positive coefficients for $i \in L$, we always use $\overline{u}$ or $\underline{\ell}$, if the bounds are not $\mathbb{F}$-representable. To avoid notational clutter, we will omit the bars on the bounds for the remainder of the manuscript.

2.3 Safe aggregation of rows

The technique from the previous Lemma can also be applied to aggregate two $\mathbb{F}$-representable rows safely.

Corollary 2. *Let $a^T x \leq b$ and $c^T x \leq d$ be two valid, $\mathbb{F}$-representable inequalities, and $0 < \lambda \in \mathbb{F}$. If Assumption 1 holds, we can generate an $\mathbb{F}$-representable, valid approximation of the aggregated inequality $(a + \lambda c)^T x \leq b + \lambda d$ by*

$$\sum_{i \in U} \overline{\alpha_i} x_i + \sum_{i \in L} \underline{\alpha_i} x_i \leq \overline{b + \lambda d + \sum_{i \in U, u_i > 0} (\overline{\alpha_i} - \underline{\alpha_i}) u_i + \sum_{i \in L, \ell_i < 0} (\underline{\alpha_i} - \overline{\alpha_i}) \ell_i},$$

with $\alpha_i := a_i + c_i \lambda$.

Proof. The proof follows the same steps as in Lemma 1. □

The rows of an optimal simplex tableau, which form the base inequalities $\alpha^T x \leq \beta$ to derive GMI cuts via the MIR formula (1), can be alternatively obtained as an aggregation of inequalities from the LP relaxation. As a matter of fact, this is also how GMI cuts are computed in floating-point MIP solvers to minimize numerical errors. Similarly, we can perform safe aggregation of problem inequalities according to Corollary 2, using multipliers from an *approximate* row of the basis inverse to generate a valid, $\mathbb{F}$-representable base inequality $\alpha^T x \leq \beta$. This is hopefully a good approximation to the exact simplex tableau row, at least if the approximate floating-point LP solution from which we obtain the dual multipliers is a good approximation to the optimum of the exact, rational LP relaxation.

2.4 Constructing the MIR cut

After obtaining such a row $\alpha^T x \leq \beta$, we proceed as we normally would in deriving a MIR cutting plane. In the following, we go through those steps

again to highlight at which points operations have to be performed in a numerically safe fashion.

First, for each variable one of the bounds is chosen to transform to non-negative variable space, since the MIR formula requires non-negative variables. If the variable has both finite upper and lower bound, we use the bound that is closest to the value of the current LP solution for that variable. Denote by U the index set of variables for which the upper bound is chosen, by L the set for which the lower bound is chosen, and by x'_i the transformed variables according to (2).

We first obtain the aggregated inequality in transformed space

$$\sum_{i \in U} -\alpha_i x'_i + \sum_{i \in L} \alpha_i x'_i \leq \overline{\beta - \sum_{i \in U} \alpha_i u_i - \sum_{i \in L} \alpha_i \ell_i} := d. \tag{3}$$

Next, we compute the coefficients in the cut, according to (1). Denote the transformed coefficients by

$$\alpha'_i = \begin{cases} \alpha_i & \text{for } i \in L \\ -\alpha_i & \text{for } i \in U, \end{cases} \tag{4}$$

and the fractionalities by $f = d - \lfloor d \rfloor$, $f_i = \alpha'_i - \lfloor \alpha'_i \rfloor$. Then the safe MIR cut in the transformed space is

$$\sum_{i \in I} \underline{\left(\lfloor \alpha'_i \rfloor + \frac{(f_i - f)^+}{1 - f} \right)} x'_i + \sum_{i \notin I : \alpha'_i < 0} \underline{\left(\frac{\alpha'_i}{1 - f} \right)} x'_i \leq \lfloor d \rfloor. \tag{5}$$

Transforming back to original variable space as in Lemma 1 yields the final safe MIR cut.

We give some additional, slightly technical details that will be especially relevant for the verification described in Section 3. Whenever we consider a row $a_i^T x \leq b_i$ for aggregation, we implicitly turn it into an equation using a slack variable, i.e., $a_i^T x + s_i = b_i, s_i \geq 0$. This allows two generalizations. Firstly, we can aggregate with a multiplier of any sign. If we use a negative multiplier λ_i for a row without additional integrality information, the slack is treated as a continuous variable and thus gets assigned the coefficient $\frac{\lambda_i}{(1-f)}$, where f is the fractionality of the one-row relaxation's right-hand side. Secondly, and more importantly, if we know that all non-zero coefficients as well as their corresponding variables are integer, then s can be treated as an integer variable for the MIR procedure and gets assigned the coefficient $\lfloor \lambda_i \rfloor + \frac{(f_i - f)^+}{(1-f)}$, regardless of the sign of λ.

At the very end, the slack is eliminated using its definition $s_i = b_i - a_i^\top x$ to return to the space of original variables. Since s_i is by definition non-negative, it is straightforward to do this safely using directed rounding. We simply need to underapproximate the coefficient and perform the back-substitution exactly as an aggregation according to Corollary 2.

2.5 Post-processing steps

The steps outlined in the previous section yield a new valid, $\mathbb{F}$-representable inequality $\alpha^T x \leq \beta$. We present two post-processing ideas that aim to improve LP performance after adding such inequalities.

2.5.1 Scaling

An important aspect to improve the numerical stability of cutting plane algorithms, both in the exact and in the inexact setting, is scaling. Large coefficient ranges in the problem are known to detriment the performance and accuracy of LP solvers. This is especially relevant for the exact MIP setting. Given a cut $\alpha^T x \leq \beta$ and a scaling coefficient $s \geq 0$, we can safely scale the cut using the same approach as in the previous subsection.

Lemma 3. *Given a valid cut $\alpha^T x \leq \beta$ and a scaling factor $s \geq 0$, the safely scaled cut*

$$\sum_{i \in U} \overline{s\alpha_i} x_i + \sum_{i \in L} \underline{s\alpha_i} x_i \leq \overline{s\beta + \sum_{i \in U, u_i > 0} (\overline{s\alpha_i} - \underline{s\alpha_i}) u_i + \sum_{i \in L, \ell_i < 0} (\underline{s\alpha_i} - \overline{s\alpha_i}) \ell_i}$$

is valid and $\mathbb{F}$-representable.

Proof. Analogous to Lemma 1. □

We employ two different scaling approaches, similar to the ones that are used by the corresponding floating-point algorithm in SCIP. If continuous variables appear in the cut, we simply scale to *equilibrium*, meaning we scale such that the largest absolute value of any coefficient becomes 1.

