Certifying MIP-Based Presolve Reductions for 0–1 Integer Linear Programs

Alexander Hoen[1](✉), Andy Oertel[3,4], Ambros Gleixner[1,2], and Jakob Nordström[3,4]

[1] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
hoen@zib.de
[2] HTW Berlin, 10313 Berlin, Germany
gleixner@htw-berlin.de
[3] Lund University, Lund, Sweden
andy.oertel@cs.lth.se
[4] University of Copenhagen, Copenhagen, Denmark
jn@di.ku.dk

Abstract. It is well known that reformulating the original problem can be crucial for the performance of mixed-integer programming (MIP) solvers. To ensure correctness, all transformations must preserve the feasibility status and optimal value of the problem, but there is currently no established methodology to express and verify the equivalence of two mixed-integer programs. In this work, we take a first step in this direction by showing how the correctness of MIP presolve reductions on 0–1 integer linear programs can be certified by using (and suitably extending) the VeriPB tool for pseudo-Boolean proof logging. Our experimental evaluation on both decision and optimization instances demonstrates the computational viability of the approach and leads to suggestions for future revisions of the proof format that will help to reduce the verbosity of the certificates and to accelerate the certification and verification process further.

Keywords: Proof logging · Presolving · 0–1 integer linear programming

1 Introduction

Boolean satisfiability solving (SAT) and *mixed-integer programming (MIP)* are two computational paradigms in which surprisingly mature and powerful solvers have been developed over the last decades. Today such solvers are routinely used to solve large-scale problems in practice despite the fact that these problems are *NP*-hard. Both SAT and MIP solvers typically start by trying to simplify the input problem before feeding it to the main solver algorithm, a process known as *presolving* in MIP and *preprocessing* in SAT. This can involve, e.g., fixing

B. Dilkina (Ed.): CPAIOR 2024, LNCS 14742, pp. 310–328, 2024.
https://doi.org/10.1007/978-3-031-60597-0_20

variables to values, strengthening constraints, removing constraints, or adding new constraints to break symmetries. Such techniques are very important for SAT solver performance [6], and for MIP solvers they often play a decisive role in whether a problem instance can be solved or not, regardless of whether the solver uses floating-point [2] or exact rational arithmetic [21].

The impressive performance gains for modern combinatorial solvers come at the price of ever-increasing complexity, which makes these tools very hard to debug. It is well documented that even state-of-the-art solvers in many paradigms, not just SAT and MIP, suffer from errors such as mistakenly claiming infeasibility or optimality, or even returning "solutions" that are infeasible [3,14,27,40,49]. During the last decade, the SAT community has dealt with this problem in a remarkably successful way by requiring that solvers should use *proof logging*, i.e., produce machine-verifiable certificates of correctness for their computations that can be verified by a stand-alone proof checker. A number of proof formats have been developed, such as DRAT [36,37,52], GRIT [18], and LRAT [17], which are used to certify the whole solving process including preprocessing.

Achieving something similar in a general MIP setting is much more challenging, amongst others because of the presence of continuous and general integer variables, which may even have unbounded domains. For numerically exact MIP solvers [15,21,22] the proof format VIPR [12] has been introduced, but it currently only allows verification of feasibility-based reasoning, which must preserve all feasible solutions. In particular, it does not support the verification of dual presolving techniques that may exclude feasible solutions as long as one optimal solution remains. This means that while exact MIP solvers could in principle generate a certificate for the main solving process, such a certificate would only establish correctness under the assumption that all the presolving steps were valid, as, e.g., in [21]. And, unfortunately, the proof logging techniques for SAT preprocessing cannot be used to address this problem, since they can only reason about clausal constraints.

Our contribution in this work is to take a first step towards verification of the full MIP solving process by demonstrating how pseudo-Boolean proof logging with VeriPB can be used to produce certificates of correctness for a wide range of MIP presolving techniques for 0–1 integer linear programs (ILPs). VeriPB is quite a versatile tool in that it has previously been employed for certification of, e.g., advanced SAT solving techniques [8,35], SAT-based optimization (MaxSAT) [4,50], subgraph solving [32,33], and constraint programming [34,42]. However, to the best of our knowledge this is the first time the tool has been used to prove the correctness of reformulations of optimization problems, and this presents new challenges. In particular, the proof system turns out not to be well suited for problem reformulations with frequent changes to the objective function, and therefore we introduce a new rule for objective function updates.

Our computational experiments confirm that this approach to certifying presolve reductions is computationally viable and the overhead for certification aligns with what is known from the literature for certifying problem transformations in other contexts [31]. The analysis of the results reveals new insights into performance bottlenecks, and these insights directly translate to possible revisions of the proof logging format that would be valuable to address in order to decrease the size of the generated proofs and speed up proof verification.

We would like to note that, while our current methods are only applicable to 0–1 ILPs, this covers already a large and important class of MIPs. In particular, there are applications where the exact and verified solution of 0–1 ILPs is highly relevant, see [1,23,46] for some examples.

The rest of this paper is organized as follows. After presenting pseudo-Boolean proof logging and VeriPB in Sect. 2, we demonstrate in Sect. 3 how to produce VeriPB certificates for MIP presolving on 0–1 ILPs. In Sect. 4 we report results of an experimental evaluation, and we conclude in Sect. 5 with a summary and discussion of future work.

2 Pseudo-Boolean Proof Logging with VeriPB

We start by reviewing pseudo-Boolean reasoning in Sect. 2.1, and then explain our extension to deal with objective function updates in Sect. 2.2. In order to make the concept of proof logging more concrete, we conclude this section by providing, in Table 1, a few examples of how the derivation rules explained below are encoded in VeriPB syntax. For space reasons, this list does not include examples of subproofs that may be necessary for some derivations that cannot be proven automatically by VeriPB. Further details on practical aspects and implementation of pseudo-Boolean proof logging can be found in the software repository of VeriPB [30].

2.1 Pseudo-Boolean Reasoning with the Cutting Planes Method

Our treatment of this material will by necessity be somewhat terse—we refer the reader to [9] for more information about the cutting planes method and to [8,31] for detailed information about the VeriPB proof system and format.

We write x to denote a $\{0,1\}$-valued variable and $\overline{x}$ as a shorthand for $1-x$, and write ℓ to denote such *positive* and *negative literals*, respectively. By a *pseudo-Boolean (PB) constraint* we mean a 0–1 linear inequality $\sum_j a_j \ell_j \geq b$, where when convenient we can assume all literals ℓ_j to refer to distinct variables and all a_j and b to be non-negative (so-called *normalized form*). A *pseudo-Boolean formula* is just another name for a 0–1 integer linear program. For optimization problems we also have an objective function $f = \sum_j c_j x_j$ that should be minimized (and f can be negated to represent a maximization problem).

