Certified Constraint Propagation and Dual Proof Analysis in a Numerically Exact MIP Solver

Sander Borst[1], Leon Eifler[2], and Ambros Gleixner[2,3]

[1] *Centrum Wiskunde & Informatica, The Netherlands,* sander.borst@cwi.nl
[2] *Zuse Institute Berlin, Germany,* eifler@zib.de
[3] *HTW Berlin, Germany,* gleixner@htw-berlin.de

March 20, 2024

Abstract

This paper presents the integration of constraint propagation and dual proof analysis in an exact, roundoff-error-free MIP solver. The authors employ safe rounding methods to ensure that all results remain provably correct, while sacrificing as little computational performance as possible in comparison to a pure floating-point implementation. The study also addresses the adaptation of certification techniques for correctness verification. Computational studies demonstrate the effectiveness of these techniques, showcasing a 23% performance improvement on the MIPLIB 2017 benchmark test set.

1 Introduction

Mixed integer programming (MIP) is a powerful modeling technique that can be used to solve a wide variety of complex problems. Algorithms that solve MIP problems, short MIPs, use an LP-based branch and bound algorithm at their core, but employ many complementary techniques to improve their performance [NW88, Ach07b, CCZ14, ABG+19, HP19]. They typically use double precision floating-point arithmetic together with the careful handling of numerical tolerances to quickly compute accurate solutions. While this approach makes the solving process highly efficient, and is accurate enough in practice for most applications, there exist problems where exact solutions are necessary. This is the case for applications in computational mathematics [BMVV19, BO12, EGP22, KS18, LPR20, Pul20] as well as for several industry applications where exact correctness is critical [Ach07b, WLH00, SBD19]. Furthermore, there exist pathological instances that exhibit such numerical difficulties that floating-point solvers produce large errors. For these reasons, there is a need for roundoff-error-free MIP solvers.

In recent years, a hybrid approach has been proposed that aims to take advantage of both floating-point and exact arithmetic in order to solve MIPs exactly [CKSW13]. This approach has been revised and further improved by [EG22], and more recently in [EG23] with the addition of cutting planes. What can be clearly seen in these works is that for numerical techniques to be beneficial, it is crucial to employ them as much as possible using floating-point arithmetic in a numerically safe way, as symbolic computation can be prohibitively slow.

An important feature of MIP solvers is *constraint propagation*, which is a technique that uses the problem constraints to tighten variable bounds, which can be used to detect infeasibility of a subproblem before the LP is solved. This saves time, as performing constraint propagation is generally much faster than solving an LP.

Another key feature of MIP solvers that has not yet been adapted to the exact setting is *conflict analysis*, which can be categorized in two distinct types. In this paper we will study *dual proof analysis*, which was introduced by [WBH16, WBH21]. This type of conflict analysis derives constraints from Farkas certificates of infeasible subproblems and dual solutions of bound-exceeding LPs. These are redundant constraints that do not strengthen the LP relaxation. However, by performing propagation on these constraints, infeasibility of subsequent subproblems can be detected more efficiently. Another form of conflict analysis is *graph-based* conflict analysis [Ach07a], which was inspired by a similar procedure in SAT solving [JS, MSS, MMZ$^+$]. This form of conflict analysis is numerically simpler, as it only generates disjunctive constraints with ± 1 coefficients. In this paper we study dual proof analysis, since our focus is on adapting numerical methods and studying the challenges that arise in the exact setting.

An important aspect in the context of roundoff-error-free algorithms is the *certification of their correctness*. A certifying algorithm creates a certificate alongside its solution that can be *independently verified* to be correct [MMNS11, ABM$^+$11]. Such techniques have become standard in SAT solving [WHJ14, CFHH$^+$17, CFMSSK17, GN03], and have also been adapted to SMT solving [dMB08, BRK$^+$22], pseudo boolean optimization [Goc19, EGMN20], as well as to MIP solving [CGS17, EG22, EG23].

To summarize, our contribution is to develop numerically safe versions constraint propagation and dual proof analysis, and to show that they can be used to improve the performance of an exact MIP solver in practice. Furthermore, we show how to certify the correctness of the derived bounds in the VIPR [CGS17] certificate format. Our paper is structured as follows. First, we introduce the existing methods of dual proof analysis and propagation in Section 2. Then, we describe how to adapt these techniques to the exact setting as well as how to certify their correctness in Section 3. We conduct a thorough computational study in Section 4, showing that these techniques can be used to improve the running time by 23% on the MIPLIB 2017 benchmark test set [GHG$^+$21] and solve more instances within a time limit of two hours. We conclude with an outlook on future research in Section 5.