If the cut contains only integer variables, we attempt to find a scaling factor that makes all coefficients close to integer. This approach uses the euclidean algorithm to compute a rational approximation and then multiplies by the least common multiple (LCM) of all denominators. We impose a limit of 10^{-6} on the error in the rational representation and of 10^5 on the size of the LCM. If this proves successful, we round the coefficients to the nearest integer and offset the right-hand side to account for the difference.

Consequently, we can round down the right-hand side to strengthen the cut. If some of the working limits are exceeded and the approach fails, we revert to scaling to equilibrium. This is the same approach that is used in presolving for linear constraints and is detailed in [1, Algorithm 10.2.4].

2.5.2 Controlling the encoding length

If scaling to integer values as outlined in the previous section was not possible or not successful, the coefficients α_i will often have large encoding lengths when represented as a rational number $\alpha_i = \frac{n_i}{d_i}$, $n_i \in \mathbb{Z}, d_i \in \mathbb{N}$.

This is not an issue when the inequality is added to the floating-point relaxation of the LP. However, we sometimes need to (or want to) solve the rational LP relaxation of P exactly, i.e., using an exact LP solver. Our experiments in Section 4 show that the exact LP solver may struggle with LPs that contain cuts with coefficients of large encoding length, in some cases causing large spikes in LP solving times.

This empirical observation also has a theoretical counterpart. The convergence analysis of the LP iterative refinement algorithm for solving rational LPs exactly is related to the smallest possible violation of any basic solution to a rational LP. In turn, this smallest possible violation is linked to the encoding length of the whole problem [18, Lemma 5]. This analysis suggests investigating whether the practical performance of the iterative refinement algorithm can be improved by decreasing the encoding length of the generated cuts.

Given a limit $M > 0$ on the size of denominators allowed in the cut, we wish to compute a relaxation $\sum_{i=1}^{n} \frac{\hat{n}_i}{\hat{d}_i} x_i \leq \hat{b}$ such that $\hat{d}_i \leq M$. Depending on the available bounds for each of the variables, we may additionally require that $\frac{\hat{n}_i}{\hat{d}_i} \leq \alpha_i$ or $\frac{\hat{n}_i}{\hat{d}_i} \geq \alpha_i$, since we use one variable bound to offset the right-hand side for maintaining feasibility of the inequality. The algorithm we use to achieve this approximation is based on *continued fraction approximations* (see, e.g., [36, Section 6.1]). For $r \in \mathbb{Q}$ we compute the continued fraction

$$[r_0; r_1, \ldots, r_n] = r_0 + \cfrac{1}{r_1 + \cfrac{1}{r_2 + \cfrac{1}{r_3 + \ldots}}} \approx r$$

and a sequence of *convergents* $(\frac{p_i}{q_i})_{i=1}^{n}$ using the following recursive formulas:

$$\begin{aligned} \rho_0 &= r & r_0 &= \lfloor \rho_0 \rfloor \\ \rho_{i+1} &= \frac{1}{\rho_i - r_i} & r_{i+1} &= \lfloor \rho_{i+1} \rfloor \end{aligned}$$

for $i = 0, 1, \ldots$ as long as $\rho_i \notin \mathbb{Z}$, and

$$\begin{aligned} p_0 &= r_0 & q_0 &= 1 \\ p_1 &= r_0 r_1 + 1 & q_1 &= r_1 \\ p_{i+1} &= r_{i+1} p_i + p_{i-1} & q_{i+1} &= r_{i+1} q_i + q_{i-1}. \end{aligned}$$

It is known that all convergents $\frac{p_i}{q_i}$ of the continued fraction approximations are *best approximations* in the sense that there exist no better approximations with a smaller denominator than q_i. This has even been proven for a stronger notion of best approximation [25, Theorem 17]. However, if we impose a fixed limit M on the denominator, as is the case here, we are not guaranteed that the best approximation

$$\arg\min \left\{ \left| \frac{n}{d} - r \right| : n \in \mathbb{Z}, d = 1, 2, \ldots, M \right\}$$

with respect to that limit is a convergent of the continued fraction approximation.

To compute the best approximation in that sense we need to consider so-called *intermediate fractions*. If $\frac{p_i}{q_i}, \frac{p_{i+1}}{q_{i+1}}$ are consecutive convergents of a continued fraction approximation, then we define the *intermediate fractions*

$$p_{i+1}^j = j p_i + p_{i-1} \qquad q_{i+1}^j = j q_i + q_{i-1}$$

for $j = 1, \ldots, r_{i+1}$. We can now state the desired best approximation guarantee [25, Theorem 15].

Lemma 4. *Given $M > 0$ and $r \in \mathbb{Q}$, the best approximation of r by a rational number with denominator at most M is either a convergent or an intermediate fraction of the continued fraction approximation for r.*

We mention two more results from the literature that we will use to make computing the best possible approximation as efficient as possible. First, we note that all convergents with even index form a strictly increasing sequence, while the convergents with odd index form a strictly decreasing one [28, Theorem 5], i.e.,

$$\frac{p_0}{q_0} < \frac{p_2}{q_2} < \ldots < \frac{p_{2k}}{q_{2k}} < \ldots < r < \ldots < \frac{p_{2k+1}}{q_{2k+1}} < \ldots < \frac{p_1}{q_1}.$$

Furthermore, we know that the intermediate fractions do the same [28, Theorem 9], i.e., if n is even, then

$$\frac{p_n}{q_n} < \frac{p_{n+2}^1}{q_{n+2}^1} < \ldots < \frac{p_{n+2}^{r_{n+2}}}{q_{n+2}^{r_{n+2}}} = \frac{p_{n+2}}{q_{n+2}} < \ldots .$$

Using this monotonicity, we can formulate an algorithm to compute the best approximation $\frac{n}{d}$ of a rational number r with denominator at most M.

The trivial case of $d \leq M$ requires no work. Otherwise, we compute convergents of the continued fraction approximation for r until $q_n > M$ for some $n \in \mathbb{N}$. Then, since both $\frac{p_{n-1}}{q_{n-1}}$, as well as $\frac{p_n}{q_n}$ are best approximations with respect to their denominators, we know that the best approximation with respect to M is either $\frac{p_{n-1}}{q_{n-1}}$ or one of the intermediate fractions $\frac{p_n^k}{q_n^k}$ for some $k \in \{1, \ldots, r_n\}$. Instead of computing all intermediate fractions and choosing the best one, we know due to monotonicity that the best is the one with the largest possible denominator, i.e., with $k = \lfloor \frac{(M - q_{n-2})}{q_{n-1}} \rfloor$. Furthermore, the intermediate fractions that are best approximations are exactly the ones with $j \geq \lfloor r_n/2 \rfloor + 1$.[1] Hence, it is trivial to check whether the intermediate fraction or the last convergent $\frac{p_{n-1}}{q_{n-1}}$ should be chosen.