The foundation of VeriPB is the *cutting planes* proof system [13]. At the start of the proof, the set of *core constraints* $\mathcal{C}$ are initialized as the 0–1 linear inequalities in the problem instance. Any constraints derived as described below are placed in the set of *derived constraints* $\mathcal{D}$, from where they can later be moved to $\mathcal{C}$ (but not vice versa). Loosely speaking, VeriPB proofs maintain the invariant that the optimal value of any solution to $\mathcal{C}$ and to the original input problem is the same. New constraints can be derived from $\mathcal{C} \cup \mathcal{D}$ by performing *addition* of two constraints or *multiplication* of a constraint by a positive integer, and *literal axioms* $\ell \geq 0$ can be used at any time. Additionally, for a constraint $\sum_j a_j \ell_j \geq b$ written in normalized form we can apply *division* by a positive integer d followed by rounding up to obtain $\sum_j \lceil a_j/d \rceil \ell_j \geq \lceil b/d \rceil$, and *saturation* can be applied to yield $\sum_j \min\{a_j, b\} \cdot \ell_j \geq b$.

For a PB constraint $C \doteq \sum_j a_j \ell_j \geq b$ (where we use $\doteq$ to denote syntactic equality), the negation of C is $\neg C \doteq \sum_j a_j \ell_j \leq b-1$. For a *partial assignment* ρ mapping variables to $\{0, 1\}$, we write $C\restriction_\rho$ for the *restricted constraint* obtained by replacing variables in C assigned by ρ by their values and simplifying the result. We say that C *unit propagates* ℓ *under* ρ if $C\restriction_\rho$ cannot be satisfied unless ℓ is assigned to 1. If unit propagation on all constraints in $\mathcal{C} \cup \mathcal{D} \cup \{\neg C\}$ starting with the empty assignment $\rho = \emptyset$, and extending ρ with new assignments as long as new literals propagate, leads to contradiction in the form of a violated constraint, then we say that C follows by *reverse unit propagation (RUP)* from $\mathcal{C} \cup \mathcal{D}$. Such (efficiently verifiable) RUP steps are allowed in VeriPB proofs when it is convenient to avoid writing out an explicit derivation of C from $\mathcal{C} \cup \mathcal{D}$. We will also write $C\restriction_\omega$ to denote the result of applying to C a *(partial) substitution* ω which can remap variables to other literals in addition to 0 and 1, and we extend this notation to sets in the obvious way by taking unions.

In addition to the cutting planes rules, which can only derive semantically implied constraints, VeriPB has a *redundance-based strengthening rule* that can derive a non-implied constraint C as long as this does not change the feasibility or optimal value of the problem. Formally, C can be derived from $\mathcal{C} \cup \mathcal{D}$ using this rule by exhibiting in the proof a *witness substitution* ω together with subproofs

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f \geq f\restriction_\omega\}, \tag{1}$$

of all constraints on the right-hand side from the premises on the left-hand side using the derivation rules above. Intuitively, what (1) shows is that if α is any assignment that satisfies $\mathcal{C} \cup \mathcal{D}$ but violates C, then $\alpha \circ \omega$ satisfies $\mathcal{C} \cup \mathcal{D} \cup \{C\}$ and yields at least as good a value for the objective function f.

During presolving, constraints in the input formula can be deleted or replaced by other constraints, and the proof needs to establish that such modifications are correct. While deletions from the derived set $\mathcal{D}$ are always in order, removing a constraint from the core set $\mathcal{C}$ could potentially introduce spurious solutions. Therefore, deleting a constraint C from $\mathcal{C}$ can only be done by the *checked deletion rule*, which requires to show that C could be rederived from $\mathcal{C} \setminus \{C\}$ by redundance-based strengthening (see [8] for a more detailed explanation).

2.2 A New Rule for Objective Function Updates

When variables are fixed or identified during the presolving process, the objective function f can be modified to a function f'. This modified objective f' can then be used in other presolver reasoning. This scenario arises also in, e.g., MaxSAT solving, and can be dealt with by deriving two PB constraints $f \geq f'$ and $f' \geq f$ in the proof, which encodes that the old and new objective are equal [4].

Table 1. Examples of basic derivation rules in VeriPB syntax. Here, (id) refers to the constraint ID assigned by VeriPB.

Rule	Syntax	Explanation
cutting planes in reverse Polish notation	`pol x1 4 +`	add $x_1 \geq 0$ and (4)
	`pol 3 2 d`	divides (3) by 2
	`pol 1 2 * ~x1 +`	multiplies (1) by 2 and adds $\overline{x}_1 \geq 0$
redundance-based strengthening	`red +1 x1 >= 1; x1 1`	verifies $x_1 \geq 1$ with $\omega = \{x_1 \mapsto 1\}$
	`red +1 x1 +1 x2 >= 1; x1 x2 x2 x1`	verifies $x_1 + x_2 \geq 1$ with $\omega = \{x_1 \mapsto x_2, x_2 \mapsto x_1\}$
RUP	`rup +1 x1 +1 x2 >= 1;`	verifies $x_1 + x_2 \geq 1$ with RUP
move to core	`core id 3`	moves (3) to the core constraints
deletion from core	`delc 3`	deletes (3) from the core constraints
objective function update	`obju new +1 x1 +1 x2 1;`	defines $x_1 + x_2 + 1$ as new objective
	`obju diff +1  x1;`	adds $\overline{x}_1$ to the objective

Whenever the solver argues in terms of f', a telescoping-sum argument with $f' = f$ can be used to justify the same conclusion in terms of the old objective.

However, if the presolver changes f to f' and then uses reasoning that needs to be certified by redundance-based strengthening, then tricky problems can arise. One of the required proof goals in (1) is that the witness ω cannot worsen the objective. If ω does not mention variables in f', then this is obvious to the presolver—ω has no effect on the objective—but if ω assigns variables in the original objective f, then one still needs to derive $f \geq f\restriction_\omega$ in the formal proof, which can be challenging. While this can often be done by enlarging the witness ω to include earlier variable fixings and identifications, the extra bookkeeping required for this quickly becomes a major headache, and results in the proof deviating further and further from the actual presolver reasoning that the proof logging is meant to certify.

