Improving Conflict Analysis in MIP Solvers by Pseudo-Boolean Reasoning

Gioni Mexi
Zuse Institute Berlin, Germany

Timo Berthold
Fair Isaac Deutschland GmbH, Berlin, Germany
TU Berlin, Germany

Ambros Gleixner
HTW Berlin, Germany
Zuse Institute Berlin, Germany

Jakob Nordström
University of Copenhagen, Denmark
Lund University, Sweden

Abstract

Conflict analysis has been successfully generalized from Boolean satisfiability (SAT) solving to mixed integer programming (MIP) solvers, but although MIP solvers operate with general linear inequalities, the conflict analysis in MIP has been limited to reasoning with the more restricted class of clausal constraint. This is in contrast to how conflict analysis is performed in so-called pseudo-Boolean solving, where solvers can reason directly with 0–1 integer linear inequalities rather than with clausal constraints extracted from such inequalities.

In this work, we investigate how pseudo-Boolean conflict analysis can be integrated in MIP solving, focusing on 0–1 integer linear programs (0–1 ILPs). Phrased in MIP terminology, conflict analysis can be understood as a sequence of linear combinations and cuts. We leverage this perspective to design a new conflict analysis algorithm based on mixed integer rounding (MIR) cuts, which theoretically dominates the state-of-the-art division-based method in pseudo-Boolean solving.

We also report results from a first proof-of-concept implementation of different pseudo-Boolean conflict analysis methods in the open-source MIP solver SCIP. When evaluated on a large and diverse set of 0–1 ILP instances from MIPLIB 2017, our new MIR-based conflict analysis outperforms both previous pseudo-Boolean methods and the clause-based method used in MIP. Our conclusion is that pseudo-Boolean conflict analysis in MIP is a promising research direction that merits further study, and that it might also make sense to investigate the use of such conflict analysis to generate stronger no-goods in constraint programming.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Solvers

Keywords and phrases Integer programming, pseudo-Boolean solving, conflict analysis, cutting planes proof system, mixed integer rounding, division, saturation

Digital Object Identifier 10.4230/LIPIcs.CP.2023.27

Funding The work for this article has been conducted within the Research Campus Modal funded by the German Federal Ministry of Education and Research (BMBF grant numbers 05M14ZAM, 05M20ZBM).
Jakob Nordström: supported by the Swedish Research Council grant 2016-00782 and the Independent Research Fund Denmark grant 9040-00389B.

Acknowledgements Part of this work was carried out while some of the authors participated in the extended reunion for the program *Satisfiability: Theory, Practice, and Beyond* at the Simons Institute for the Theory of Computing at UC Berkeley in the spring of 2023. This work has also benefited greatly from discussions during the Dagstuhl Seminar 22411 *Theory and Practice of SAT and Combinatorial Solving*.

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).
Editor: Roland H. C. Yap; Article No. 27; pp. 27:1–27:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The area of Boolean satisfiability (SAT) solving has witnessed dramatic performance improvements over the last couple of decades, and several techniques from SAT have also inspired developments for other combinatorial optimization paradigms such as SAT-based and (linear) pseudo-Boolean optimization, constraint programming, and mixed integer programming. In particular, conflict analysis as introduced in the works on *conflict-driven clause learning (CDCL)* [3, 38, 40] ushering in the modern SAT solving revolution has been picked up and generalized in different ways to these more general settings. Interestingly, precursors of this version of conflict analysis and nonchronological backtracking can be traced back all the way to early work in the AI community [47], and related ideas have been used in constraint programming for decades [24, 31]. Our focus in this paper is on conflict analysis in mixed integer programming and pseudo-Boolean optimization, which we proceed to discuss next.

1.1 Mixed Integer Programming and Conflict Analysis

The core method of mixed integer programming (MIP) is that a linear programming (LP) relaxation of the problem is fed to an LP solver. If the LP solver finds a solution that assigns real values to integral variables, then either additional cut constraints can be generated that eliminate such solutions, or the problem can be split into subproblems by branching on integer variables, generating new nodes in the search tree. During the solving process infeasible nodes in the search tree are pruned. Unlike in SAT, there can be different reasons for backtracking due to infeasibility of the LP relaxation, node presolving (propagation), or to the current objective value of the relaxed problem being worse than the best solution found so far (branch-and-bound). MIP solvers employ a multitude of further techniques such as symmetry detection, disjoint subtree detection, restarts, et cetera. For a comprehensive description of MIP solving we refer the reader to, e.g., [2].

The use of SAT techniques in MIP solvers has been a fruitful direction of research over the last decades. Specifically, CDCL conflict analysis has proven to be a useful tool to enhance the performance of MIP solvers by learning constraints from infeasibilities detected by propagation or from the LP relaxation [1, 44, 48]. However, SAT and MIP solvers differ fundamentally in how they explore the search space, in that SAT solvers search depth-first, maintaining only the current state of the search, whereas in MIP the search tree is generated in a “best-first” manner based on careful analysis on search statistics such as dual bounds and integrality of LP solutions to subproblems. These differences make it harder for MIP solvers to profit from conflict analysis, and so in contrast to SAT solving, for which this technique is absolutely crucial, in MIP solving it plays more of a supplemental if still highly valuable role.

Although the setting is different, the *graph-based conflict analysis* [1] used to learn from infeasibilities in MIP is very similar to the classic SAT approach. First, a partial assignment is extracted that consists of branching decisions and implications that led to the infeasibility. If the LP relaxation is infeasible, the information which bound changes led to infeasibility is gathered from the non-zero duals of the LP. Next, a directed acyclic graph is constructed that encodes information about the conflict, in that source nodes correspond to branching decisions, non-source nodes encode implications, and the sink node represents the infeasibility. Each cut in this graph that separates the source nodes from the sink is a valid constraint. It is important to note that all implications correspond to clausal constraints, and so this conflict analysis operates not on the linear constraints of the problem but on clauses extracted from these linear constraints. (There are also methods that can learn general linear constraints from infeasibilities, one notable example being *dual-proof analysis* [48], but this technique is limited to conflicts arising from infeasibility of the LP relaxation and does not analyze or strengthen the partial assignment that led to infeasibility.)