One last case to consider is when the corresponding variable is only bounded in one direction but lacks either an upper or lower bound. In that case, we are required to find an approximation that is either not larger or not smaller than the original number r. This poses no additional challenge since we know that

- convergents with even index are always less than or equal to r, and
- convergents with odd index are always greater than or equal to r,

and the same holds for intermediate fractions [28]. W.l.o.g. , assume that we are required to compute an approximation that is not larger than r. Again, assume that $\frac{p_n}{q_n}$ is the first convergent such that $q_n > M$. If n is odd, then we know immediately that $\frac{p_{n-1}}{q_{n-1}}$ is the best approximation, since all intermediate convergents $\frac{p_n^k}{q_n^k} > r$. If n is even, then we know that $\frac{p_{n-1}}{q_{n-1}} > r$ and should therefore not be considered. Instead, we immediately choose the intermediate fraction $\frac{p_n^k}{q_n^k}$ with $k = \lfloor \frac{M - q_{n-2}}{q_{n-1}} \rfloor$, as described above.

A compact algorithmic description of the whole procedure for obtaining the best approximation $\frac{p}{q} \leq r$ is presented in Algorithm 1.

3 Generating elementary certificates

Although we have proven mathematically that the proposed methods are correct and can therefore be used to solve MIPs exactly, it is unrealistic

[1] We could not find a proof for this in the literature and have included it in the Appendix A, Lemma 7

Algorithm 1 Approximate $r = \frac{n}{d} \in \mathbb{Q}$ by a rational number with bounded denominator.

Input: $r = \frac{n}{d} \in \mathbb{Q}, M \in \mathbb{Z}^+$
Output: $p, q \in \mathbb{Z}$ with $\frac{p}{q} = \arg\min\{|\frac{a}{b} - r| : \frac{a}{b} \leq r, b = 1, 2, \ldots, M\}$
  Compute $[r_0; r_1, \ldots, r_n]$ and $\frac{p_1}{q_1}, \ldots, \frac{p_n}{q_n}$ with $q_{n-1} \leq M$, $q_n > M$
  if n odd, i.e., $\frac{p_{n-1}}{q_{n-1}} \leq r < \frac{p_n}{q_n}$ **then**
    $p \leftarrow p_{n-1}$
  else
    $j = \lfloor \frac{M - q_{n-2}}{q_{n-1}} \rfloor$
    $p \leftarrow jp_{n-1} + p_{n-2}$
    $q \leftarrow jq_{n-1} + q_{n-2}$
  end if
  Return $\frac{p}{q}$

to offer a formal guarantee of correctness for an implementation of these methods. Due to the high algorithmic complexity and the sheer code size, it is neither easy for a user to verify that the implementation is indeed correct, nor is it feasible to prove correctness formally. A more realistic goal is to provide a certificate of optimality for each individually solved instance that follows a much simpler logic and can be checked and verified independently of the solving algorithm.

For MIPs, the VIPR [9] certificate format provides a standard for verification. It can be viewed as a tree-less encoding of (the leaves of) a branch-and-bound tree. To guarantee a high degree of confidence, the format only supports the following three elementary steps:

- aggregation of constraints,
- disjunction logic,
- Chvátal-Gomory cuts, i.e., rounding down the right-hand side of a constraint if all coefficients and variables are integer.

We give a short overview of that certificate language, for a more detailed description, see [9]. A VIPR certificate consists of the problem statement, the optimal solution and objective value, followed by a derivation section that proves a lower bound or infeasibility. That derivation section is a list of constraints with proofs of their validity. Bounds on the objective function are also seen as one type of constraint. For an *aggregation*, that proof is a list of multipliers and line indices. For disjunction logic, there

Given

	x_1, x_2	$\in \mathbb{Z}$
$C1:$	$2x_1 + 3x_2$	≥ 1
$C2:$	$3x_1 - 4x_2$	≤ 2
$C3:$	$-x_1 + 6x_2$	≤ 3

Derived		**Reason**	**Assumptions**
$A1:$	$x_1 \leq 0$	{assume}	
$A2:$	$x_1 \geq 1$	{assume}	
$A3:$	$x_2 \leq 0$	{assume}	
$C4:$	$0 \geq 1$	$\{C1 + (-2) \cdot A1 + (-3) \cdot A3\}$	$A1, A3$
$A4:$	$x_2 \geq 1$	{assume}	
$C5:$	$0 \geq 1$	$\left\{\left(-\frac{1}{3}\right) \cdot C3 + \left(-\frac{1}{3}\right) \cdot A1 + 2 \cdot A4\right\}$	$A1, A4$
$C6:$	$x_2 \geq \frac{1}{4}$	$\left\{\left(-\frac{1}{4}\right) \cdot C2 + \left(\frac{3}{4}\right) \cdot A2\right\}$	$A2$
$C7:$	$x_2 \geq 1$	{round up $C6$}	$A2$
$C8:$	$0 \geq 1$	$\left\{\left(-\frac{1}{3}\right) \cdot C2 + (-1) \cdot C3 + \frac{14}{3} \cdot C7\right\}$	$A2$
$C9:$	$0 \geq 1$	{unsplit $C4, C5$ on $A3, A4$}	$A1$
$C10:$	$0 \geq 1$	{unsplit $C8, C9$ on $A2, A1$}	

Figure 1: Certificate example for an infeasible instance [9].

are two operations. A new disjunction can be introduced by printing two *assumption* constraints, and then other constraints can use that assumption as part of a derivation. If at some point a constraint holds for both parts of a disjunction, it can be *unsplit* and the disjunction can be discharged. At the end of the checking procedure, the checker needs to ensure that no undischarged assumptions remain. Figure 1 shows a short example of a VIPR certificate for an infeasible instance. Note that the assumptions column is not provided in the certificate, but is tracked during verification.

For the verification of MIR cuts, let us first consider the theoretical case where all operations (aggregation, rounding, substitution) are carried out in exact arithmetic. In that case, assuming the aggregation has already been certified, a disjunctive proof can be extracted directly from [30].

Lemma 5. *Given the simple case of a two-variable set*

$$X = \{(w, u) \in \mathbb{Z} \times \mathbb{R}_+ : w - u \leq b\}, \tag{6}$$

the MIR cut

$$w - \frac{u}{1 - f} \leq \lfloor b \rfloor, \text{ with } f := b - \lfloor b \rfloor$$

is valid for X.