2 Constraint Propagation and Conflict Analysis

In this paper we consider MIPs of the form

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \geq b, \\ & \ell_i \leq x_i \leq u_i \quad \text{for all } i \in [n], \\ & x_i \in \mathbb{Z} \quad \text{for all } i \in S, \end{aligned}$$

for $A \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n, b \in \mathbb{R}^m$. For the sake of simplicity of presentation, we assume that all variables have finite bounds. The techniques are still applicable to MIPs with infinite bounds, but propagation of some constraints involving unbounded variables may become ineffective and is skipped. In the following, we first review the techniques of constraint propagation and dual proof analysis in the floating-point setting.

2.1 Constraint Propagation

Constraint propagation is a technique that uses the problem constraints to tighten variable bounds, see, e.g. [Ach07b, p. 426]. Consider a constraint $a^\top x \leq b$. The maximum activity is defined as $\text{act}^+ := \sum_{i=1}^n \max(a_i\ell_i, a_iu_i)$. Similarly, the minimum activity is defined to be $\text{act}^- := \sum_{i=1}^n \min(a_i\ell_i, a_iu_i)$. We define the maximal activity relative to variable x_k as

$$\text{act}_k^+ := \sum_{i \in [n] \setminus \{k\}} \max(a_i\ell_i, a_iu_i),$$

and the minimal activity act_k^- analogously.

Any feasible vector x will satisfy $a^\top x \leq \text{act}^+$. Hence, for any variable x_i with $a_k > 0$, we have

$$x_k \geq \frac{b - \text{act}_k^+}{a_k}, \tag{1}$$

and if $a_k < 0$ we have

$$x_k \leq \frac{b - \text{act}_k^+}{a_k}. \tag{2}$$

Constraint propagation is the technique of computing the appropriate bounds from Eqs. (1) and (2) and using them to tighten one of the variable bounds ℓ_k, u_k if possible. It can be performed in an iterated fashion, as long as some of the bounds have been tightened. This is also the case at every node in a branch-and-bound tree, where the local bounds of the subproblem are taken into account after they have been tightened by branching decisions.

In addition, by computing the minimal activity of a constraint, it can sometimes be determined that a subproblem is infeasible. This happens when $\text{act}^- < b$. In this case, the problem is infeasible because the constraint is violated by any solution that respects the variable bounds. Conversely, whenever $\text{act}^+ \geq b$, then constraint is redundant and can be safely removed for the corresponding subproblem.

2.2 Dual Proof Analysis

Dual proof analysis is a technique that allows to learn from previously solved subproblems that are infeasible or exceed the objective bound. In this type of conflict analysis, for any subproblem with an infeasible or bound-exceeding LP relaxation a constraint is added to the global problem from which the infeasibility of the subproblem can be derived by applying constraint propagation to this single constraint instead of solving the LP relaxation over all constraints. Note that the constraint will be redundant, and therefore does not need to be added to the LP. However, it can be used for constraint propagation in other nodes of the tree.

If branching is only performed on variables, then such a subproblem is given as a sequence of bounds ℓ'_i, u'_i that have to be satisfied in addition to the problem constraints. So, the LP relaxation in any node in the tree will be of the form

$$\begin{aligned} \min c^\top x & \\ Ax &\geq b \\ \ell'_i \leq x_i &\leq u'_i \quad \text{for all } i \in [n]. \end{aligned}$$

The dual program to this LP is given by

$$\begin{aligned} \max\ b^\top y + \ell'^\top r^+ - u'^\top r^- & \\ r^+ - r^- - A^\top y &= -c \\ y \geq 0, r^+ \geq 0, r^- &\geq 0. \end{aligned}$$

If $\ell' \leq u'$, we can assume that for all $i \in [n]$ either r_i^+ or r_i^- is zero, so we will write $r = r^+ - r^-$. If the subproblem is infeasible, then the dual program will be unbounded. This implies that there must exist a ray y, r^+, r^- with

$$\begin{aligned} b^\top y + \ell'^\top r^+ - u'^\top r^- > 0, \\ A^\top y + r = 0. \end{aligned}$$