1.2 Pseudo-Boolean Solving and Conflict Analysis

Pseudo-Boolean (PB) solving is another approach specific to integer linear programs with only binary variables, or 0–1 ILPs, which are referred to as (linear) pseudo-Boolean formulas in the PB solving literature. While MIP solvers find real-valued solutions and try to push such solutions closer and closer to integrality, PB solvers follow the SAT approach of considering only Boolean assignments and trying to extend partial assignments to more and more variables without violating any constraints. Just as in SAT, this search is performed in a depth-first manner.

Some PB solvers stick very closely to SAT in that they immediately translate the 0–1 ILP into conjunctive normal form (CNF) using auxiliary variables and then run a standard CDCL SAT solver [21, 39, 43]. Another approach, which is what is of interest in the context of this work, is to extend the solvers to reason natively with 0-1 linear inequalities [12, 46, 34, 23]. Such *conflict-driven pseudo-Boolean solvers* have the potential to run exponentially faster than CDCL-based solvers, since their conflict analysis method is exponentially stronger than that used in CDCL SAT solvers.

Since it is crucial for our work to understand the differences between conflict analysis in MIP and PB solvers, let us try to provide a somewhat simplified exposition of PB solving in a language that is meant to convey a MIP perspective (and where what follows below is heavily indebted to [19]). During the search phase, the pseudo-Boolean solver always first tries to extend the current partial solution with any variable assignments that are propagated by some linear inequality. When no further propagations are possible, the solver chooses some unassigned variable and makes a *decision* to assign this variable 0 or 1, after which it again turns to propagation. This cycle of decisions and propagations repeats until either a satisfying assignment is found or some 0–1 linear inequality C is violated. In the latter case, the solver switches to the conflict analysis phase, which works as follows:

1. The linear inequality R responsible for propagating the last variable x in C to the "wrong value" from the point of view of C is identified; this inequality R is referred to as the *reason constraint* for x.
2. A *division* or *saturation* rule is applied to R to generate a modified inequality R^* that propagates x tightly to its assigned value even when considered over the reals.
3. A new linear constraint D is computed as the smallest integer linear combination of R^* and C for which the variable x cancels and is eliminated. It is not too hard to show that it follows from the description above that this constraint D is violated by the current partial assignment of the solver with the value of x removed, and we can set $C := D$ and go to step 1 again.

This continues until a termination criterion analogous to the *unique implication point (UIP)* notion used in SAT solving leads to D being declared as the learned constraint. At this point, the solver undoes further assignments in reverse chronological order until D is no longer violated, and then switches back to the search phase. We refer the reader to the chapter [11] for a more detailed description of conflict-driven pseudo-Boolean solving (and to the handbook [8] for an in-depth treatment of SAT and related topics in general).

In contrast to MIP conflict analysis, the algorithm described above is not phrased in terms of the conflict graph, but focuses on the syntactic *resolution* method [9, 17, 16, 42] employed in CDCL conflict analysis and harnesses the observation by Hooker [29, 30] that resolution can be understood as a cut rule and extended to 0–1 integer linear inequalities. The conflict-graph-based analysis in MIP does not operate on the reason constraints R as described above, but instead on disjunctive clauses extracted from these constraints. It is not hard to prove formally (appealing to [4, 14, 28]) that this incurs an exponential loss in reasoning power compared to performing derivations on the linear constraints themselves.

In practice, however, it seems fair to say that current pseudo-Boolean solvers do not quite deliver on this promise of exponential gains in performance. Although there are specific problem domains where PB solvers outperform even commercial MIP solvers [35, 45], evaluations over larger sets of benchmarks [5, 19, 20] have demonstrated that the open-source MIP solver SCIP [7] tends to be clearly more effective in solving pseudo-Boolean optimization problems, and is also quite competitive for decision problems. This is especially so for some decision problems that are in some sense close to LP-infeasibility – such problems are almost trivial for MIP solvers, but can be extremely challenging for pseudo-Boolean solvers [22].

1.3 Questions Studied in This Work and Our Contributions

Since mixed integer programming solvers and pseudo-Boolean solvers approach 0–1 integer linear problems from quite different angles, and seem to have complementary performance profiles, it is natural to ask whether techniques from one of the paradigms can be used to improve solvers based on the other paradigm.

Some MIP-inspired approaches have been integrated with success in SAT and PB solvers, perhaps most recently in [19], where the PB solver RoundingSAT [23] makes careful use of the LP solver SoPlex [7] to detect infeasibility of LP relaxations and generate cut constraints (though this paper also raises many questions that would seem to merit further study). However, in the other direction we are not aware of any work trying to harness state-of-the-art techniques from pseudo-Boolean solving to improve the performance of MIP solvers.

In this work, we consider how the clausal conflict analysis in MIP solvers can be replaced by pseudo-Boolean reasoning, focusing on 0–1 integer linear programs. A key difference between the clausal and pseudo-Boolean conflict analysis methods is that in the latter algorithm the linear reason constraint R propagating a variable assignment might need to be modified, or *reduced*, to another constraint R^* that propagates tightly also over the reals (which is already guaranteed to hold if R is a clausal constraint). Viewed from a MIP perspective, this *reduction* step deriving R^* from R can be seen to be an application of one of two specific cut rules, where saturation-based reduction as in [34] corresponds to coefficient tightening and division-based reduction as in [23] uses Chvátal-Gomory cuts.

This observation raises the question of whether more general cuts could also used to obtain other, and potentially more powerful, reduction methods for pseudo-Boolean conflict analysis. The answer turns out to be yes, and, in particular, we introduce a new reduction algorithm utilizing mixed integer rounding (MIR) cuts [27, 37]. A theoretical comparison of the MIR-based reduction rule with the reduction methods currently used in PB solvers show that MIR-based reduction dominates the division-based method that is considered to be state of the art in pseudo-Boolean solving, while saturation-based reduction and MIR-based reduction appear to be incomparable.