For this reason, a better solution is to introduce a new *objective function update rule* that formally replaces f by a new objective f', so that all future reasoning about the objective can focus on f' and ignore f. Such a rule needs to be designed with care, so that the optimal value of the problem is preserved. Due to space constraints we cannot provide a formal proof here, but recall that intuitively we maintain the invariant for the core set $\mathcal{C}$ that it has the same optimal value as the original problem. In agreement with this, the formal requirement for updating the objective from f to f' is to present in the proof log derivations of the two constraints $f \geq f'$ and $f' \geq f$ from the core set $\mathcal{C}$ only.

3 Certifying Presolve Reductions

We now describe how feasibility- and optimality-based presolving reductions can be certified by using VeriPB proof logging enhanced with the new objective function update rule described in Sect. 2.2 above. We distinguish between *primal* and *dual* reductions, where primal reductions strengthen the problem formulation by tightening the convex hull of the problem and preserve all feasible solutions, and dual reductions may additionally remove feasible solutions using optimality-based arguments. More precisely, *weak* dual reductions preserve all optimal solutions, but may remove suboptimal solutions. *Strong* dual reductions may remove also optimal solutions as long as at least one optimal solution is preserved in the reduced problem. Our selection of methods is motivated by the recent MIP solver implementation described in [44]. Before explaining the individual presolving techniques and their certification, we introduce a few general techniques that are needed for the certification of several presolving methods.

3.1 General Techniques

Substitution. In order to reduce the number of variables, constraints, and non-zero coefficients in the constraints, many presolving techniques first try to identify an equality $E \doteq x_k = \sum_{j \neq k} \alpha_j x_j + \beta$ with $\alpha_j, \beta \in \mathbb{Q}$. Subsequently, all occurrences of x_k in the objective and constraints besides E are substituted by the affine expression on the right-hand side and x_k is removed from the problem. The simplest case when x_k is fixed to zero or one, i.e., when $\beta \in \{0, 1\}$ and all $\alpha_j = 0$, is straightforward to handle by deriving a new lower or upper bound on x_k. During presolving, every fixed variable is removed from the model. In the cases where some $\alpha_j \neq 0$, first the equation is expressed as a pair of constraints $E_{\geq} \wedge E_{\leq}$ and then the variable is removed by aggregation as follows.

Aggregation. In order to substitute variables or reduce the number of non-zero coefficients, certain presolving techniques add a scaled equality $s \cdot E \doteq s \cdot E_{\geq} \wedge s \cdot E_{\leq}$, $s \in \mathbb{Q}$, to a given constraint D. We call this an *aggregation*. Since VeriPB certificates expect inequalities with integer coefficients, s is split into two integer scaling factors $s_E, s_D \in \mathbb{Z}$ with $s = s_E / s_D$. In the certificate, the aggregation is expressed as a newly derived constraint

$$D_{new} \doteq \begin{cases} |s_E| \cdot E_{\geq} + |s_D| \cdot D & \text{if } \frac{s_D}{s_E} > 0 \\ |s_E| \cdot E_{\leq} + |s_D| \cdot D & \text{otherwise} \end{cases} .$$

Note that the presolving algorithm may decide to keep working with the constraint $(1/s_D) D_{new}$ internally. In this case, it must store the scaling factor s_D in order to correctly translate between its own state and the state in the certificate; this happens in the implementation used in Sect. 4.

Checked Deletion. The derivation of a new constraint D_{new} can render a previous constraint D redundant. A typical example is the case of substituting a variable above. In a (pre)solver, the previous constraint is overwritten, and in order to keep the constraint database in the proof aligned with the solver, one may want to delete the previous constraint from the proof. In order to check the deletion of D, a subproof is required that proves its redundancy. In most cases, this subproof contains the "inverted" derivation of D_{new}. As an example, consider an aggregation $D_{new} \doteq D + E_{\leq}$ with an equality $E \doteq E_{\leq} \wedge E_{\geq}$. In this case, the subproof for the checked deletion is $D_{new} + E_{\geq}$. Unless stated otherwise, the new constraints are moved to the core and redundant constraints are always removed by inverting the derivation of the constraint that replaces them.

3.2 Primal Reductions

Primal reductions can be certified purely by implicational reasoning.

Bound Strengthening. This preprocessor [24,47] tries to tighten the variable domains by iteratively applying well-known *constraint propagation* to all variables in the linear constraints. Each reduced variable domain is communicated to the affected constraints and may trigger further domain changes. This process is continued until no further domain reductions happen or the problem becomes infeasible due to empty domains. Specifically, for an inequality constraint

$$\sum_{j \in N} a_j x_j \geq b \tag{2}$$

with $a_k \neq 0$, we first underestimate $a_k x_k$ via

$$a_k x_k \geq b - \sum_{j \neq k} a_j x_j \geq b - \sum_{j \neq k, a_j > 0} a_j \, .$$

If $a_k > 0$, this yields the lower bound

$$x_k \geq \left\lceil (b - \sum_{j \neq k, a_j > 0} a_j)/a_k \right\rceil , \tag{3}$$

and if $a_k < 0$ we can obtain an analogous upper bound on x_k.

The bound change can be proven either by RUP, or more explicitly by stating the additions and division needed to form (3) from (2) and the bound constraints. We analyze the effect of both variants in Sect. 4.4.

Parallel Rows. Two constraints C_j and C_k are parallel if a scalar $\lambda \in \mathbb{R}^+$ exists with $\lambda(a_{j1}, \dots, a_{jn}, b_j) = (a_{k1}, \dots, a_{kn}, b_k)$. Hence, one of these constraints is redundant and can be removed from the model [2,26]. The subproof for deleting the redundant rows must contain the remaining parallel row and λ to prove the redundancy. For a fractional λ the two constraint are scaled to ensure integer coefficients in the certificate.

Probing. The general idea of *probing* [1,47] is to tentatively fix a variable x_j to 0 or 1 and then apply constraint propagation to the resulting model. Suppose x_k is an arbitrary variable with $k \neq i$, then we can learn fixings or implications in the following cases:

1. If $x_j = 0$ implies $x_k = 1$ and $x_j = 1$ implies $x_k = 0$ we can add the constraint $x_j = 1 - x_k$. Analogously, we can derive $x_k = x_j$ in the case that $x_j = 0$ implies $x_k = 0$ and $x_j = 1$ implies $x_k = 1$.
2. If $x_j = 0$ propagates to infeasibility we can fix $x_j = 1$. Analogously, if $x_j = 1$ propagates to infeasibility we can fix $x_j = 0$.
3. If $x_j = 0$ implies $x_k = 0$ and $x_j = 1$ implies $x_k = 0$ we can fix x_k to 0. Analogously, x_k can be fixed to 1 if $x_j = 0$ implies $x_k = 1$ and $x_j = 1$ implies $x_k = 1$.