Such a ray is called a Farkas proof of infeasibility [Far02]. Rewriting this gives

$$0 < b^\top y + \ell'^\top r^+ - u'^\top r^- = b^\top y - \ell'^\top (A^\top y)^- + u'^\top (A^\top y)^+. \tag{3}$$

In dual proof analysis, the MIP solver adds the constraint $y^\top Ax \leq y^\top b$ to the problem. Note that this constraint is globally valid, since it is a linear combination of problem

constraints. The minimum activity of this constraint in the current subproblem is $\text{act}^- = \ell'^\top (A^\top y)^+ - u'^\top (A^\top y)^-$. Substituting this into Eq. (3) shows that $\text{act}^- > b^\top y$. Hence, for this subproblem, infeasibility can immediately be derived using propagation of the newly added constraint.

Now consider the case of a bound-exceeding subproblem. This occurs when the objective value to the LP relaxation of the current problem is higher than the objective value $c^\top x^\bullet$ of the best IP-solution $x^\bullet$ found so far. We can cut off bound-exceeding subproblems by imposing $c^\top x \leq c^\top x^\bullet$. Let y be the optimal dual solution. Dual proof analysis adds the constraint $(y^\top A - c)^\top x \geq y^\top b - c^\top x^\bullet$. This constraint is valid, as it is a linear combination of the constraints of A and $c^\top x \leq c^\top x^\bullet$. Note that for any feasible x for the subproblem, we have

$$b^\top y + \ell'^\top (A^\top y - c)^+ - u'^\top (A^\top y - c)^- = b^\top y + \ell'^\top r^+ - u'^\top r^- \geq c^\top x^\bullet, \tag{4}$$

that is

$$\ell'^\top (A^\top y - c)^+ - u'^\top (A^\top y - c)^- \geq c^\top x^\bullet - b^\top y. \tag{5}$$

Note that the left hand is the minimum activity of the newly added conflict constraint. So just like in the case of an infeasible subproblem, we see that the bound-exceeding subproblem can now be cut of using constraint propagation on the newly added constraint.

The constraints will not be added to the LP, but are only used for propagation. Since dual proof analysis might be applied many times it can get too expensive to store all the derived constraints. Therefore, we employ a method called aging, to dynamically remove constraints that have not been useful for a long time [WBH16]. In the exact setting, we make use of the same method.

3 Application in Numerically Exact MIP Solving

To efficiently implement constraint propagation and dual proof analysis in an exact MIP solver, we will perform most computations in floating-point arithmetic. To guarantee correctness, we carefully make use of directed rounding. We start with some definitions.

Let $\mathbb{F} \subseteq \mathbb{Q}$ denote the set of floating-point numbers. In practice, these will be standard IEEE double-precision [IEE08] numbers with 11 bits for the exponent and 52 bits for the mantissa. For all $x \in \mathbb{Q}$, we define:

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

Whenever we write $\overline{a + b + c + \cdots}$ we mean $\overline{\overline{a + b} + \cdots}$. An expression $\underline{a + b + \cdots}$ is defined analogously, as well as multiplication, division, and any combination thereof. We note that this is consistent with how these expressions are computed in practice.

3.1 Numerically Safe Constraint Propagation

Constraint propagation is applied at every node in the branch-and-bound tree. Recomputing the minimum and maximum activity each time would be costly. Therefore, a solver will keep track of these activities and update them whenever they change. This happens when the variable bounds change, i.e., when the bounds (ℓ, u) of variable x_i become (ℓ', u'), we update the maximum activity as

$$\text{act}^+ \leftarrow \begin{cases} (u' - u)a_i & \text{if } a_i \geq 0, \\ (\ell' - \ell)a_i & \text{otherwise.} \end{cases}$$

In the exact setting, the activities would ideally be computed in exact arithmetic, to give the tightest possible bounds. However, updating the activities using symbolic computations can be prohibitively expensive. So instead our implementation uses floating-point arithmetic. To ensure that no incorrect bounds are derived, we enforce the maximum activity that we maintain to be at least as large as the actual maximum activity. So when bounds (ℓ, u) are updated to (ℓ', u'), the maximum activity is updated to

$$\text{act}^+ \leftarrow \begin{cases} \overline{\text{act}^+ + u'a_i - ua_i} & \text{if } a_i \geq 0, \\ \underline{\text{act}^+ + \ell'a_i - \ell a_i} & \text{otherwise,} \end{cases}$$

using directed rounding. Similarly, for the minimum activity we maintain only a lower bound on the exact value.