We implement pseudo-Boolean conflict analysis for 0–1 ILPs in the MIP solver SCIP, including all three reduction methods discussed above, and compare these different flavours of PB conflict analysis with each other as well as with clausal MIP conflict analysis on a large benchmark set consisting of pure 0–1 ILP instances from MIPLIB 2017. We find that the MIR-based pseudo-Boolean conflict analysis has the best performance, beating not only the conflict analysis methods in the PB literature but also the standard clausal conflict analysis in SCIP. Interestingly, the new method is better measured not only in terms of number of nodes in the search tree, but also in terms of the number of instances solved, even though we only provide a proof-of-concept implementation lacking many of the optimizations that would be included in a full integration of this method into the SCIP codebase. Although our experimental data cannot provide conclusive evidence as to what causes this improved performance, we observe that the constraints learned from pseudo-Boolean conflict analysis seem more useful in that they take part more actively in propagations than constraints obtained by clausal conflict analysis.

1.4 Organization of This Paper

After reviewing preliminaries in Section 2, we give a detailed description of clausal and pseudo-Boolean conflict analysis for 0–1 integer linear programs in Section 3, including a discussion of the reduction methods found in the PB literature and our new version using mixed integer rounding cuts, and study how the different reduction rules compare in theory. In Section 4 we present our experimental results. We conclude the paper in Section 5 by summarizing our work and discussing direction for future research.

2 Preliminaries and Notation

Let $n \in \mathbb{Z}_{>0}$, and $\mathcal{N} := [1, \dots, n]$. We let x_i denote Boolean (i.e., $\{0,1\}$-valued) variables and ℓ_i denote literals, which can be either x_i or its negation $\overline{x}_i = 1 - x_i$. A *pseudo-Boolean constraint* is a 0–1 integer linear inequality

$$\sum_{i \in \mathcal{N}} a_i \ell_i \geq b \ , \tag{1}$$

where we can assume without loss of generality that $a_i \in \mathbb{Z}_{\geq 0}$ for all $i \in \mathcal{N}$ and $b \in \mathbb{Z}_{\geq 0}$ (so-called *normalized form*). We can convert "$\leq$"-constraints with 0–1 variables to "$\geq$"-constraints by multiplying the constraint by -1 and normalizing, i.e., replacing the variables by literals. Moreover, equalities "$=$" can be viewed as syntactic sugar for two opposing inequalities, which can also be transformed into normalized pseudo-Boolean format. In particular, every pure 0–1 integer linear program can be transformed to a normalized pseudo-Boolean representation. Note that in Section 3 we develop our theory and algorithms using normalized PB constraints for simplicity of exposition. However, in our actual implementation and experiments (described in Section 4), we directly operate on general linear constraints.

A (partial) assignment ρ is a (partial) map from variables to $\{0,1\}$, which is extended to literals by respecting the meaning of negation. We call a literal ℓ_i *falsified* or *false* if $\rho(\ell_i) = 0$ and *satisfied* or *true* if $\rho(\ell_i) = 1$. If ρ is undefined for a literal, we call the literal *unassigned* or *free*. A constraint is satisfied under some partial assignment ρ if the respective inequality holds, independently of which values the unassigned literals take, and is falsified if no assignment to the unassigned literals can make the inequality true.

The slack of a PB constraint $C : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ under a partial assignment ρ is defined as $\mathit{slack}(C, \rho) := \sum_{\{i \in \mathcal{N} : \rho(i) \neq 0\}} a_i - b$. With this definition, C is falsified under ρ if and only if $\mathit{slack}(C, \rho) < 0$. For example the constraint $C : 2\overline{x}_1 + 2x_2 + 3x_3 \geq 4$ is falsified under the partial assignment $\rho = \{x_1 = 1, x_2 = 0\}$ since $\mathit{slack}(C, \rho) = -1 < 0$. For a non-falsified constraint C and an unassigned literal ℓ_i with coefficient a_i, the constraint propagates ℓ_i if and only if $\mathit{slack}(C, \rho) < a_i$. For instance, the same constraint $C : 2\overline{x}_1 + 2x_2 + 3x_3 \geq 4$

propagates both variables x_2 and x_3 to 1 under the partial assignment $\rho = \{x_1 = 1\}$ since $\mathit{slack}(C, \rho) = 1$ is strictly smaller than the coefficients of each of the variables. A constraint propagates the assignment of a free variable tightly if the slack under the current partial assignment is 0. For any two pseudo-Boolean constraints C and C' and partial assignment ρ it holds that the slack is subadditive, i.e., $\mathit{slack}(C + C', \rho) \leq \mathit{slack}(C, \rho) + \mathit{slack}(C', \rho)$. The decision level of a literal ℓ_i under a partial assignment ρ is the number of decisions prior to the fixing of ℓ_i. Note that the first fixing in every decision level is a decision literal.

3 Conflict Analysis Algorithms

For simplicity, in this section we present all algorithms in a pseudo-Boolean framework, where all coefficients and constants are integral, and the proofs of correctness that we provide also make crucial use of this fact. It is important to note that this is not the case in the actual implementation in SCIP, which operates with real-valued coefficients and constants. In fact, one of the challenges in implementing pseudo-Boolean conflict analysis in a MIP framework is that careful thought is required to rephrase the algorithms in such a way that they can deal with real-valued data but are still correct. Next, we describe the details of conflict analysis algorithms used in PB solvers and the different techniques that we consider in this paper.

3.1 Clausal Conflict Analysis

To explain the idea of conflict analysis, we first consider the case where all constraints are clauses. Conflict analysis begins at the stage where a conflict clause C_{confl} is falsified by the current partial assignment ρ. Let ℓ_r be the literal in C_{confl} that was last propagated to false, and let C_{reason} be the reason clause in chronological order that is responsible for the propagation, i.e., we have $C_{\text{confl}} = C' \vee \ell_r$ and $C_{\text{reason}} = C'' \vee \bar{\ell}_r$. Using the resolution rule, we can derive the so-called *resolvent* $C' \vee C''$ as a new learned clause C_{learn}.

Note that, even after removing ℓ_r from the partial assignment ρ, both C' and C'' remain falsified: C' because $C_{\text{confl}} = C' \vee \ell_r$ and ℓ_r were false, and C'' because $C_{\text{reason}} = C'' \vee \bar{\ell}_r$ propagated. This is the key invariant of the algorithm: At any point during the algorithm the resolvent is falsified by the remaining partial assignment ρ.