The proof introduces a simple split disjunction:

$$X^1 = X \cup \{(w,u) : w \leq \lfloor b \rfloor\} \text{ and } X^2 = X \cup \{(w,u) : w \geq \lfloor b \rfloor + 1\}.$$

Then validity of the cut for X^1 is proven by aggregating $w \leq \lfloor b \rfloor$ and $u \geq 0$ with weights 1 and $\frac{-1}{1-f}$, respectively. For X^2 the inequalities $w \geq \lfloor b \rfloor + 1$ and $w - u \leq b$ are aggregated with coefficients $-\frac{f}{1-f}$ and $\frac{1}{1-f}$, respectively.

For the multi-variable version needed in practice we have:

Theorem 6. *Given $a \in \mathbb{R}^n$, $b \in \mathbb{R}$, the single-constraint set*

$$X = \{(x, y^+, y^-) \in \mathbb{Z}_+^{|N|} \times \mathbb{R}_+ \times \mathbb{R}_+ : a^T x + y^+ \leq b + y^-\}, \tag{7}$$

and let $f = b - \lfloor b \rfloor, f_j = a_j - \lfloor a_j \rfloor$. Then the MIR cut

$$\sum_{j \in N} \Big(\lfloor a_j \rfloor + \frac{(f_j - f)^+}{1-f} \Big) x_j - \frac{y^-}{1-f} \leq \lfloor b \rfloor \tag{8}$$

is valid for X.

The proof first defines two index sets $N_1 = \{j : f_j \leq f\}$ and $N_2 = \{j : f_j > f\}$. Based on these, we define $w = \sum_{j \in N_1} \lfloor a_j \rfloor x_j + \sum_{j \in N_2} \lceil a_j \rceil x_j$, which takes the place of the single integer variable of Lemma 5. For the continuous part $u = y^- + \sum_{j \in N_2} (\lceil a_j \rceil - a_j) x_j)$ is used. Variable y^+ is discarded.

Therefore, to certify the correctness of inequality (8), we need to first print the split disjunction for w as well as a proof for $u \geq 0$ to the certificate. Then we aggregate both sides of the disjunction as pointed out in the proof of Lemma 5, unsplit both sides, and have proven validity under the restrictions of VIPR.

In practice, we perform all operations using the safe directed rounding approach discussed in the previous section. This means that we need to account for the rounding errors that were made during the solving process. A derivation for an inequality $a^T x \leq b$ is only accepted if the exact aggregation $\tilde{a}^T x \leq \tilde{b}$ *dominates* $a^T x \leq b$, meaning that $a = \tilde{a}$ and $\tilde{b} \leq b$. However, $a = \tilde{a}$ will not necessarily hold when safe rounding was used to derive a, since the coefficients were not computed in exact arithmetic.

In order to handle this without creating additional overhead for generating the certificate during the solving process, we extend the certificate language by a notion of *weak domination* as follows.

Definition 2. *An inequality $a^T x \leq b$ weakly dominates $a'^T x \leq b'$ with respect to bounds $\ell \leq x \leq u$ if there exist coefficients $\delta_i^+, \delta_i^- \geq 0$ such that*

$$a_i + \delta_i^+ - \delta_i^- = a'_i \text{ for all } i = 1, \ldots, n,$$
$$b + \sum_{i:\delta_i^+ > 0} \delta_i^+ u_i - \sum_{i:\delta_i^- > 0} \delta_i^- \ell_i \leq b'.$$

If a derived inequality is weakly dominating, then it is easy to post-process that derivation into a (strongly) dominating inequality by iterating through the variables, computing the coefficient differences $\delta_i = a'_i - a_i$, and adding them as additional aggregation multipliers with their corresponding bound constraints.

Note that the safe rounding technique uses precisely the "opposite" concept: Each arithmetic operation is accompanied by adding an overestimation of the rounding error to the right-hand side. Therefore, if we print the proof according to Theorem 6 to the certificate using safe rounding, we are guaranteed to obtain a weakly dominating inequality. Figure 2 shows a toy example of such a weak domination and its completed strong part.

To work with this extended certificate language, we extended the certificate checking software available for VIPR[2] by a newly created completion script `viprcomp` that converts certificates containing weakly dominating inequalities into classic VIPR certificate files. This has the benefit of not complicating the logic used inside the proof checker itself. Thus, confidence in the certification process remains as high as before.

Note that in theory it would be possible to perform this completion step already during the solving process, directly inside the MIP solver. However, in practice this would negate all positive performance impact from using fast safe rounding instead of expensive symbolic computations. Performing the completion *a posteriori* has further advantages. Only selected cuts must be post-processed, and we know in advance how many cuts need to be certified; these could be post-processed independently in parallel.

One issue that this approach cannot fix is the slight floating-point inaccuracy incurred from the elimination of slack variables as described at the end of Section 2.4. To illustrate this issue, consider a slack variable s_i that is present in the cut, belonging to some row $a_i^T x \leq b_i$. After safely applying the MIR procedure, that slack variable has some coefficient r_i in the resulting cut. The same procedure applied in exact arithmetic would yield a slightly different coefficient r'_i. If we could compute the difference $r'_i - r_i$

[2]The code for VIPR is open-source and available on [8].

Given			
	$x, y \in \mathbb{Z}$		
$C1:$	$3x_1 - 4x_2 \leq 2$		
$C2:$	$-x_1 + 6x_2 \leq 3$		
$B3:$	$x_2 \leq 4$		
Derived		**Reason**	**Comment**
$C4:$	$4\frac{2}{3}x_2 \leq 3\frac{2}{3}$	$\frac{1}{3} \cdot C1 + C2$	Exact derivation
$C5:$	$5x_2 \leq 5$	$\frac{1}{3} \cdot C1 + C2$	Incomplete
$C6:$	$5x_2 \leq 5$	$\frac{1}{3} \cdot C1 + C2 + \frac{1}{3} \cdot B3$	Completed version of C5

Figure 2: Toy example for weak domination. $C4$ is the actually derived constraint using the multipliers $\frac{1}{3}$, and 1. Using the same multipliers, we might have obtained (by using some rounding operations) $C5$ instead. The reasoning for $C5$ is incomplete, but it can be converted into a proper derivation by taking into account $B3$.

in the certificate completion step, we could correct for the rounding errror using the definition of $s_i = b_i - a_i^T x \geq 0$. However, inside the certificate, we can only observe the coefficients in the finalized cut, where the slack variable has been eliminated. Therefore, we cannot compute the correction factor just from the difference in coefficients since it is not possible to determine which rows contributed how much to the inaccuracy.

To overcome this issue, we need to actually compute the exact coefficients r'_i for slack variables inside the solver and print the difference $r'_i - r_i$ to the certificate. While this does increase the number of symbolic computations needed during the solve, we can limit it to only those cuts that are selected to enter the LP, and need to perform it only when certificate printing is enabled.