In the floating-point setting the maintained activities need to be recomputed regularly to keep the aggregated inaccuracy from accumulating. In the exact setting this is not necessary, since the activities are guaranteed to remain valid. For this reason we do not recompute them.

3.2 Numerically Safe Dual Proof Analysis

As explained in Section 2.2, the dual ray y is computed and the constraint $y^\top Ax \leq y^\top b$ is added to the problem. For the sake of efficiency, we let the MIP solver compute y inexactly, i.e., y is obtained by a floating-point LP solve, using the standard error tolerances. This suffices, since slightly negative dual multipliers can be set to zero and any conic combination of the constraints is globally valid.

To aggregate the constraints we also use floating-point arithmetic. We start from the constraint $\hat{a}^\top x \leq \hat{b}$ for $\hat{a} = 0$ and $\hat{b} = 0$, which holds trivially. Now we add the constraints $y_j A_{\cdot j}^\top x \leq y_j b_j$ (where $A_{\cdot j}$ is the jth column of A) to this constraint one by one: For each j, we set $\hat{b} \leftarrow \overline{\hat{b} + y_j b_j}$ and the components of $\hat{a}$ are updated in the following way:

- If $u_i \leq 0$, then we set $\hat{a}_i \leftarrow \overline{\hat{a}_i + y_j A_{ji}}$.
- Otherwise, if $\ell_i \geq 0$, then we round $\hat{a}_i \leftarrow \underline{\hat{a}_i + y_j A_{ji}}$.

- Otherwise, if $u_i \leq \infty$, then we set $\hat{b} \leftarrow \overline{\hat{b} + (\overline{\hat{a}_i + y_j A_{ji}} - \underline{(\hat{a}_i + y_j A_{ji})})u_i}$ and set $\hat{a}_i \leftarrow \overline{\hat{a}_i + y_j A_{ji}}$.
- Otherwise, if $\ell_i \geq -\infty$, then we set $\hat{b} \leftarrow \overline{\hat{b} - (\overline{\hat{a}_i + y_j A_{ji}} - \underline{(\hat{a}_i + y_j A_{ji})})\ell_i}$ and set $\hat{a}_i \leftarrow \underline{\hat{a}_i + y_j A_{ji}}$.

Note that this way of rounding guarantees that the constraint stays valid, and yields an approximation of the exact conflict constraint. This approach has been used before to implement Gomory mixed integer cuts [EG23].

3.3 Producing Certificates of Correctness

MIP solvers are complex pieces of software, that might contain bugs. Hence, it can be desirable to verify the correctness of a solution that was found using a MIP solver, especially when finding exact solutions is important. While checking feasibility is easy, verifying optimality is harder, since this requires checking many nodes in the branch-and-bound tree without using the solver. To make it possible to verify optimality, a MIP solver can be extended to emit a certificate of optimality [CGS17]. The certificate contains all derived bounds, plus a reason for why each bound holds. These certificates can be verified using a simple piece of software that does not make use of the solver.

SCIP has been extended to generate certificates that can be verified using the program VIPR [CGS17]. A VIPR certificate contains all the initial problem constraints and any derived constraints. Each derived constraint contains a reason, that justifies why the constraint holds. Previously derived constraints can be referred to by their line number in the certificate. In VIPR certificates there are two deduction rules to derive an inequality:

- **Linear combination.** If a constraint is a linear combination of previously derived constraints, it has to be valid. To certify the derivation, the line numbers of these constraints are printed, along with the corresponding coefficients.
- **Rounding.** If $c^\top x \leq b$ has been previously derived such that c is zero for all indices that correspond to non-integral variables, then $\sum_i \lfloor c_i \rfloor x_i$ must be integral for any feasible solution x. Hence, $\sum_i \lfloor c_i \rfloor x_i \leq \lfloor b \rfloor$ holds. To certify the derivation, the line number of the constraint $c^\top x \leq b$ is printed to the certificate. This procedure is called a *Chvátal-Gomory cut* in the context of pure integer programming.