Hence, we can replace the conflict clause by the resolvent and continue this process. At each step either a propagating literal is removed from ρ or the learned clause is empty (at which point unsatisfiability is proven) or the last fixed literal is a decision literal. In the third case, we have reached a *first unique implication point* (FUIP) and conflict analysis terminates, with the final resolvent being the learned clause C_{learn}. With C_{learn} added, propagation on the previous decision level will prevent the last infeasible decisions to happen as the search continues.

It is straightforward to apply this algorithm to problems with 0–1 linear constraints. Suppose $\sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ is the initial conflict constraint falsified under ρ, then $\bigvee_{i: a_i > 0 \wedge \rho(\ell_i) = 0} \ell_j$ can be used as initial conflict clause. Analogously, we can extract at each step a reason clause from the linear constraint that propagated the last literal and perform resolution. After terminating at an FUIP, the learned clause can be added as linear constraint to the solver.

3.2 PB Conflict Analysis

As in the clausal version, the main idea of PB conflict analysis is also to find a new constraint that explains the infeasibility of the current subproblem under a falsifying partial assignment. Algorithm 1 shows the base algorithm for all variants of PB conflict analysis considered in this paper, using the first unique implication point (FUIP) learning scheme. It is initialized with a falsifying partial assignment ρ and a conflicting constraint C_{confl} under ρ. First, the learned conflict constraint C_{learn} is set equal to the conflict constraint C_{confl}. In each iteration, we extract the latest literal ℓ_r from ρ. If the literal assignment was due to propagation of a constraint and the negated literal $\bar{\ell}_r$ occurs in C_{confl}, then we extract the reason constraint C_{reason} that propagated ℓ_r. In line 6 we "reduce" the reason constraint such that the resolvent of C_{learn} and the reduced reason C_{reason} (Line 7) that cancel the last literal ℓ_r is still falsified under the remaining partial assignment ρ. The conflict constraint is set to the resolvent and we continue until we reach an FUIP (C_{learn} is *asserting*) or we prove C_{learn} makes the problem infeasible. We have reached an FUIP if C_{learn} would propagate some literal after removing at least all literal assignments in the current decision level from ρ. We have shown that the problem is infeasible if C_{learn} is falsified under an empty partial assignment ρ. At this point, the learned constraint can be added to the constraint database of our problem to prevent the solver from exploring the same search space again.

Algorithm 1 Pseudo-Boolean Conflict Analysis Algorithm.

Input : conflict constraint C_{confl}, falsifying partial assignment ρ
Output : learned conflict constraint C_{learn}

1 $C_{\text{learn}} \leftarrow C_{\text{confl}}$
2 **while** C_{learn} not asserting **and** $C_{\text{learn}} \neq \perp$ **do**
3     $\ell_r \leftarrow$ literal last assigned on ρ
4   **if** ℓ_r propagated **and** $\bar{\ell}_r$ occurs in C_{learn} **then**
5         $C_{\text{reason}} \leftarrow \texttt{reason}(\ell_r, \rho)$
6         $C_{\text{reason}} \leftarrow \texttt{reduce}(C_{\text{reason}}, C_{\text{learn}}, \ell_r, \rho)$
7         $C_{\text{learn}} \leftarrow \texttt{resolve}(C_{\text{learn}}, C_{\text{reason}}, \ell_r)$
8     $\rho \leftarrow \rho \setminus \{\ell_r\}$
9 **return** C_{learn}

The key invariant of Algorithm 1 is that in each iteration the resolvent C_{learn} remains falsified. In the clausal version this holds even without the reduction step in line 6. However, for general linear constraints this is not the case, as shown by the next example.

▶ **Example 1.** Consider the two PB constraints $C_{\text{reason}} = x_1 + x_2 + 2x_3 \geq 2$ and $C_{\text{confl}} = x_1 + 2\overline{x}_3 + x_4 + x_5 \geq 3$ and the partial assignment $\rho = \{x_1 = 0, x_3 = 1\}$ where $x_1 = 0$ is a decision, and $x_3 = 1$ is propagated by C_{reason}. Under ρ the constraint C_{confl} is falsified. Applying generalized resolution to cancel x_3 yields the constraint $2x_1 + x_2 + x_4 + x_5 \geq 3$ which is not falsified under ρ.

In the following sections, we present three different reduction techniques for Algorithm 1 that operate directly on PB constraints. The main idea is to apply valid operations on the reason constraint to reduce the slack and ensure that the resolvent will have negative slack. The two main ingredients of the reduction techniques are *weakening* and *cutting planes* and are applied to the reason constraint until the invariant is fulfilled.

Weakening a literal in a PB constraint simply sets it to 1. For example, weakening a constraint $C : x_1 + x_2 + 2x_3 \geq 2$ on x_1 yields $x_2 + 2x_3 \geq 1$. Weakening is a valid operation since it simply adds a multiple of the valid bound constraint $\overline{x}_1 \geq 0$ to C. Note that weakening entails a loss of information. However, as we will see, it is a necessary operation to reduce the slack of the reason constraint. Note that whenever weakening is applied on non-falsified literal, it does not change the slack of the constraint. See Section 3.7 for more details on weakening.

Our main focus in this paper, however, is the second necessary ingredient of the reduction algorithm: cutting planes (cuts). Cuts are applied to the "weakened" version of the reason constraint in order to reduce its slack. We first present two well-documented cuts from existing literature, namely *Saturation* (Section 3.3) and *Division* (Section 3.4). Both ensure the reduction of the slack of the reason constraint to 0 at least after weakening all non-falsified literals in the original reason constraint. In Section 3.5, we introduce a new cut based on the *Mixed Integer Rounding (MIR)* procedure and prove that it has the same property. In Section 3.6 we show that the reduction algorithm using MIR always returns an equally strong or stronger reason constraint than the reduction using Division.

3.3 Saturation-based Reduction

First, we present the *Saturation* cut. Then, we provide details about the *Saturation-based Reduction* algorithm and demonstrate how the reduction ensures that the key invariant of conflict analysis holds.