A different approach to verifying any kind of constraint that can be derived by aggregation is to solve an exact LP in the certificate completion step. Assume we are given an inequality $\alpha^T x \leq \beta$ that should be verified. Given the initial model constraints $Ax \leq b$, as well as additional local constraints $A^l x \leq b^l$,

$$\max\{\, \alpha^T x : Ax \leq b,\ A^l x \leq b^l \,\}.$$

If the optimal value of that exact LP is at most β, we know that $\alpha^T x \leq \beta$ is valid, and we can print a proof by aggregation using the dual multipliers of the exact LP solution. We have extended the `viprcomp` script to also contain that feature. It is not the default option to complete certificate files

for two reasons. Firstly, solving an exact LP for every cut that should be verified is computationally expensive. Secondly, it is not guaranteed that the exact LP solver can prove optimality for each cut, e.g., due to numerical troubles, which would render the whole certificate useless. The completion using only the variable bounds, on the other hand, is guaranteed to always work.

4 Computational analysis

We investigate the performance of the techniques covered in the previous sections in a practical setting. Our goal is to answer three main questions.

First, how does the separation of safe GMI cuts affect the solving behaviour? To answer this question we evaluate safe GMI cuts both with and without rounding to small denominators as proposed in Section 2.5.2. Besides comparing the number of solved instances, runtime, and number of branch-and-bound nodes, we measure the *gap closed* both after the root node and after the time limit.

Second, how does the effect of safe GMI cuts in the exact MIP framework compare to that of non-safe GMI cuts in the floating-point MIP setting? To analyze this, we first create a common baseline by disabling all features that do not exist in exact SCIP also in the floating-point version of SCIP. Within that reduced solver, we then perform experiments with and without GMI cuts and compare the results to the results with numerically safe cuts in the exact setting.

Third, what is the overhead for producing, completing, and verifying certificates? Here, we are particularly interested to see how the certification of cuts impacts the overhead beyond the pure branch-and-bound setting.

In total, we compare the following five configurations of SCIP:

- NOCUTS: Exact solving mode with separation disabled.
- SAFEGMI: Exact solving mode with separation enabled, but without rounding to small denominators according to Section 2.5.2.
- DENSMALL: Exact solving mode with separation enabled *and* size of denominators limited to 2^{17}; this value proved a good tradeoff between accuracy and speed.
- FPNOCUTS: floating-point mode with all plugins disabled that are not available in exact solving mode and no separation.

- FPGMI: floating-point mode with all plugins disabled that are not available in exact solving mode and enabled the separation of GMI cuts.

4.1 Computational setup

All experiments were performed on a cluster of Intel Xeon Gold 5122 CPUs with 3.6 GHz and 96 GB main memory. We use SoPlex 6.0.2 [31] for solving all LP and exact LP subproblems. For all symbolic computations, we use the GNU Multiple Precision Library (GMP) 6.1.2 [20]. For symbolic presolving, we use PaPILO 2.0.1 [23]; all other SCIP presolvers are disabled.

As test set we use the MIPLIB 2017 benchmark instances; in order to save computational effort, we exclude all those that could not be solved by the floating-point default version of SCIP 8.0 within two hours. We use three random seeds for the remaining 132 instances, making the size of our test set 396. The time limit was set to 7200 seconds for all experiments. Unless stated otherwise, all averages are reported as shifted geometric means with a shift of 1 seconds and 100 nodes, respectively.

4.2 Dual bound improvement

First, we analyze the *gap closed* to understand how much the added safe GMI cuts help at improving dual bounds. Given a reference bound p, the dual bound d_1 after the first LP solve, and the dual bound d_2 after the algorithm terminates, the gap closed is defined as $GC(p, d_1, d_2) = \frac{(d_2-d_1)}{(p-d_1)}$. As reference solutions, we used the best-known floating-point feasible solutions to the MIPLIB 2017 instances. We exclude all infeasible instances, as well as all instances where the floating-point and exact solutions do not agree (e.g., because the instance is infeasible in the exact setting, but not in the floating-point setting). The results for the remaining 367 instances are shown in Table 1.

We observe almost identical values for the runs with SAFEGMI and DENSMALL. DENSMALL closes on average 0.2% more gap at the root (0.152 vs. 0.150) and 0.5% more gap after the time limit (0.580 vs. 0.585). This shows that the further weakening of the cuts from that technique is not severe. Although this experiment does not present a clear picture as to which setting should be preferred, we will see a much clearer picture in favor of DENSMALL when looking at runtime experiments in Section 4.3.

When comparing the impact of GMI cuts in the exact and the floating-point setting, we first observe that there is very little difference at the root

Table 1: Arithmetic means of gap closed at the root node and after the time limit.

	NOCUTS	SAFEGMI	DENSMALL	FPNOCUTS	FPGMI
gap closed root	-	0.150	0.152	-	0.161
gap closed tlim	0.485	0.580	0.585	0.543	0.653

node: FPGMI closes only 0.9% more gap than DENSMALL (0.161 vs. 0.152). After time limit, 11% more gap is closed by enabling cuts in the floating-point setting (0.653 vs. 0.543). In the exact setting 9.5% additional gap is closed by SAFEGMI (0.580 vs. 0.485) and 10% by DENSMALL (0.585 vs. 0.485). All in all, although safely generated MIR cuts are in general weaker than floating-point cuts, our experiments show that their relative impact on the average dual performance of the respective solvers is only minimally reduced.