Additionally, there are deduction rules that allow to encode a branching proof.

To be able to verify bounds that were derived using constraint propagation or dual proof analysis, these bounds need to be written to the certificate in terms of the above derivation rules. Because Eqs. (1) and (2) are both linear combinations of valid variable bounds,

constraint propagation steps can be added to the certificate using the 'linear combination' deduction rule. If $a_k > 0$ the constraint is

$$x_k \geq \frac{1}{a_k} \left(b_j - \sum_{i \in [n] \setminus \{k\}}^{n} \max(a_i \ell_i, a_i u_i) \right). \tag{6}$$

For each $i \neq k$, the line number of $x_i \geq \ell_i$ is printed to the certificate if $a_i \ell_i > a_i u_i$. Otherwise, the line number of $x_i \leq u_i$ is printed. Finally, the line number corresponding to the constraint $\sum_{i=1}^{n} a_i x_i \geq b_j$ is printed as well. For each of the constraints, the corresponding coefficient is $1/a_k$.

If x_k is an integer variable and for all i with nonzero a_i, both a_i and x_i are integral, a rounding derivation is printed to the certificate, certifying

$$x_k \geq \left\lceil \frac{1}{a_k} (b_j - \sum_{i \in [n] \setminus \{k\}}^{n} \max(a_i \ell_i, a_i u_i)) \right\rceil. \tag{7}$$

In dual proof analysis, only linear combinations of globally valid constraints are added as constraints. However, these constraints are computed using safe rounding methods instead of exact arithmetic. For that reason, the derived constraints can not be computed directly using the given multipliers and the "linear compbination" deduction rule.

Instead, we let the solver write just the linear combination corresponding to the original constraint to the certificate along with a list of the current bounds for all variables appearing in the constraint. Then afterwards a post-processing tool, called `viprcomplete` is applied to the certificate to repair the linear combination of all of the constraints that have slight inaccuracies due to the safe rounding procedure. One of the advantages of this approach is that the repair step only needs to be done for constraints that will actually be used in the solving process. For details, we refer to [EG23].

4 Computational Study

To analyze the actual impact of constraint propagation and dual proof analysis in the context of exact MIP solving, we implemented these techniques in the exact variant of SCIP and compared them with the impact of the same techniques in the floating-point version of SCIP. Our runtime experiments do not include the time for certificate generation in order to be as comparable as possible with the floating-point version. We stress that while certificate generation does incur a computational cost, it does not change the solving path and therefore does not affect the solvability of instances. We refer to [EG22, EG23] for a detailed discussion of the computational cost of certificate generation.

4.1 Experimental Setup and Test Set

The experiments were all performed on a cluster of Intel Xeon Gold 5122 CPUs with 3.6 GHz and 96 GB main memory. For all symbolic computations, we use the GNU Multiple Precision Library (GMP) 6.1.2 [GT15]. All compared algorithms are implemented within SCIP 8.0.3 [BBC+23], using SoPlex 6.0.3 [MGK+22] as both the floating-point and the exact LP solver. For presolving in exact arithmetic, we use PAPILO 2.0.1 [Hoe22] and disable all other presolving steps. Our proposed algorithms are freely available on GitHub.[1]

We use the MIPLIB 2017 benchmark test set [GHG+21]; in order to save computational effort, we exclude all those that could not be solved by the floating-point default version of SCIP 8.0.3 within two hours. For the remaining 132 instances we use three random seeds, making the size of our test set 396. We report aggregated times in shifted geometric mean with a shift of 1 second, and node numbers in shifted geometric mean with a shift of 100 nodes. For all tests, a time limit of two hours is used.

4.2 Experimental Results

The experimental results can be found in Table 1. From the table we see that enabling propagation leads to an improvement in the running time of 11.6%. Also, 12 more instances are solved to optimality within the time limit. The number of nodes decreases by 13.6%. Enabling dual proof analysis decreases the running time by another 13.4% and the node count by 17.7%. 3 more instances are solved to optimality within the time limit. Together this comes down to a 23.4% decrease in running time, a 28.9% decrease in node count and 15 more instances that can be solved. From the performance profile in Fig. 1 it is also clear that enabling constraint propagation and conflict analysis consistently speeds up the solver. We also measured the total time that is spent in propagation. As can be seen from the table, this amount is small in comparison with the total time.