▶ **Definition 2** (Saturation Cut). *Let $C : \sum_{i\in\mathcal{N}} a_i\ell_i \geq b$. The* ***Saturation Cut*** *of C is given by the constraint*

$$\sum_{i\in\mathcal{N}} \min\{a_i, b\}\ell_i \geq b.$$

Saturation is a valid cut known as coefficient tightening cut in the MIP literature [10] and does not entail a loss of information. Algorithm 2 is used to reduce the reason constraint C_{reason} before applying generalized resolution. Similar to the implementation in [12], in each iteration, the algorithm picks a non-falsified literal in the reason constraint different from the literal we are resolving on and weakens it. Then it applies the Saturation cut to the resulting constraint. The algorithm terminates when the slack of the resolvent becomes negative.

Algorithm 2 Saturation-based Reduction Algorithm.

Input : conflict constraint C_{confl}, reason constraint C_{reason}, literal to resolve ℓ_r, partial assignment ρ

Output : reduced reason C_{reason}

while $\mathit{slack}((\texttt{resolve}(C_{reason}, C_{confl}, \ell_r)), \rho) \geq 0$ **do**

  $\ell_j \leftarrow$ non falsified literal in $C_{\text{reason}} \setminus \{\ell_r\}$

  $C_{\text{reason}} \leftarrow \texttt{weaken}(C_{\text{reason}}, \ell_j)$

  $C_{\text{reason}} \leftarrow \texttt{saturate}(C_{\text{reason}})$

return C_{reason}

For completeness, we prove the following well-known fact that demonstrates that the slack of the reason constraint will be reduced to 0 at the latest after weakening the last non-falsified literal and applying the Saturation cut. Since the slack is subadditive, the resolvent's slack becomes negative and the resolvent is thus falsified.

▶ **Lemma 3.** *Let ρ be a partial assignment, and $C_{reason} : \sum_{i\in\mathcal{N}} a_i\ell_i \geq b$ a constraint propagating a literal ℓ_r to 1. Further, assume that $\mathit{slack}(C_{reason}, \rho) > 0$. Then, after weakening all non-falsified literals in C_{reason} (except for ℓ_r) and applying Saturation on C_{reason}, the slack of the reduced reason constraint is 0.*

Proof. First, we rewrite the constraint C_{reason} as

$$\sum_{j:\rho(j)=0} a_j\ell_j + \sum_{i\neq r:\rho(i)\neq 0} a_i\ell_i + a_r\ell_r \geq b.$$

Since $slack(C_{\text{reason}}, \rho) := \sum_{i\neq r:\rho(i)\neq 0} a_i + a_r - b > 0$, it holds that

$$a_r > b - \sum_{i\neq r:\rho(i)\neq 0} a_i. \tag{2}$$

After weakening all literals from $\{i \neq r : \rho(i) \neq 0\}$ the constraint C_{reason} becomes

$$\sum_{j:\rho(j)=0} a_j\ell_j + a_r\ell_r \geq \tilde{b} := b - \sum_{i\neq r:\rho(i)\neq 0} a_i. \tag{3}$$

Applying Saturation on (3) sets a_r to $\tilde{b}$ because of (2). Therefore the slack of the reduced reason constraint becomes $\tilde{b} - \tilde{b} = 0$. ◀

3.4 Division-based Reduction

A very competitive alternative to Saturation in the reduction algorithm is based on Division cuts. Division is also a valid cut known as Chvátal-Gomory cut in the MIP literature [13].

▶ **Definition 4** (Division Cut). *Let $C : \sum_{i\in\mathcal{N}} a_i\ell_i \geq b$. The **Division Cut** of C with divisor $d \in \mathbb{Z}_{>0}$ is given by the constraint*

$$\sum_{i\in\mathcal{N}} \left\lceil \frac{a_i}{d} \right\rceil \ell_i \geq \left\lceil \frac{b}{d} \right\rceil.$$

To see why this procedure is valid, we can think of it as three steps: dividing by d maintains the validity of the constraint; rounding up coefficients on the left-hand side relaxes the constraint and is hence valid; the validity of rounding up the right-hand side follows from the integrality of the left-hand side coefficients and literals.

In the Division-based reduction algorithm, the divisor d used is the coefficient of the literal ℓ_r we are resolving on. As proven in [23], it suffices to weaken non-falsified variables with a coefficient that is not a multiple of a_r, i.e., from the index set $W := \{i \in \mathcal{N} : \rho(i) \neq 0 \text{ and } a_i \nmid a_r\}$. After weakening all literals in W and applying Division on C_{reason}, the slack of the reduced reason constraint is 0, which for completeness we include in Lemma 6 below.

3.5 MIR-based Reduction

Next, we define a new cut for the reduction algorithm based on the *Mixed Integer Rounding* formula [37], which is a generalization of Gomory's mixed integer cuts [27].

▶ **Definition 5** (Mixed Integer Rounding Cut). *Let $C : \sum_{i\in\mathcal{N}} a_i\ell_i \geq b$. The **Mixed Integer Rounding (MIR) Cut** of C with divisor $d \in \mathbb{Z}_{>0}$ is given by the constraint*

$$\sum_{i\in I_1} \left\lceil \frac{a_i}{d} \right\rceil \ell_i + \sum_{i\in I_2} \left(\left\lfloor \frac{a_i}{d} \right\rfloor + \frac{f(a_i/d)}{f(b/d)} \right) \ell_i \geq \left\lceil \frac{b}{d} \right\rceil, \tag{4}$$

where

$$I_1 = \{i \in \mathcal{N} \,:\, f(a_i/d) \geq f(b/d) \text{ or } f(a_i/d) \in \mathbb{Z}\},$$

$$I_2 = \{i' \in \mathcal{N} \,:\, f(a_{i'}/d) < f(b/d) \text{ and } f(a_{i'}/d) \notin \mathbb{Z}\},$$

and $f(\cdot) = \cdot - \lfloor \cdot \rfloor$. *To obtain a normalized version of the MIR cut, we multiply both sides of the constraint by* $(b \mod d)$.

The proof that applying MIR to a constraint is a valid procedure can be found in [37]. Similar to the Division-based reduction, it suffices to weaken non-falsified variables with a coefficient that is not a multiple of a_r before applying MIR in order to reduce the slack of the reason constraint to at most 0. This is shown in the following lemma.