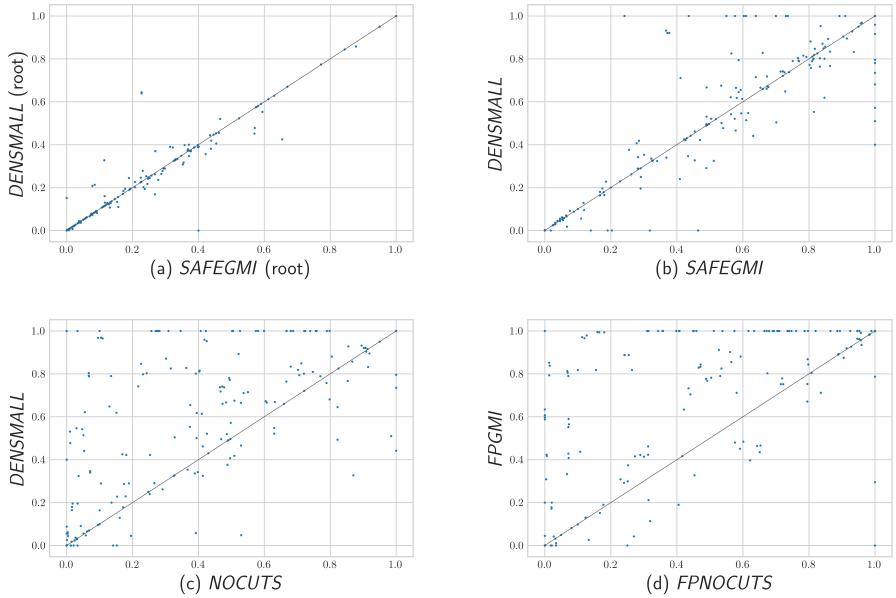
Since mean values often do not show the complete picture, Figure 3 provides pairwise comparisons between different settings.While we can also observe the very similar root node performance for SAFEGMI and DENSMALL (Figure 3a), there exists a significant amount of instances, where either DENSMALL or SAFEGMI vastly outperforms the other after time limit (Figure 3b); some of these may be due to performance variability, which is more pronounced when comparing runs involving branching. In Figure 3c it is clearly visible that safe GMI cuts help with closing the gap in exact solving mode. When comparing the impact in the exact and in the floating-point setting (Figure 3c vs. Figure 3d), we see a slightly more consistent positive impact in the floating-point setting. The number of instances where enabling cuts leads to smaller gaps closed after time limit is larger in the exact setting.

4.3 Runtime experiments

Table 2 reports aggregated results for the number of instances solved, the number of nodes, and the running times for the three exact settings. Comparing the SAFEGMI against the NOCUTS setting shows 34 new instances could be solved, while 12 could be solved with NOCUTS but not with SAFEGMI, leading to a total of 22 more instances solved.

The number of branch-and-bound nodes was reduced by 50.9%, and solving time was reduced by 18.4%. Furthermore, we observe that while the time spent in (floating-point) LP solving expectedly decreases by 33.1% due to the smaller tree sizes, the time spent in exact LP solving increases by

Figure 3: Comparison of gap closed with different versions of safe GMI cuts.



163.4%.

While the overall time spent in exact LP calls is still low, only looking at averages does not tell the full story. For many of the instances that can be solved to optimality by NOCUTS but not by SAFEGMI, the exact LP solving time is the major bottleneck. The issue that arises within the exact LP solver for some of the subproblems is that the numerical accuracy of the underlying double-precision floating-point solver is not high enough, and that after computing the residuals and rescaling inside the iterative refinement procedure, the resulting refinement LP is numerically too difficult to solve. This results either in large solving time spikes for some exact LPs or even for the exact LP solver to terminate without solving the problem to optimality.

In comparison to that, DENSMALL solves 6 more instances than SAFEGMI (10 gained, 4 lost), with a reduction in nodes by 50.5%, and a reduction in solving time by 26.8% when comparing with NOCUTS. The average time spent solving exact LPs still increases by 56.3% compared to NOCUTS, but most of the bottlenecks in exact LP solving disappear, and thus performance is improved. This is although the size of the branch-and-bound tree increases slightly compared to SAFEGMI.

Table 2: Comparison of safe cutting plane variants on all MIPLIB 2017 benchmark instances that could be solved by at least one solver. Columns exLP anf fpLP show the time spent in exact LP solving and floating-point LP solving, respectively.

				time [s]			
setting	solved	nodes	(rel)	total	(rel)	exLP	fpLP
NOCUTS	130	20106.5	1.00	1121.06	1.00	7.1	324.0
SAFEGMI	152	9867.5	0.49	914.26	0.82	18.7	216.8
DENSMALL	158	9952.9	0.5	820.69	0.73	11.1	209.5

Table 3: Impact of GMI cuts in floating-point setting on all MIPLIB 2017 benchmark instances that could be solved by at least one solver.

				time [s]		
setting	solved	nodes	(rel)	total	(rel)	LP
FPNOCUTS	164	32741.6	1.00	695.29	1.00	378.7
FPGMI	203	12604.8	0.38	451.12	0.65	233.1

This brings us to the second question, of how this measures up to the benefit from GMI cuts in the floating-point setting. The results for FPNOCUTS and FPGMI are presented in Table 3. Enabling GMI cuts in that setting solves 39 more instances, reduces the number of nodes by 61.5%, and reduces solving time by 35.1%. We see two main reasons for this slightly better performance in the floating-point case. First, the cuts are weakened by using the safe rounding procedure, both in their construction as well as during post-processing. We believe that this would not pose such a drastic effect in the floating-point setting, where tolerances are used to detect optimality and prune nodes. However, since both feasibility as well as optimality tolerance is exactly zero in the exact setting, even a slight weakening of cuts may prevent nodes from being pruned or optimality from being detected. We also have to note that performance drastically worsens if the frequency of exact LP calls is increased. In this experiment, we only solve exact LPs when strictly necessary.

4.4 Certificate overhead

We measure the overhead that is introduced by enabling the verification using VIPR certificates as described in Section 3. Since the overhead is not subject to performance variability, we only compare for a single seed, and only for our best setting, DENSMALL. To ensure that the same number of instances is solved, we extended the time limit for the run with enabled verification to 6 hours. As in [13], presolving reductions are currently not certified, so we only verify that the presolved problem has been solved correctly. However, we do check the optimal solution for feasibility in the original problem space.

SCIP does not force all generated cuts to enter the LP, but rather adds them to a storage from which efficacious cuts are selected greedily and redundant or near-orthogonal cuts are filtered. Therefore, it would be unnecessary and inefficient to write all generated cuts to the certificate file. Instead, we only print the verification of a safe GMI cut to the certificate when the corresponding row enters the LP. To make this possible we save the aggregation information, the split information, which variable bounds are used in transformation, and the scaling factor in a hashmap.

We observe an increase of 47.1% in MIP solving time, which is very similar to what was measured in [13] for the pure branch-and-bound version. This shows that the additional verification of cuts does not increase the proportional overhead for printing certificates.

The total overhead during solving, completion, and checking is on average 57.7%, which is slightly smaller than the 65.8% reported in [13], although the extra step of completing the certificates is added. This reduction shows that the effort added for verifying cutting planes is more than compensated by the effort saved due to the smaller tree sizes observed in Section 4.3.

The time spent on completing the weakly dominating inequalities is the smallest fraction of the overhead. The biggest share stems from bookkeeping and printing of certificates during the MIP solve. All except one instance that could be solved within a time limit of two hours without certification could be solved and successfully certified within 3 hours; the only outlier was instance `mas74`, which took almost 8 hours to solve and verify.