To compare this with the impact that these techniques have in the floating-point setting, we ran the same experiments on floating-point SCIP. In these experiments we disabled the features not present in the exact version of SCIP: all cutting planes, graph-based conflict analysis, restarts, presolving (except for PAPILO), custom propagation rules. As shown in Table 2 in the floating-point setting, enabling propagation reduces solving time by 12.7%. The node count increases by 6.2% and 11 more instances are solved to optimality. Dual proof analysis reduces the running time by another 36.5% and the node count by 67.1%. 20 more instances are solved to optimality. Together this comes down to a decrease in the running time of 44.5%, a 65.1% decrease in node count and 31 more instances being solved to optimality.

It is clear that the performance boost is significantly larger in the floating-point setting. The same is true for the increase in the number of instances solved. While the speedup

[1] As part of the development version of exact SCIP mirrored under `https://github.com/scipopt/scip/tree/exact-rational`.

Settings	# Solved	Time				Nodes	(rel)
		Total	(rel)	CP	DPA		
Baseline	135	759.21	—	—	—	8735.1	—
+ CP	147	671.22	(0.88)	9.11	—	7550.4	(0.86)
+ CP + DPA	150	581.46	(0.77)	9.22	5.44	6211.1	(0.71)

Table 1: Experimental results comparing the performance of exact SCIP with and without constraint propagation (CP) and dual proof analysis (DPA) enabled. All times shown are in seconds. For the times and node counts, the shifted geometric with shift 1 seconds or 100 nodes respectively is used. Columns (rel) show times and nodes relative to the baseline. Only the instances that could be solved by at least one of the given configurations are included in these statistics.

Settings	# Solved	Time	(rel)	Nodes	(rel)
Baseline	164	584.84	—	11415.0	—
+ CP	175	510.85	(0.87)	12119.4	(1.06)
+ CP + DPA	195	324.41	(0.55)	6986.1	(0.61)

Table 2: Experimental results comparing the performance of floating-point SCIP with and without constraint propagation (CP) and dual proof analysis (DPA) enabled. All times shown are in seconds. For the times and node counts, the shifted geometric with shift 1 seconds or 100 nodes respectively is used. Columns (rel) show times and nodes relative to the baseline. Only the instances that could be solved by at least one of the given configurations are included in these statistics.

from only enabling propagation is comparable, we observe a much greater reduction in the number of nodes, and a much larger speedup from conflict analysis in the floating-point setting. We see several reasons contributing to this difference.

First, performance variability plays a part and the fact that we look at different subsets of the instances. Second, propagation takes up more time in the exact solving mode. Although the shifted geometric mean of the propagation times shown in Table 1 is not large, there exist instances where exact propagation even takes up a major portion of the solving time. This is the case for instances in which propagation leads to a large number of bound changes, since applying these bound changes is computationally more expensive in the exact setting. This is because, while the propagation procedure itself is implemented using floating-point arithmetic, applying a bound change in exact SCIP is currently done using exact arithmetic. This means every floating-point bound change needs to be converted to a rational number first, and then that rational number needs to

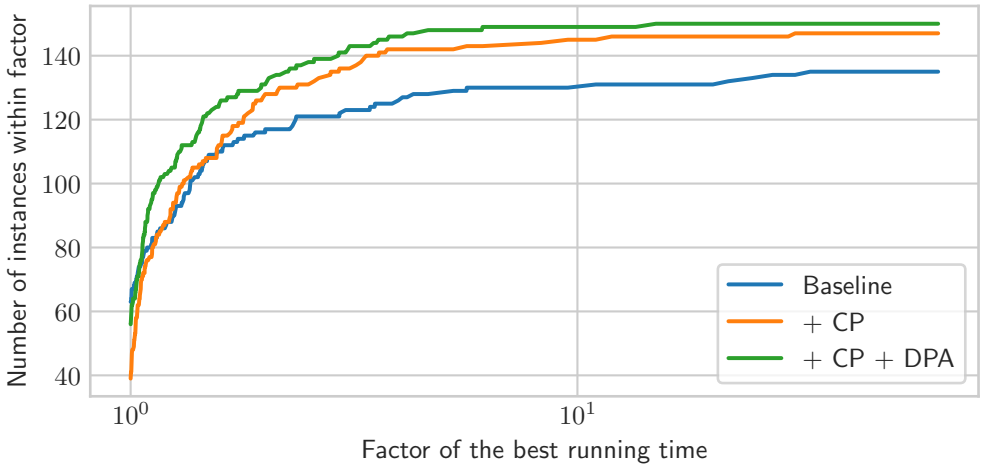

Figure 1: Performance profile for different configurations. For each configuration it is shown how many instances achieved a running time within a given factor of the shortest running time for that instances. Only the instances that could be solved by at least one of the given configurations are included in these statistics.