Cases 1 and 2 can be proven with RUP. To prove correctness of fixing $x_k = 1$ in Case 3 we first derive two new constraints $x_k + x_j \geq 1$ and $x_k - x_j \geq 0$ in the proof log by RUP. Adding these two constraints leads to $x_k \geq 1$. To prove $x_k = 0$ we derive the constraints $x_k + x_j \leq 0$ and $x_k - x_j \leq 0$ leading to $x_k = 0$.

Simple Probing. On equalities with a special structure, a more simplified version of probing called *simple probing* [2, Sect. 3.6] can be applied. Suppose the equation

$$\sum_{j \in N} a_j x_j = b \text{ with } \sum_{j \in N} a_j = 2 \cdot b \text{ and } |a_k| = \sum_{j \in N, a_j > 0} a_j - b$$

holds for a variable x_k with $a_k \neq 0$. Let $\hat{N} = \{p \in N \mid a_p \neq 0\}$. Under these conditions, $x_k = 1$ implies $x_p = 0$ and $x_k = 0$ implies $x_p = 1$ for all $p \in \hat{N}$ with $a_p > 0$. Further, $x_k = 1$ implies $x_p = 1$ and $x_k = 0$ implies $x_p = 0$ for all $p \in \hat{N}$ with $a_p < 0$. These implications can be expressed by the constraints

$$x_k = 1 - x_p \text{ for all } p \in \hat{N} \text{ with } a_p > 0\,, \tag{4}$$

$$x_k = x_p \text{ for all } p \in \hat{N} \text{ with } a_p < 0\,. \tag{5}$$

The constraints (4) and (5) can be proven with RUP and used to substitute variables x_p for all $p \in \hat{N}$ from the problem.

Sparsifying the Matrix. The presolving technique sparsify [2,11] tries to reduce the number of non-zero coefficients by adding (multiples of) equalities to other constraints using aggregations. This can be certified as described in Sect. 3.1.

Coefficient Tightening. The goal of this MIP presolving technique, which goes back to [47], is to tighten the LP relaxation, i.e., the relaxation obtained when the integrality requirements are replaced by $x_j \in [0, 1]$. To this end, the coefficients of constraints are modified such that LP relaxation solutions are removed, but all integer feasible solutions are preserved. Suppose we are given a constraint $\sum_{j \in N} a_j x_j \geq b$ with $a_k \geq \varepsilon := a_k - b + \sum_{j \neq k, a_j < 0} a_j > 0$, then the constraint can be strengthened to $(a_k - \varepsilon)x_k + \sum_{j \neq k} a_j x_j \geq b$. The case $a_k < 0$ is handled analogously. This technique is also known as *saturation* in the SAT community [10] and VeriPB provides a dedicated saturation rule that can be used directly for proving the correctness of coefficient tightening. The deletion of the original, weaker constraint can be proven automatically.

GCD-Based Simplification. This presolving technique from [51] uses a divisibility argument to first eliminate variables from a constraint and then tighten its right-hand side. Given $C \doteq \sum_{j \in N} a_j x_j \geq b$ with $|a_1| \geq \cdots \geq |a_n| > 0$. We define the greatest common divisor $g_k = \gcd(a_1, \ldots, a_k)$ as the largest value g such that $a_j / g \in \mathbb{Z}$ for all $j \in \{1, \ldots, k\}$. If for an index k it holds that $b - g_k \cdot \left\lceil \frac{b}{g_k} \right\rceil \geq \sum_{k < j \leq n, a_j > 0} a_j$ and $b - g_k \cdot \left\lceil \frac{b}{g_k} \right\rceil - g_k \leq \sum_{k < j \leq n, a_j < 0} a_j$, then all $a_{k+1}, \ldots, a_n$ can be set to 0. This first step can be certified as *weakening* [41] and VeriPB provides an out-of-the-box verification function for it. Finally, b can be rounded to $g_k \cdot \lceil b/g_k \rceil$. This rounding step can be certified by dividing C with g_k and then multiply it again with g_k.

Substituting Implied Free Variables. A variable x_j is called *implied free* if its lower bound and its upper bound can be derived from the constraints. For example, the constraints $x_1 - x_2 \geq 0$ and $x_2 \geq 0$ imply the lower bound $x_1 \geq 0$. If we have an implied free variable x_j in an equality $E \doteq a_j x_j + \sum_{k \neq j} a_k x_k = b$ with $a_j > 0$, then we can remove x_j from the problem by substituting it with $x_j = \big(b - \sum_{k \neq j} a_k x_k\big)/a_j$, see [2] for details.

To apply the substitution in the certificate we use aggregations to remove x_j from all constraints and the objective function update to remove x_j from the objective. If coefficients c_j/a_j or a_k/a_j are non-integer, then the resulting constraints are scaled as described in Sect. 3.1. To prove the deletion of E, we derive two constraints by adding $x_j \geq 0$ and $1 \geq x_j$ to E each, which results in

$$b \geq \sum_{k \neq j} a_k x_k \ \wedge \ \sum_{k \neq j} a_k x_k \geq b - a_j \,. \tag{6}$$

Then the deletion of $E_{\geq}$ can be certified by a witness $\omega = \{x_j \mapsto 1\}$. The constraint simplifies to (6) and is therefore fulfilled. Analogously, we use the witness $\omega = \{x_j \mapsto 0\}$ to certify the deletion of $E_{\leq}$. Finally, to delete the constraints in (6) we generate a subproof that shows that negation of the auxiliary constraints in (6) leads to $x_j \notin \{0, 1\}$. This is a contradiction to the implied variable bounds $0 \leq x_j \leq 1$. Since these bounds are still present through the implying constraints, we can add these implying constraints to (6) in the subproof to arrive at a contradiction.

Singleton Variables. It is well-known that variables that appear only in one inequality constraint or equality can be removed from the problem [2, Sect. 5.2]. This can be certified by applying one of the following primal or dual strategies in this order: First, try to apply duality-based fixing, see Sect. 3.3; second, an implied free singleton variable can be substituted as explained above; otherwise, the singleton variable can be treated as a *slack variable*: substitute the variable in the objective, then relax the equality as in (6), and delete the original constraint.

3.3 Dual Reductions

Dual reductions remove solutions while preserving at least one optimal solution. Hence, to prove the correctness of dual reductions we need to involve the redundance-based strengthening rule of VeriPB. For each derived constraint C we only explain how to prove $f \geq f\restriction_\omega$ (subject to the negation $\neg C$); the proof goals for $C\restriction_\omega$ can be derived in a very similar fashion.