Comparing this with the overhead from other certified algorithms, e.g., in recent work on symmetry breaking for pseudo-Boolean solvers [6], shows that the behaviour is quite different. In exact MIP, the overhead for printing is much larger, while the certification overhead is much smaller. The main reason for this difference is that the logic allowed in VIPR certificates is more elementary than in [6], thus putting more strain on the solver to produce

Table 4: Overhead from producing and verifying certificates for DENSMALL on the 49 MIPLIB 2017 benchmark instances that could be solved to optimality within two hours without certification.

	time [s]			
	solving	completion	checking	total
Baseline	621.5	-	-	621.5
With verification	914.4	25.73	40.10	980.2
Overhead	47.1%	-	-	57.7%

certificates that are accepted.

5 Conclusion

In this paper, we adapt the numerically safe Gomory mixed-integer cuts introduced by Cook et al. [11] to the setting of exact rational mixed-integer programming. We identify exact LP solving as the main difficulty in making safe cutting planes performant, and show ways to overcome this difficulty by post-processing their coefficients. Using these methods, our algorithm is able to solve 21.5% more instances with a reduction of 26.8% in solving time on the MIPLIB 2017 benchmark test set. This is a significant improvement, although it is slightly less than what is achievable in the floating-point setting.

We conjecture that a slight weakening of the cuts from the numerically safe method together with the absence of error tolerances are the reasons for the smaller speedup in the exact setting.

We see several future research opportunities to further improve the performance of safe cutting planes. Extending the iterative refinement procedure of the exact LP solver to be able to perform precision-boosting would help with LPs that become difficult due to the separation of cutting planes. In a more straightforward direction, it might also be possible to tune separation parameters for the exact MIP setting, whereas in our experiments we use default settings of the floating-point solver to make the results as comparable as possible.

On the verification side, we show that the VIPR certificate format can be used to verify the correctness of the safe GMI cuts, without significantly increasing the overall certificate overhead.

All in all, we show that a careful implementation of numerically safe

cutting planes can significantly improve the solving behaviour of an exact MIP solver. It should be pointed out that this result was achieved for one of the numerically most challenging class of cutting planes: cuts generated from the simplex tableaux. Hence, we are reasonably confident that there is room for further improvement with the addition of other types of cuts that are arithmetically easier to generate and exhibit nicer numerical properties, such as cover cuts for knapsack constraints [29], flow covers [21], or zero-half cuts [26].

A Proof for intermediate fractions

We consider a rational number q, a bound on the denominator $M > 0$, its continuous fraction

$$[r_i; r_0, \ldots, r_n] = r_0 + \frac{1}{r_1 + \frac{1}{r_2 + \frac{1}{r_3 + \ldots}}},$$

convergents $(\frac{p_i}{q_i})_{i=1,\ldots,n}$, and intermediate fractions $(\frac{p_i}{q_i})^j_{i=1,\ldots,n}$. Then the following holds

Lemma 7. *Exactly the intermediate fractions with $j \geq \lfloor \frac{r_i}{2} \rfloor + 1$ are best approximations for q.*

Proof. Due to the monotonicity of intermediate fractions, we only need to prove that $\frac{p^j_{i+1}}{q^j_{i+1}}$ is a best approximation for $j = \lfloor \frac{r_i}{2} \rfloor + 1$, and that it is not a best approximation for $\lfloor \frac{r_i}{2} \rfloor$. Furthermore, we only need to prove that $\frac{p^j_{i+1}}{q^j_{i+1}}$ is a better approximation than $\frac{p_i}{q_i}$, i.e., that

$$\left| \frac{p^j_{i+1}}{q^j_{i+1}} - r \right| < \left| \frac{p_i}{q_i} - r \right|.$$

We know by ([25, Theorem 9] and [25, Theorem 13] that

$$\frac{1}{q_i(q_{i+1} + q_i)} < \left| \frac{p_i}{q_i} - r \right| \leq \frac{1}{q_i q_{i+1}}. \tag{9}$$

Since $\frac{p^j_{i+1}}{q^j_{i+1}}$ and $\frac{p_i}{q_i}$ lie on opposite sides of r, it holds that

$$\left| \frac{p^j_{i+1}}{q^j_{i+1}} - r \right| = \left| \frac{p^j_{i+1}}{q^j_{i+1}} - \frac{p_i}{q_i} \right| - \left| \frac{p_i}{q_i} - r \right|, \tag{10}$$

and

$$\left| \frac{p^j_{i+1}}{q^j_{i+1}} - \frac{p_i}{q_i} \right| = \left| \frac{q_i(jp_i + p_{i-1}) - p_i(jq_i + q_{i-1})}{q_i q^j_{i+1}} \right| = \left| \frac{q_i p_{i-1} - p_i q_{i-1}}{q_i q^j_{i+1}} \right|.$$

Iteratively applying the definitions of p_i, q_i, it is easy to see that $|q_i p_{i-1} - p_i q_{i-1}| = 1$, so we have shown

$$\left| \frac{p^j_{i+1}}{q^j_{i+1}} - \frac{p_i}{q_i} \right| = \frac{1}{q_i q^j_{i+1}} \tag{11}$$

Using this with (10) and (9) yields

$$\begin{aligned}\left|\frac{p_{i+1}^j}{q_{i+1}^j} - r\right| &< \frac{1}{q_i q_{i+1}^j} - \frac{1}{q_i(q_{i+1}+q_i)} = \frac{q_{i+1}+q_i-q_{i+1}^j}{q_i q_{i+1}^j(q_{i+1}+q_i)} \\ &= \frac{r_{i+1}q_i + q_{i-1} + q_i - (jq_i + q_{i-1})}{q_i q_{i+1}^j(q_{i+1}+q_i)} = \frac{q_{i+1}^{1+r_{i+1}-j} - q_{i-1}}{q_i q_{i+1}^j(q_{i+1}+q_i)} \\ &< \frac{q_{i+1}^{1+r_{i+1}-j}}{q_i q_{i+1}^j(q_{i+1}+q_i)}\end{aligned}$$

Setting $j = \lfloor r_{i+1}/2 \rfloor + 1$ yields

$$\begin{aligned}\left|\frac{p_{i+1}^j}{q_{i+1}^j} - r\right| &< \frac{q_{i+1}^{\lceil r_{i+1}/2 \rceil}}{q_i q_{i+1}^{\lfloor r_{i+1}/2 \rfloor + 1}(q_{i+1}+q_i)} \\ &\leq \frac{1}{q_i(q_{i+1}+q_i)} \leq \left|\frac{p_i}{q_i} - r\right|\end{aligned}$$

It still remains to show $\frac{p_{i+1}^j}{q_{i+1}^j}$ is not a best approximation for $j = \lfloor \frac{r_{i+1}}{2} \rfloor$. The idea is the same, we just use the other direction of (9).