be used to update the exact bound. This can slow down the process significantly. If we only look at the solving time without the time spent in propagation, the speedup from enabling propagation is 17.1%, and the speedup from enabling conflict analysis is 28.3%.

There are more reasons that might lead to a lower impact of dual proof analysis in the exact setting. The most important one is that constraint propagation is performed using floating-point arithmetic, and that we cannot determine infeasibility within tolerances in the exact setting. This might sometimes lead to derived bounds not being precise enough to be useful in the exact settings or conflicts not being able to prove infeasibility. For example, it is possible that in floating-point mode such an inexact bound can be used to decide the infeasibility of a node, whereas in exact mode the corresponding LP still needs to be solved. In numerically difficult cases, this might even lead to incorrect cutoffs in the floating-point setting. It is clear that this is a trade-off that we can never fully avoid, guaranteeing correctness while sacrificing performance.

Another possible reason that the techniques are slightly less effective in exact SCIP could be that the derived variable bounds on non-integral variables can have a large denominator. This can slow down solving LPs exactly, as has been observed for safe cutting plane generation in [EG23]. We experimented with two variants that try to prevent this from happening:

1. Limit the size of the denominators of the derived variable bounds for continuous variables to a fixed upper bound.

2. Disable constraint propagation for continuous variables.

However, we did not observe any positive effects on the solving times. This does not mean that the described negative effects do not occur, but rather that all in all, by enabling propagation for non-integral variables and allowing arbitrary denominators in their bounds, we gain more than we lose.

5 Conclusion and Outlook

In this study, we investigated the feasibility and impact of constraint propagation and dual proof analysis in the exact MIP solver SCIP. We found that enabling both techniques decreases the running time by 23.4% and the number of nodes by 28.9%. This performance boost is significant, but less than the 44.5% decrease in running time and 65.1% decrease in node count that we observed in the equivalent floating-point setting. Such weaker performance is partially the price to be paid for exactness, coming from weaker inequalities due to directed rounding and the fact that we cannot determine infeasibility within tolerances in the exact setting.

Still, we see several future research opportunities to further improve these results. On the implementation side, we believe there is potential to decrease the time spent in propagation by adding support for floating-point bound changes in exact SCIP. Also, due to technical reasons, it is currently not possible to apply propagation within strong branching in the exact solving mode. Enabling this could also be beneficial.

Algorithmically, incorporating the strengthening techniques from [WBH21] for conflict constraints also in the exact setting is likely to yield additional performance improvements. Finally, implementing graph-based conflict analysis [Ach07a] would be straightforward due to its combinatorial nature and is sure to yield positive impact in the exact setting. More open-ended is the question if sporadically recomputing the activities also in the exact setting could lead to tighter bounds and thus better performance.

References

[ABG+19] Tobias Achterberg, Robert Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32:473–506, 2019.

[ABM+11] Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology*, 53(6):287–293, 2011.

[Ach07a] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, March 2007.

[Ach07b] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[BBC+23] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chimela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Gregor Hendel, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. scipopt/scip: v8.0.3, July 2023.

[BMVV19] Miquel Bofill, Felip Manyà, Amanda Vidal, and Mateu Villaret. New complexity results for Łukasiewicz logic. *Soft Computing*, 23:2187–2197, 2019.

[BO12] Benjamin A. Burton and Melih Ozlen. Computing the crosscap number of a knot using integer programming and normal surfaces. *ACM Transactions on Mathematical Software*, 39(1), 2012.

[BRK+22] Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark Barrett. Flexible proof production in an industrial-strength SMT solver. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning*, pages 15–35, Cham, 2022. Springer.

[CCZ14] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer Programming*. Springer, 2014.

[CFHH+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.

[CFMSSK17] Luís Cruz-Filipe, Joao Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings, Part I, of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 10205*, page 118–135, Berlin, Heidelberg, 2017. Springer-Verlag.