Duality-Based Fixing. This presolving step described in [2, Sect. 4.2] counts the *down-* and *up-lock* of a variable. A down-lock on variable x_j is a negative coefficient, an up-lock on variable x_j is a positive coefficient (for $\geq$ constraints). If x_j has no down-locks and $c_j \leq 0$, it can be fixed to zero; if x_j has no up-locks and $c_j \geq 0$, it can be fixed to one. These reductions can be certified with redundance-based strengthening using the witness $\omega = \{x_j \mapsto v\}$, where v is the fixing value. The proof goal for $f \geq f\restriction_\omega$ is equivalent to $c_j x_j \geq c_j v$, which is fulfilled by the conditions of duality-based fixing.

Dominated Variables. A variable x_j is said to *dominate* another variable x_k [2, 25], in notation $x_j \succ x_k$, if

$$c_j \leq c_k \;\wedge\; a_{ij} \geq a_{ik} \text{ for all } i \in \{1, \ldots, m\}\,, \tag{7}$$

where a_{ij} and a_{ik} are the coefficients of variable x_j and x_k, respectively, in the i-th constraint. Variable x_j is then favored over x_k since x_j contributes less to the objective function, but more to the feasibility of the constraints. For every domination $x_j \succ x_k$, a constraint $C \doteq x_j \geq x_k$ can be introduced. This constraint can be certified by redundance-based strengthening with the witness $\omega = \{x_k \mapsto x_j, x_j \mapsto x_k\}$. The proof goal for $f \geq f\restriction_\omega$ is equivalent to

$$c_j x_j + c_k x_k \geq c_j x_k + c_k x_j\,. \tag{8}$$

The negated constraint $\neg C \doteq x_j < x_k$ leads to $x_k = 1$ and $x_j = 0$. Substituting these values in (8) leads to $c_k \geq c_j$, which follows directly from Condition (7).

Dominated Variables Advanced. For an implied free variable we can drop the variable bounds and pretend the variable is unbounded. This allows for additional fixings in the following cases of dominated variables:

(a) If the upper bound of x_j is implied and $x_j \succ x_k$, then $x_k = 0$.
(b) If the lower bound of x_k is implied and $x_j \succ x_k$, then $x_j = 1$.
(c) If the upper bound of x_j is implied and $x_j \succ -x_k$, then $x_k = 1$.
(d) If the lower bound of x_j is implied and $-x_j \succ x_k$, then $x_j = 0$.

We use redundance-based strengthening with witness $\omega = \{x_k \mapsto 0\}$ to prove the correctness of (a) as follows. If the upper bound of x_j is implied, this means there exists a constraint with $a_{ij} < 0$ such that $x_j \leq \left\lfloor \frac{b_\ell - \sum_{\ell \neq j, a_{i\ell} > 0} a_{i\ell}}{a_{ij}} \right\rfloor = 1$. Due to Condition (7), it must hold that $0 > a_{ij} \geq a_{ik}$, and the constraint $x_j + x_k \leq 1$ can be derived. Hence, negating and propagating $C \doteq x_k = 0$ with RUP leads to contradiction, which proves the validity of C. Case (b) can be handled analogously using the witness $\omega = \{x_k \mapsto 1\}$. To derive $C \doteq x_k = 1$ in (c) we use redundance-based strengthening with witness $\omega = \{x_k \mapsto 1, x_j \mapsto 1\}$. Then, the proof goal for $f \geq f\restriction_\omega$ is $c_j \cdot x_j + c_k \cdot x_k \geq c_j + c_k$. After propagating $\neg C$, this becomes equivalent to $c_j \leq -c_k$, which is true by Condition (7). Case (d) can be handled analogously using the witness $\omega = \{x_k \mapsto 0, x_j \mapsto 0\}$.

3.4 Example

We conclude this section with an example of a small certificate for the substitution of an implied free variable in Fig. 1, also available with a more detailed description at the software repository of PaPILO [38]. Consider the 0–1 ILP

$$\min\; x_1 + x_2 \;\text{ s.t. }\; x_1 + x_2 - x_3 - x_4 = 1\,, \tag{9}$$

$$-x_1 + x_5 \geq 0\,, \tag{10}$$

```
* generates ID 4:
pol 1 ~x1 + ;
core id 4
* generates ID 5:
pol 2 x1 + ;
core id 5
```

```
* generates ID 6:
pol 3 1 + ;
core id 6
delc 3 ;  ; begin
   pol 6 2 +
end
obju new +1 x3 +1 x4 1 ;
```

```
delc 2 ; x1 -> 0
delc 1 ; x1 -> 1
delc 5
delc 4 ; ; begin
   pol 6 -1 +
end
```

Fig. 1. A VeriPB certificate to substitute an implied free variable x_1.

in which the lower bound of x_1 is implied by (9) and the upper bound of x_1 is implied by (10). Hence, x_1 is implied free and we can use (9) to substitute it.

In the left section of Fig. 1 we first derive the two auxiliary constraints

$$0 \leq x_2 - x_3 - x_4 \leq 1\,, \tag{11}$$

which receives the constraint IDs 4 and 5 and are moved to the core. Note that the equality in (9) is split into two inequalities with IDs 1 and 2. In the middle section, we first remove x_1 from (10) by aggregation with (9), perform checked deletion, then remove x_1 from the objective (automatically proven by VeriPB). Last, in the right section, we delete the equality in (9) used for the substitution and the auxiliary constraints in (11) and arrive at the reformulated problem min $x_3 + x_4 + 1$ s.t. $x_2 - x_3 - x_4 + x_5 \geq 1$. From here, we could continue to derive $x_2 = 1$ by duality-based fixing, since x_2 has zero up-locks and objective coefficient zero. This displays the importance of the objective update, as without it x_2 would still contribute to the objective with a positive coefficient, and this would prohibit duality-based fixing to 1.

4 Computational Study

In this section we quantify the cost of *certifying* presolve reductions in a state-of-the-art implementation for MIP-based presolve (Sect. 4.2) and the cost of *verifying* the resulting certificates (Sect. 4.3). In Sect. 4.4, we analyze the impact of certifying constraint propagation by RUP or by an explicit cutting planes proof.

4.1 Experimental Setup

For generating the presolve certificates we use the solver-independent presolve library PaPILO [44], which provides a large set of MIP and LP techniques from the literature, described in Sect. 3. Additionally, it accelerates the search for presolving reductions by parallelization, encapsulating each reduction in a so-called transaction to avoid expensive synchronization [28]. Logging the certificate, however, is performed sequentially while evaluating the transactions.