▶ **Lemma 6.** *Let* ρ *be a partial assignment and* $C_{reason} : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ *a constraint propagating a literal* ℓ_r *to 1. Then, after weakening all non-falsified literal in* $W := \{i \in \mathcal{N} : \rho(i) \neq 0 \text{ and } a_r \nmid a_i\}$ *and applying Division or MIR on* C_{reason} *with* $d = a_r$*, the slack of the reduced reason is at most 0.*

Proof. After weakening all literals in W, the constraint C_{reason} becomes

$$a_r \ell_r + \sum_{j \in \mathcal{N} \setminus W} a_j \ell_j \geq \tilde{b} := b - \sum_{i \in W} a_i. \tag{5}$$

Its slack is

$$\mathit{slack}(C_{\text{reason}}, \rho) = a_r + \sum_{\substack{j \in \mathcal{N} \setminus W: \\ \rho(j) \neq 0}} a_j - \tilde{b} = a_r + \sum_{\substack{j \in \mathcal{N}: \\ \rho(j) \neq 0, a_r | a_j}} a_j - \tilde{b}.$$

Since weakening does not affect the slack, we have $\mathit{slack}(C_{\text{reason}}, \rho) < a_r$.

1. After applying the Division cut to (5) with $d = a_r$, the slack becomes

$$\mathit{slack}(C_{\text{reason}}, \rho) = 1 + \sum_{\substack{j \in \mathcal{N}: \\ \rho(j) \neq 0, a_r | a_j}} \left\lceil \frac{a_j}{a_r} \right\rceil - \left\lceil \frac{\tilde{b}}{a_r} \right\rceil \leq 1 + \sum_{\substack{j \in \mathcal{N}: \\ \rho(j) \neq 0, a_r | a_j}} \frac{a_j}{a_r} - \frac{\tilde{b}}{a_r} < \frac{a_r}{a_r} = 1. \tag{6}$$

Because C_{reason} contains only integer coefficients after applying the division rule, its slack is integer; hence, it must be at most 0.

2. Applying the MIR cut to (5) with $d = a_r$ results in the same slack as in (6). This is because all left-hand side coefficients in the slack computation are divisible by d, hence they fall into the index set I_1 and are transformed the same way as by the Division cut. ◀

3.6 Dominance Relationships

In this section, we would like to discuss briefly known dominance relationships between the different reduction techniques. The ultimate goal is to find a reduction technique that yields the strongest possible reason constraint to use in the resolution step of conflict analysis. The following lemma states the well-known fact that constraints from Saturation-based reduction are always at least as strong as the resolvents created during clausal conflict analysis as described in Section 3.1.

▶ **Lemma 7.** *Let* ρ *be a partial assignment and* $C_{reason} : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ *be a PB constraint which propagates literal* ℓ_r *to 1. Let* C'_{reason} *and* C''_{reason} *be the constraints obtained by clausal and Saturation-based reduction, respectively. Then* C''_{reason} *implies* C'_{reason}*.*

Proof. Under the current partial assignment ρ, the disjunctive clause reason is given by $\ell_r \bigvee_{j : \rho(\ell_j) = 0} \ell_j$, which can be linearized as

$$C'_{\text{reason}} \,:\, \ell_r + \sum_{j \,:\, \rho(\ell_j) = 0} \ell_j \geq 1.$$

Now let W be the set of all non-falsified literals, except ℓ_r. After weakening all literals in W and applying Saturation, we obtain the constraint

$$C''_{\text{reason}} : \min\{a_r, b - \sum_{i \in W} a_i\}\ell_r + \sum_{j:\rho(l_j)=0} \min\{a_j, b - \sum_{i \in W} a_i\}\ell_j \geq b - \sum_{i \in W} a_i.$$

As in the proof of Lemma 3, it holds that $\min\{a_r, b - \sum_{i \in W} a_i\} = b - \sum_{i \in W} a_i$. Now, after scaling C'_{reason} by $b - \sum_{i \in W} a_i$ we see that C''_{reason} has the same right-hand side as C'_{reason}, but smaller or equal coefficients on the left-hand side. ◀

In [26] the authors show that using Division instead of Saturation can be exponentially stronger, and that a single Saturation step can be simulated by an exponential number of Division steps.

The dominance of MIR cuts over Chvátal-Gomory cuts is a well-known fact in the MIP literature. The following lemma shows essentially the same result as in [15], but in the context of conflict analysis for pseudo-Boolean problems.

▶ **Lemma 8.** *Let ρ, C_{reason}, ℓ_r be given as in Lemma 7. Let C'_{reason} and C''_{reason} be the constraints obtained by Division-based and MIR-based reduction, respectively. Then C''_{reason} implies C'_{reason}.*

Proof. Let $C'_{\text{reason}}, C''_{\text{reason}}$ be constraints as in Definition 4 and 5, respectively, with divisor $d = a_r$. The constraints have the same right-hand side and the same coefficients for all literals ℓ_i with $i \in I_1$. For $i \in I_2$ the coefficient of literal ℓ_i in C'_{reason} is given by $\left\lceil \frac{a_i}{a_r} \right\rceil$ and in C''_{reason} by $\left\lfloor \frac{a_i}{a_r} \right\rfloor + \frac{f(a_i/a_r)}{f(b/a_r)}$. The coefficients of the literals ℓ_i in C'_{reason} are always greater than or equal to the coefficients in C''_{reason}, since by definition of the set I_2 it holds that $f(a_i/a_r)/f(b/a_r) < 1$. Therefore C''_{reason} implies C'_{reason}. ◀

As an example, consider the partial assignment $\rho = \{x_1 = 0, x_2 = 0, x_3 = 1\}$ and the constraint $C_{\text{reason}} : 2x_1 + 6x_2 + 10x_3 \geq 8$ which propagates variable x_3 to 1. Then the Division cut with divisor 10 is $x_1 + x_2 + x_3 \geq 1$. The MIR cut with the same division is $\frac{0.2}{0.8}x_1 + \frac{0.6}{0.8}x_2 + x_3 \geq 1$. Multiplying with $8 \mod 10 = 8$ gives the normalized MIR cut $2x_1 + 6x_2 + 8x_3 \geq 8$. The normalized MIR cut is stronger than the Division cut, which can be easily seen after scaling the Division cut by 8.