[CGS17] Kevin K. H. Cheung, Ambros Gleixner, and Daniel E. Steffy. Verifying Integer Programming Results. volume 10328, pages 148–160. 2017.

[CKSW13] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, September 2013.

[dMB08] Leonardo Mendonça de Moura and Nikolaj S Bjørner. Proofs and refutations, and Z3. In *LPAR Workshops*, volume 418, pages 123–132. Doha, Qatar, 2008.

[EG22] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, January 2022.

[EG23] Leon Eifler and Ambros Gleixner. Safe and verified gomory mixed integer cuts in a rational MIP framework. *SIAM Journal on Optimization*, 2023. accepted for publication.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-boolean reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1486–1494, 2020.

[EGP22] Leon Eifler, Ambros Gleixner, and Jonad Pulaj. A safe computational framework for integer programming applied to chvátal's conjecture. *ACM Transactions on Mathematical Software*, 48(2), 2022.

[Far02] Julius Farkas. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1902(124):1–27, January 1902.

[GHG+21] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021.

[GN03] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, page 10886, USA, 2003. IEEE Computer Society.

[Goc19] Stephan Gocht. VeriPB. 10.5281/zenodo.3548582, 2019. version 0.1.0.

[GT15] Torbjrn Granlund and Gmp Development Team. *GNU MP 6.0 Multiple Precision Arithmetic Library*. Samurai Media Limited, London, GBR, 2015.

[Hoe22] Alexander Hoen. scipopt/papilo: v2.0.0, April 2022.

[HP19] Christopher Hojny and Marc E. Pfetsch. Polytopes associated with symmetry handling. *Mathematical Programming*, 175:197–240, 2019.

[IEE08] IEEE standard for floating-point arithmetic. Standard IEEE Std 754-2008, IEEE Computer Society, New York, NY, USA, August 2008.

[JS] Roberto J Bayardo Jr and Robert C Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances.

[KS18] Franklin Kenter and Daphne Skipper. Integer-programming bounds on pebbling numbers of cartesian-product graphs. In Donghyun Kim, R. N. Uma, and Alexander Zelikovsky, editors, *Combinatorial Optimization and Applications*, pages 681–695, 2018.

[LPR20] Giuseppe Lancia, Eleonora Pippia, and Franca Rinaldi. Using integer programming to search for counterexamples: A case study. In Alexander Kononov, Michael Khachay, Valery A Kalyagin, and Panos Pardalos, editors, *Mathematical Optimization Theory and Operations Research*, pages 69–84, 2020.

[MGK+22] Matthias Miltenberger, Ambros Gleixner, Thorsten Koch, Marc Pfetsch, Alexander Hoen, Tobias Achterberg, Franziska Schlösser, Stefan Vigerske, Daniel Rehfeldt, Michael Winkler, Gregor Hendel, Jakob Witzig, Mathieu Besançon, Dan Steffy, Timo Berthold, Gerald Gamrath, Leona Gottwald, Felipe Serrano, Philipp Wellner, Ali Ebrahim, Harikrishnan Mulackal, Jules Nicolas-Thouvenin, Stephen J Maher, and Matthias Walter. scipopt/soplex: v6.0.0, April 2022.

[MMNS11] Ross M McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Survey: Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[MMZ+] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver.

[MSS] J.P. Marques-Silva and K.A. Sakallah. GRASP: A search algorithm for propositional satisfiability. 48(5):506–521.

[NW88] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Ltd, 1988.

[Pul20] Jonad Pulaj. Cutting planes for families implying Frankl's conjecture. *Mathematics of Computation*, 89(322):829–857, 2020.

[SBD19] Youcef Sahraoui, Pascale Bendotti, and Claudia 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.

[WBH16] Jakob Witzig, Timo Berthold, and Stefan Heinz. Experiments with Conflict Analysis in Mixed Integer Programming, November 2016.

[WBH21] Jakob Witzig, Timo Berthold, and Stefan Heinz. Computational aspects of infeasibility analysis in mixed integer programming. *Mathematical Programming Computation*, March 2021.

[WHJ14] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe 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.

[WLH00] Kent Wilken, Jack Liu, and Mark Heffernan. Optimal instruction scheduling using integer programming. *SIGPLAN Not.*, 35(5):121–133, 2000.