$$\begin{aligned}\left|\frac{p_{i+1}^j}{q_{i+1}^j} - r\right| &\geq \frac{1}{q_i q_{i+1}^j} - \frac{1}{q_i q_{i+1}} = \frac{(r_{i+1}-j)q_i}{q_i q_{i+1}^j q_{i+1}} = \frac{\lceil \frac{r_{i+1}}{2} \rceil}{(\lfloor \frac{r_{i+1}}{2} \rfloor) q_{i+1} q_i + q_{i+1} q_{i-1}} \\ &\geq \frac{\lceil \frac{r_{i+1}}{2} \rceil}{(\lfloor \frac{r_{i+1}}{2} \rfloor) q_{i+1} q_i} \geq \frac{1}{q_{i+1} q_i} \geq \left|\frac{p_i}{q_i} - r\right|\end{aligned}$$

□

Acknowledgments

We would like to thank Fabian Frickenstein for his work on the VIPR checking and completion

References

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[2] T. Achterberg and R. Wunderling. Mixed integer programming: Analyzing 12 years of progress. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481, 2013.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, and K. Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

[4] J. Berg, B. Bogaerts, J. Nordström, A. Oertel, and D. Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction*, 2023.

[5] M. Bofill, F. Manyà, A. Vidal, and M. Villaret. New complexity results for Łukasiewicz logic. *Soft Computing*, 23:2187–2197, 2019.

[6] B. Bogaerts, S. Gocht, C. McCreesh, and J. Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3698–3707, Jun. 2022.

[7] B. A. Burton and M. Ozlen. Computing the crosscap number of a knot using integer programming and normal surfaces. *ACM Transactions on Mathematical Software*, 39(1), 2012.

[8] K. Cheung, A. Gleixner, and D. Steffy. VIPR. Verifying Integer Programming Results. https://github.com/ambros-gleixner/VIPR (accessed December 14, 2022).

[9] K. K. Cheung, A. Gleixner, and D. E. Steffy. Verifying Integer Programming Results. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 148–160. Springer, 2017.

[10] M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Springer Publishing Company, Incorporated, 2014.

[11] W. Cook, S. Dash, R. Fukasawa, and M. Goycoolea. Numerically safe gomory mixed-integer cuts. *INFORMS Journal on Computing*, 21:641–649, 2009.

[12] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305 – 344, 2013.

[13] L. Eifler and A. Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 2022.

[14] L. Eifler, A. Gleixner, and J. Pulaj. A safe computational framework for integer programming applied to chvátal's conjecture. *ACM Transactions on Mathematical Software*, 48(2), 2022.

[15] J. Elffers, S. Gocht, C. McCreesh, and J. Nordström. Justifying all differences using pseudo-boolean reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1486–1494, Apr. 2020.

[16] D. G. Espinoza. *On Linear Programming, Integer Programming and Cutting Planes*. PhD thesis, Georgia Institute of Technology, 2006.

[17] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. Mittelmann, D. Ozyurt, T. Ralphs, D. Salvagnin, and Y. Shinano. Miplib 2017: Data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443 – 490, 2021.

[18] A. Gleixner and D. E. Steffy. Linear programming using limited-precision oracles. *Mathematical Programming*, 183:525–554, 2020.

[19] S. Gocht. Veripb, Nov. 2019.

[20] T. Granlund and G. D. Team. *GNU MP 6.0 Multiple Precision Arithmetic Library*. Samurai Media Limited, London, GBR, 2015.

[21] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Complexity. *INFORMS J. Comput.*, 11:117–123, 1999.

[22] M. J. Heule, W. A. Hunt, and N. Wetzler. Trimming while checking clausal proofs. In *2013 Formal Methods in Computer-Aided Design*, pages 181–188, 2013.

[23] A. Hoen and L. Gottwald. scipopt/papilo: v2.0.0, Apr. 2022.

[24] F. Kenter and D. Skipper. Integer-programming bounds on pebbling numbers of cartesian-product graphs. In D. Kim, R. N. Uma, and A. Zelikovsky, editors, *Combinatorial Optimization and Applications*, pages 681–695, 2018.

[25] A. Y. Khinchin. *Continued Fractions*. Dover Books on Mathematics. Dover Publications, revised edition.

[26] A. M. Koster, A. Zymolka, and M. Kutschka. Algorithms to separate $\{0, \frac{1}{2}\}$ chvátal-gomory cuts. *Algorithmica*, 55:375–391, 2009. http://dx.doi.org/10.1007/s00453-008-9218-7.

[27] G. Lancia, E. Pippia, and F. Rinaldi. Using integer programming to search for counterexamples: A case study. In A. Kononov, M. Khachay, V. A. Kalyagin, and P. Pardalos, editors, *Mathematical Optimization Theory and Operations Research*, pages 69–84, 2020.

[28] S. Lang. *Introduction to diophantine approximations*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1966.

[29] A. N. Letchford and G. Souli. On lifted cover inequalities: A new lifting procedure with unusual properties. *Operations Research Letters*, 47(2):83–87, 2019.

[30] H. Marchand and L. Wolsey. Aggregation and mixed integer rounding to solve mips. *Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), CORE Discussion Papers*, 49, 01 1998.

[31] M. Miltenberger, A. Gleixner, T. Koch, M. Pfetsch, A. Hoen, tobiasachterberg, F. Schlösser, S. Vigerske, D. Rehfeldt, micwinx, G. Hendel, jakobwitzig, M. Besançon, D. Steffy, bzfberth, gerald gamrath, L. Gottwald, fserra, P. Wellner, A. Ebrahim, H. Mulackal, J. Nicolas-Thouvenin, stephenjmaher, and M. Walter. scipopt/soplex: v6.0.0, Apr. 2022.

[32] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Program.*, 99:283–296, 2002.

[33] M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. Society for Industrial and Applied Mathematics, 2001.

[34] J. Pulaj. Cutting planes for families implying Frankl's conjecture. *Mathematics of Computation*, 89(322):829–857, 2020.

[35] Y. Sahraoui, P. Bendotti, and C. D'Ambrosio. Real-world hydro-power unit-commitment: Dealing with numerical errors and feasibility issues. *Energy*, 184:91–104, 2019. Shaping research in gas-, heat- and electric-energy infrastructures.

[36] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., USA, 1986.

[37] D. E. Steffy and K. Wolter. Valid linear programming bounds for exact mixed-integer programming. *INFORMS Journal on Computing*, 25(2):271–284, 2013.

[38] N. Wetzler, M. Heule, and W. A. H. Jr. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.

[39] K. Wilken, J. Liu, and M. Heffernan. Optimal instruction scheduling using integer programming. *SIGPLAN Not.*, 35(5):121–133, 2000.