3.7 Practical Aspects of Weakening

While the evaluation of different weakening strategies is not the focus of this paper, we would like to discuss briefly some practical aspects of weakening literals. In our implementation we consider the following iterative weakening strategy: weaken free literals first followed by implied literals. We stop as soon as the resolvent is falsified under the remaining partial assignment. Intuitively, this order is motivated by the fact that free literals are not relevant for the propagation of literals in the reason constraint and do not affect the falsification of the conflict constraint.

However, the optimal order in which to weaken literals is not yet fully understood, and remains an open research question. Possible approaches include weakening literals in order of increasing or decreasing coefficient size. In [33] the authors conducted experiments with various weakening techniques, including partial weakening of literals and applying weakening on the conflict constraint, but the results did not yield a conclusive “best” weakening strategy.

A simple alternative is to weaken literals in a single sweep. For all three reduction algorithms, we can weaken the entire candidate set of literals as stated in Lemma 3 and Lemma 6 at once. Weakening literals all at once leads to a faster reduction algorithm since repeated slack computations are avoided and only one cut is applied in each iteration. However, this may result in the constraint being less informative due to unnecessary weakening of literals.

4 Experiments

It is well known in the SAT and PB communities that efficient conflict-driven search requires substantial amounts of very careful engineering. In this first work, our focus has been on importing and adapting the pseudo-Boolean conflict analysis to a MIP setting – which is a nontrivial task in its own right – leaving further optimizations as future work.

All techniques from Section 3 have been implemented in the open source MIP solver SCIP 8.0.3 [7] and we conducted extensive experiments to compare the different reduction techniques in isolation. Obtaining accurate performance results for MIP solvers requires a carefully designed experimental setup since even small changes to algorithms or the input data can have a large impact on the behavior and the performance of the solver. This is a well-known fact in the MIP literature known as *performance variability* [36]. To lessen the effects of performance variability and obtain a fair comparison of the different reduction techniques in the context of MIP solving, we use a fairly large and diverse testset of instances and different permutations of each instance, see, e.g., [25]. Our experiments were carried out on all pure 0–1 models from the MIPLIB 2017 collection [25]. After removing numerically unstable models (with the tag "numerics") our testset consists of 195 instances permuted by 5 different random seeds, giving a total of 975 measurements per run. For the remainder of this paper, we will refer to the combination of a model and a permutation as an *instance*. All experiments are conducted on a cluster with Intel Xeon Gold 6338 CPUs with a limit of 16GB of RAM.

It's worth noting that SCIP, along with its underlying LP solver, is based on floating-point arithmetic. Implementing a Pseudo-Boolean Optimization solver using a limited-precision LP-based branch-and-cut framework comes with some technical challenges which are discussed, e.g., in [5, 6]. From a theoretical standpoint, switching between reals and integers (rather than between limited and arbitrary precision) is straightforward:

All the algorithms presented in Section 3 can be naturally extended to the case of 0–1 constraints with coefficients that are real numbers instead of nonnegative integers. The Chvátal-Gomory procedure, MIR cutting, and coefficient tightening algorithm were originally designed for MIP with real coefficients.

However, in practice, floating-point arithmetic may cause numerical issues due to imprecise representations of real numbers and cancellation effects. To mitigate the risk of numeric instability, many components of SCIP, such as MIR-cut generation, utilize double-double precision arithmetic [18], which could be also employed in conflict analysis. Currently, for constraints generated in conflict analysis, we use the following standard techniques:

- We terminate conflict analysis if the coefficients of the constraints span too many orders of magnitude. Specifically, if the quotient of the largest to smallest coefficient is large (in our implementation, 10^6), we stop conflict analysis.
- We remove variables from the conflict constraint if their coefficients are too small (in our implementation, 10^{-9}), thereby relaxing the constraint slightly.

The latter threshold is a common default value for the zero tolerance in MIP solvers, and the former is a common modeling recommendation for MIP.

Table 1 Average percentage of true or unassigned literals that should be weakened to preserve the conflict analysis invariant. This experiment is conducted on the test set with 3 random seeds.

Setting	avg(%) literals weakened
Division	98.0
Saturation	99.7
MIR	97.3

4.1 Pre-Experiment: Weaken-All-At-Once vs. Weaken-Iteratively

As noted earlier, the weakening rule can be applied iteratively or in a single sweep. In preliminary experiments, we noticed that in almost all cases, most unassigned or true literals must be weakened to achieve the conflict analysis invariant that the resolved constraint has a negative slack. Table 1 summarizes this finding for different reduction techniques: Over all instances and all conflict analysis calls, an average between 97.3% (MIR) and 99.7% (Saturation) of all literals had to be weakened. Furthermore, for most instances both weakening variants did not lead to different execution paths.

In this case, weakening all literals at once avoids the overhead of iterative use of cuts and expensive slack computations. Consequently, we decided to always weaken all literals at once and apply the cut rule on the reason side only once for the remaining experiments presented in this paper.

4.2 Main Experiments: Comparing Different Reasoning Techniques

In the following, we compare all different reduction techniques from Section 3 to SCIP without any conflict analysis.

In our comparisons, we report for each technique the number of optimally solved instances, as well as the shifted geometric means of the number of processed nodes and the CPU time in seconds. The shifted geometric mean, a standard performance aggregator in the MIP literature, of the values $t_1, \ldots, t_n$ is defined as

$$\left(\prod_{i=1}^{n}(t_i + s)\right)^{1/n} - s, \tag{7}$$

for some $s > 0$. We set the shift s to 1 second for the CPU time and to 100 nodes for the number of nodes. Our base of comparison is SCIP without conflict analysis (“No Conflicts”). We report absolute values for the shifted means, and also quotients comparing them to our base setting. A factor below 1 means that a setting was faster (or needed less nodes), and a factor greater than 1 means that it was detrimental.