We base our experiments on models from the Pseudo-Boolean Competition 2016 [45] including 1398 linear small integer decision and 532 linear small integer optimization instances of the competitions PB10, PB11, PB12, PB15, and PB16 and 295 decision and 145 optimization instances from MIPLIB 2017 [29] in the OPB translation [19], excluding 10 large-scale instances[1] for which PaPILO reaches the memory limit. This yields a total of 671 optimization and 1681 decision instances. We use PaPILO 2.2.0 [39] running on 6 threads and VeriPB 2.0 [30]. The experiments are carried out on identical machines with an 11th Gen Intel(R) Core(TM) i5-1145G7 @ 2.60 GHz CPU and 16 GB of memory and are assigned 14,000 MB of memory. The strict time limit for presolve plus certification and verification is three hours. Times (reported in seconds) do not include the time for reading the instance file. For all aggregations, we use the shifted geometric mean with a shift of 1 s.

4.2 Overhead of Proof Logging

In the first experiment, we analyze the overhead of proof logging in PaPILO. The average results are summarized in Table 2, separately over decision (dec) and optimization (opt) instances for PB16 and MIPLIB. Column "relative" indicates the average slow-down incurred by printing the certificate.

The relative overhead of proof logging is less than 6% across all test sets. VeriPB supports two variants to change the objective function. Either printing the entire objective (`obju new`) or printing only the changes in the objective (`obju diff`). In our experiments, we only print the changes, since printing the entire objective for each change can lead to a large certificate and overhead, especially for instances with large and dense objective functions. On the PB16 instance normalized-datt256, for example, PaPILO finds 135 206 variable fixings. Updating the entire objective function with 262 144 non-zeros for each of these variables leads to a huge certificate of about 138 GB and increases the time from 3.3 s (when printing only the changes) to 6625 s.[2]

For 99% of the instances, we can further observe that the *overhead per applied reduction* is below $0.001 \cdot 10^{-3}$ s over both test sets. This means that the proof logging overhead is not only small on average, but also small per applied reduction on the vast majority of instances. These results show that the overhead scales well with the number of applied reductions and that proof logging remains viable even for instances with many transactions. Here, under applied reductions we subsume all applied transactions and each variable fixing or row deletion in the first model clean-up phase. During model clean-up, PaPILO fixes variables and removes redundant constraints from the problem. While PaPILO technically does not count these reductions as full transactions found during the parallel presolve phase, their certification can incur the same overhead.

4.3 Verification Performance on Presolve Certificates

In this section, we analyze the time to verify the certificates generated by PaPILO. The results are summarized in Table 3. The "verified" column lists

[1] normalized-184, normalized-pb-simp-nonunif, a2864-99blp, ivu06-big, ivu59, supportcase11, a2864-99blp.0.s/u, supportcase11.0.s/u.

[2] Certificate generated on Intel Xeon Gold 5122 @ 3.60GHz 96 GB with 50,000 MB of memory assigned.

Table 2. Runtime comparison of PaPILO with and without proof logging.

test set	size	default [s]	w/proof log [s]	relative
PB16-dec	1397	0.06	0.06	1.00
MIPLIB-dec	291	0.42	0.43	1.02
PB16-opt	531	0.65	0.66	1.02
MIPLIB-opt	142	0.33	0.35	1.06

Table 3. Time to verify the certificates. VeriPB timeouts are treated with PAR2.

test set	size	verified	PaPILO time [s]		VeriPB	relative time w.r.t.	
			default	w/proof log	time [s]	default	w/proof log
PB16-dec	1397	1397	0.06	0.06	0.88	14.67	14.67
MIPLIB-dec	291	267	0.42	0.43	9.64	22.85	22.42
PB16-opt	531	520	0.65	0.66	10.44	16.06	15.82
MIPLIB-opt	142	139	0.33	0.35	5.25	15.91	15.00

the number of instances verified within 3 h. VeriPB timeouts are counted as twice the time limit, i.e., PAR2 score. Similar to Table 2, the "relative" columns report the relative overhead of VeriPB runtime compared to PaPILO.

First note that all certificates are verified by VeriPB (partially on the 38 instances where VeriPB times out). On average, it takes between 14.7 and 22.4 times as much time to verify the certificates than to produce them. Nevertheless, some instances take a longer than average time to verify. Over all test sets, 25% of the instances have an overhead of at least a factor of 193, see also Fig. 2.

To put this result into context, note that presolving amounts more to a transformation than to a (partial) solution of the problem. Each reduction has to be certified and verified while a purely solution-targeted algorithm may be able to skip certifying of a larger part of the findings that are not form a part of the final proof of optimality. Hence, it makes sense to compare the performance of VeriPB on presolve certificates to the overhead for, e.g., for verifying CNF translations [31]. For this study, a similar performance overhead is reported as in Fig. 2.

4.4 Performance Analysis on Constraint Propagation

Finally, we investigate how the performance of VeriPB depends on whether we use RUP (as in Sect. 4.2 and Sect. 4.3) or explicit cutting planes derivations (POL) to certify bound strengthening reductions from constraint propagation. Here, we additionally exclude 9 large-scale instances[3] for which PaPILO

[3] neos-4754521-awarau.0.s, neos-827015.0.s/u, neos-829552.0.s/u, s100.0.s/u, normalized-datt256, s100.

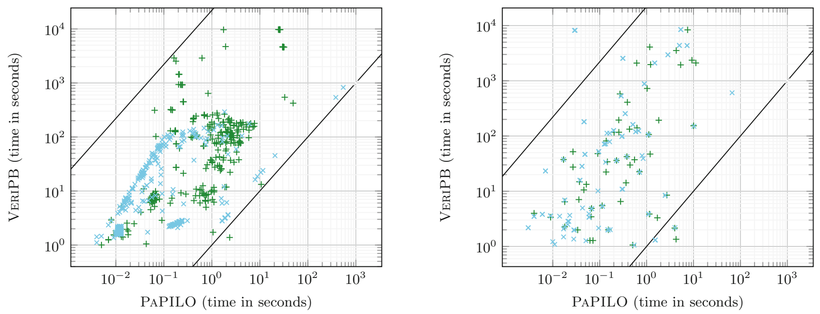

Fig. 2. Running times of VeriPB vs. PaPILO on test sets PB16 (left) and MIPLIB (right), including all instances with more than 1 s in VeriPB and less than 30 min in PaPILO, and excluding timeouts. Green + signs mark optimization and blue × signs mark decision instances. (Color figure online)

reaches the memory limit when certifying with POL. The results are summarized in Table 4. The "verified" column contains the number of instances verified by VeriPB within the time limit. The "time" column reports the time for verification.

Deriving the propagation directly with cutting planes is 3.2% faster on PB16-dec, 2.8% faster on MIPLIB-dec, 13.1% faster on MIPLIB-opt, and 0.7% faster on PB16-opt. On 95% of the decision instances using RUP is at most 9.7% slower. While it is expected that verification is faster when the cutting planes proof is given explicitly, it is surprising that the performance difference between the methods is not more pronounced. This is partly due to the cost of the watched-literal scheme [43,48] used by VeriPB for unit propagation. The overhead of maintaining the watches is present regardless of whether (reverse) unit propagation is used or not. Furthermore, unit propagation is also used for automatically verifying redundance-based strengthening. Together, this limits the potential for runtime savings by providing the explicit cutting planes proof.

Furthermore, providing an explicit cutting planes proof for propagation requires printing the constraint into the certificate. Hence, the certificate size becomes dependent on the number of non-zeros in the constraints leading to propagations. In contrast, the overhead of RUP is constant and much smaller.

All in all, these results suggest to prefer RUP when deriving constraint propagation since it barely impacts the performance of VeriPB and keeps the size of the certificate smaller. The computational cost of RUP could be further reduced by extending it to accept an ordered list of constraints that shall be propagated first, similar as in [16]. Such an extension could also be used for other presolving techniques, in particular probing and simple probing.

Table 4. Comparison of the runtime of VeriPB with RUP and POL over instances with at least 10 propagations.

test set	size	RUP		POL		relative
		verified	time [s]	verified	time [s]	
PB16-dec	284	284	2.21	284	2.14	0.968
MIPLIB-dec	35	31	153.23	31	148.88	0.972
PB16-opt	153	142	28.43	142	28.22	0.993
MIPLIB-opt	16	14	147.11	14	127.83	0.869

5 Conclusion

In this paper we set out to demonstrate how presolve techniques from state-of-the-art MIP solvers can be equipped with certificates in order to verify the equivalence between original and reduced models. Although the pseudo-Boolean proof logging format behind VeriPB [7] was not designed with this purpose in mind, we could show that a limited extension needed for handling updates of the objective function is sufficient to craft a certified presolver for 0–1 ILPs.

However, our experimental study on instances from pseudo-Boolean competitions and MIPLIB also exhibited that the verification of MIP-based presolving can suffer from large and overly verbose certificates. To shrink the proof size we introduced a sparse objective update function but identified further possible improvements. First, a native substitution rule in VeriPB would remove the need for the explicit derivation of new aggregations and the verification of checked deletion as described in Sect. 3.1. For instances where presolving is dominated by substitutions, we estimate that this would reduce certificate sizes by up to 90%, and no more time would be spent on checked deletion for substitutions. Second, augmenting the RUP syntax by the option to specify an ordered list of constraints to propagate first, similarly as in [16], would accelerate RUP, in particular for fast verification of bound strengthenings by constraint propagation.

While VeriPB is currently restricted to operate on integer coefficients only, the certification techniques presented in Sect. 3 do not rely on this assumption and are applicable to general binary programs. It has been shown how to construct VeriPB certificates for bounded integer domains [34,42], and within the framework of the generalized proof system laid out in [20], our certificates would even translate to continuous and unbounded integer domains. To conclude, we believe our results show convincingly that this type of proof logging techniques is a very promising direction of research also for MIP presolve beyond 0–1 ILPs.

Acknowledgements. The authors wish to acknowledge helpful technical discussions on VeriPB in general and the objective update rule in particular with Bart Bogaerts, Ciaran McCreesh, and Yong Kiam Tan. The work for this article has been partly conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM). Jakob Nordström was supported by the Swedish Research Council grant 2016-00782 and the Independent Research Fund Denmark grant 9040-00389B. Andy Oertel was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computational experiments were enabled by resources provided by LUNARC at Lund University.

References

1. Achterberg, T.: Constraint Integer Programming. Doctoral thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin (2007). https://doi.org/10.14279/depositonce-1634
2. Achterberg, T., Bixby, R., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS J. Comput. **32** (2019). https://doi.org/10.1287/ijoc.2018.0857
3. Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Metamorphic testing of constraint solvers. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 727–736. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_46
4. Berg, J., Bogaerts, B., Nordström, J., Oertel, A., Vandesande, D.: Certified core-guided MaxSAT solving. In: Pientka, B., Tinelli, C. (eds.) CADE 2023. LNCS, vol. 14132, pp. 1–22. Springer, Cham (2023)
5. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 336, 2nd edn. IOS Press, Amsterdam (2021)
6. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving. In: Biere et al. [5], chap. 9, pp. 391–435
7. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified symmetry and dominance breaking for combinatorial optimisation. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022), pp. 3698–3707 (Feb 2022)
8. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified dominance and symmetry breaking for combinatorial optimisation. J. Artif. Intell. Res. **77**, 1539–1589 (2023). preliminary version in AAAI 2022
9. Buss, S.R., Nordström, J.: Proof complexity and SAT solving. In: Biere et al. [5], chap. 7, pp. 233–350
10. Chai, D., Kuehlmann, A.: A fast pseudo-Boolean constraint solver. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **24**(3), 305–317 (2005). preliminary version in *DAC '03*
11. Chang, S.F., McCormick, S.T.: Implementation and computational results for the hierarchical algorithm for making sparse matrices sparser. ACM Trans. Math. Softw. **19**(3), 419-441 (1993). https://doi.org/10.1145/155743.152620
12. Cheung, K.K.H., Gleixner, A.M., Steffy, D.E.: Verifying integer programming results. In: Eisenbrand, F., Koenemann, J. (eds.) IPCO 2017. LNCS, vol. 10328, pp. 148–160. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59250-3_13
13. Cook, W., Coullard, C.R., Turán, G.: On the complexity of cutting-plane proofs. Discret. Appl. Math. **18**(1), 25–38 (1987)
14. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: A hybrid branch-and-bound approach for exact rational mixed-integer programming. Math. Program. Comput. **5**(3), 305–344 (2013)
15. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: A hybrid branch-and-bound approach for exact rational mixed-integer programming. Math. Program. Comput. **5**(3), 305–344 (2013). https://doi.org/10.1007/s12532-013-0055-6

16. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified rat verification. In: de Moura, L. (ed.) CADE 2017. LNCS, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_14
17. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Jr., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: de Moura, L. (ed.) CADE 2017. LNCS, vol. 10395, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_14
18. Cruz-Filipe, L., Marques-Silva, J.P., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 118–135. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_7
19. Devriendt, J.: Miplib 0-1 instances in opb format (2020). https://doi.org/10.5281/zenodo.3870965
20. Doornmalen, J.V., Eifler, L., Gleixner, A., Hojny, C.: A proof system for certifying symmetry and optimality reasoning in integer programming. Technical report 2311.03877, arXiv.org (2023)
21. Eifler, L., Gleixner, A.: A computational status update for exact rational mixed integer programming. Math. Program. (2022). https://doi.org/10.1007/s10107-021-01749-5
22. Eifler, L., Gleixner, A.: Safe and verified gomory mixed integer cuts in a rational MIP framework. SIAM J. Optim. **34**(1), 742–763 (2024). https://doi.org/10.1137/23M156046X
23. Eifler, L., Gleixner, A., Pulaj, J.: A safe computational framework for integer programming applied to chvátal's conjecture. ACM Trans. Math. Softw. **48**(2) (2022). https://doi.org/10.1145/3485630
24. Fügenschuh, A., Martin, A.: Computational integer programming and cutting planes. In: Aardal, K., Nemhauser, G., Weismantel, R. (eds.) Discrete Optimization, Handbooks in Operations Research and Management Science, vol. 12, pp. 69–121. Elsevier (2005). https://doi.org/10.1016/S0927-0507(05)12002-7
25. Gamrath, G., Koch, T., Martin, A., Miltenberger, M., Weninger, D.: Progress in presolving for mixed integer programming. Math. Programm. Comput. **7** (2015). https://doi.org/10.1007/s12532-015-0083-5
26. Gemander, P., Chen, W.K., Weninger, D., Gottwald, L., Gleixner, A.: Two-row and two-column mixed-integer presolve using hashing-based pairing methods. EURO J. Comput. Optim. **8**(3–4), 205–240 (2020). https://doi.org/10.1007/s13675-020-00129-6
27. Gillard, X., Schaus, P., Deville, Y.: SolverCheck: declarative testing of constraints. In: Schiex, T., de Givry, S. (eds.) CP 2019. LNCS, vol. 11802, pp. 565–582. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30048-7_33
28. Gleixner, A., Gottwald, L., Hoen, A.: PaPILO: a parallel presolving library for integer and linear programming with multiprecision support. INFORMS J. Comput. (2023). https://doi.org/10.1287/ijoc.2022.0171.cd, https://github.com/INFORMSJoC/2022.0171
29. Gleixner, A., et al.: MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. Math. Program. Comput. **13**, 443–490 (2021). https://doi.org/10.1007/s12532-020-00194-3
30. Gocht, S., Oertel, A.: Veripb (2023). https://gitlab.com/MIAOresearch/software/VeriPB, githash: dd7aa5a1
31. Gocht, S., Martins, R., Nordström, J., Oertel, A.: Certified CNF translations for pseudo-Boolean solving. In: Proceedings of the 25th International Conference on

Theory and Applications of Satisfiability Testing (SAT 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 236, pp. 16:1–16:25 (Aug 2022)
32. Gocht, S., McBride, R., McCreesh, C., Nordström, J., Prosser, P., Trimble, J.: Certifying solvers for clique and maximum common (connected) subgraph problems. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 338–357. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_20
33. Gocht, S., McCreesh, C., Nordström, J.: Subgraph isomorphism meets cutting planes: Solving with certified solutions. In: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20), pp. 1134–1140 (2020)
34. Gocht, S., McCreesh, C., Nordström, J.: An auditable constraint programming solver. In: Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22). Leibniz International Proceedings in Informatics (LIPIcs), vol. 235, pp. 25:1–25:18 (2022)
35. Gocht, S., Nordström, J.: Certifying parity reasoning efficiently using pseudo-Boolean proofs. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21), pp. 3768–3777 (2021)
36. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13), pp. 181–188 (2013)
37. Heule, M.J.H., Hunt, W.A., Wetzler, N.: Verifying refutations with extended resolution. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 345–359. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_24
38. Hoen, A.: Papilo: Parallel presolve integer and linear optimization (2023). https://github.com/scipopt/papilo/tree/develop/check/VeriPB, githash: 5df3dd6d
39. Hoen, A., Gottwald, L.: Papilo: parallel presolve integer and linear optimization (2023). https://github.com/scipopt/papilo, githash: 3b082d4
40. Klotz, E.: Identification, assessment, and correction of ill-conditioning and numerical instability in linear and integer programs. In: Newman, A., Leung, J. (eds.) Bridging Data and Decisions, pp. 54–108. TutORials in Operations Research (2014). https://doi.org/10.1287/educ.2014.0130
41. Le Berre, D., Marquis, P., Wallon, R.: On weakening strategies for PB solvers. In: Pulina, L., Seidl, M. (eds.) SAT 2020. LNCS, pp. 322–331. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_23
42. McIlree, M., McCreesh, C.: Proof logging for smart extensional constraints. In: Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23). Leibniz International Proceedings in Informatics (LIPIcs), vol. 280, pp. 26:1–26:17 (2023)
43. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC '01), pp. 530–535 (2001)
44. PaPILO — parallel presolve for integer and linear optimization. https://github.com/lgottwald/PaPILO
45. Roussel, O.: Pseudo-boolean competition 2016 (2016). http://www.cril.univ-artois.fr/PB16/
46. Sahraoui, Y., Bendotti, P., D'Ambrosio, C.: Real-world hydro-power unit-commitment: dealing with numerical errors and feasibility issues. Energy **184**, 91–104 (2019). https://doi.org/10.1016/j.energy.2017.11.064, shaping research in gas-, heat- and electric- energy infrastructures
47. Savelsbergh, M.: Preprocessing and probing techniques for mixed integer programming problems. ORSA J. Comput. **6** (1994).https://doi.org/10.1287/ijoc.6.4.445

48. Sheini, H.M., Sakallah, K.A.: Pueblo: a hybrid pseudo-Boolean SAT solver. J. Satisfiability, Boolean Model. Comput. **2**(1–4), 165–189 (2006). preliminary version in *DATE '05*
49. Steffy, D.E.: Topics in exact precision mathematical programming. Ph.D. thesis, Georgia Institute of Technology (2011). http://hdl.handle.net/1853/39639
50. Vandesande, D., De Wulf, W., Bogaerts, B.: QMaxSATpb: a certified MaxSAT solver. In: Gottlob, G., Inclezan, D., Maratea, M. (eds.) LPNMR 2022. LNCS, vol. 13416, pp. 429–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15707-3_33
51. Weninger, D.: Solving mixed-integer programs arising in production planning. Phd thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (2016)
52. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31