In Table 2 we report the results of our experiments. The table is split in four parts. We show results for “all” instances, as well as for three subsets of instances: (i) instances that are “affected” by conflict analysis, hence where the execution path of at least one setting differs from the others, (ii) “ $[100, \text{limit}]$ ” instances, which take at least 100 seconds to solve to optimality or hit the time limit and (iii) “all-optimal”, which are instances solved by all settings. Note that the number of nodes can only be fairly compared on the “all-optimal” subset, since the number of nodes when hitting a time limit is hard to interpret and hard to aggregate with the same statistics on instances that are solved to optimality.

The variant of SCIP with clausal conflict analysis is referred to as “Clausal-CA”. For a fair comparison of the different strategies, we disabled the upgrading of constraints to specialized types, i.e., all generated conflicts are treated as linear constraints, and further

Table 2 Main results.

	Setting	solved	time(s)	# nodes	time quot	nodes quot
all(975)	No Conflicts	394	656.75	784	1.0	1.0
	Clausal-CA	405	603.55	682	0.92	0.87
	Division	419	601.4	683	0.92	0.87
	MIR	420	599.37	677	0.91	0.86
	Saturation	418	599.76	692	0.91	0.88
affected(295)	No Conflicts	259	160.46	1096	1.0	1.0
	Clausal-CA	270	122.64	776	0.76	0.71
	Division	284	119.24	707	0.74	0.65
	MIR	285	118.29	700	0.74	0.64
	Saturation	283	118.09	735	0.74	0.67
[100, limit](218)	No Conflicts	182	667.14	2056	1.0	1.0
	Clausal-CA	193	486.45	1466	0.73	0.71
	Division	207	486.23	1345	0.73	0.65
	MIR	208	485.26	1336	0.73	0.65
	Saturation	206	491.98	1428	0.74	0.69
all-optimal(374)	No Conflicts	374	46.16	320	1.0	1.0
	Clausal-CA	374	40.58	259	0.88	0.81
	Division	374	40.75	244	0.88	0.77
	MIR	374	40.40	241	0.88	0.75
	Saturation	374	40.24	246	0.87	0.77

only generated one conflict per call. Conversely, we accept PB reasoning conflicts only if the number of nonzeros is less than 15% of the original problem variables, as in the default clausal implementation in SCIP. Our preliminary experiments confirmed that in our implementation, it is indeed detrimental to accept too-long conflicts. We did, however, add a fallback strategy, of applying weakening on the conflict constraint if the constraints are too long. This happens for about 9% of the instances.

We observe that all conflict analysis variants solved more instances than SCIP without conflict analysis, needed significantly less nodes on the all-optimal set, and less time on all four instance sets. Note that on average, the time spent in conflict analysis is only about 0.1% of the total run time. The three PB conflict analysis variants could solve more instances than the clausal variant, and needed significantly less nodes. The difference in time was less pronounced.

The performance of the PB conflict analysis variants is quite similar in all three cases. Nevertheless, MIR-based reduction could solve the most instances and needed the least nodes on the all-optimal set. When looking at the seemingly identical time-wise performance in more detail, it turns out that MIR also slightly improves on the other settings in this measure. There are 104 instances for which the path differs between Saturation-based resolution and MIR-based resolution and MIR was on average 1.1% faster on those. There are 86 instances for which the path differs between Division-based resolution and MIR-based resolution and MIR was on average 3.6% faster on those. Consequently, we decided to concentrate on MIR-based resolution for our next statistic.

Ultimately, the purpose of conflict constraints is to restrict the future search space by propagating literal assignments and pruning the search tree. Hence we analyzed how many conflicts each of the methods generates in shifted geometric mean, how large these conflicts are on average, and how many of them lead to propagations down the road. Table 3 shows

Table 3 Shifted geometric mean of number of conflicts, average percentage of conflict constraints that propagate at least once and average length of learned conflicts.

Setting	mean # conflicts	avg % prop. conflicts	avg # literals
Clausal-CA	290.77	34.54	82.45
MIR	169.61	58.54	80.20

the results on the set of all instances that have a search tree of at least 100 nodes (to get a decent chance of conflict generation and propagation) and for which at least one conflict was generated with one of the methods. We consider only instances where the two settings have the same execution path. We observe that our MIR-based conflict analysis generated about a third less conflicts, but at the same time, they are much more likely to propagate: for the classic clausal conflict analysis of SCIP, about a third of the generated conflicts are used for propagation later on, while for our MIR-based variant, slightly more than half (58.54%) of all conflicts propagate at least once. At the same time, MIR-based conflicts are about the same size as clausal conflicts.

At first glance, this might appear as a contradiction, given that, as a rule of thumb, shorter conflicts tend to propagate more often and one might expect similar-sized conflicts to be similarly likely to propagate. Note, however, that the conflicts are of a quite different nature in the two cases. On the one hand, clausal conflicts are always logic clauses that only propagate when all but one literal are assigned. On the other hand, MIR-based conflicts are general pseudo-Boolean constraints, which might propagate some assignments (of literals with large coefficients) even when a majority of literals are still unassigned. This goes nicely together with the above observation that the reduction in the number of nodes is more pronounced than the reduction in runtime. As a final remark, integrating PB conflict analysis in a production-grade MIP solver would require substantially more work, but should also be expected to provide substantial further improvements measured in wallclock time.

5 Conclusion

In this work, we study how to integrate pseudo-Boolean conflict analysis for 0–1 integer linear programs into a MIP solving framework. In contrast to standard MIP conflict analysis, the pseudo-Boolean method operates directly on the linear constraints, rather than on clauses extracted from these constraints, and this makes it exponentially stronger in terms of reasoning power. Viewing PB conflict analysis from a MIP perspective is also helpful since it provides a view of the algorithm as a sequence of linear combinations and cuts, and we use this to strengthen the pseudo-Boolean conflict analysis further by developing a new conflict analysis method using the powerful mixed integer rounding (MIR) cuts.

We have made a first proof-of-concept implementation of our new pseudo-Boolean conflict analysis method, as well as methods from the PB literature based on saturation [34] and division [23], in the open-source MIP solver SCIP, and have run experiments on 0–1 ILP instances from MIPLIB 2017 comparing the different methods with each other and with standard clause-based MIP conflict analysis. We find that solving 0–1 ILPs with MIR-based pseudo-Boolean conflict analysis performs better than other methods, not only in the sense that it reduces the size of the search tree, but also in that our implementation can beat the highly optimized MIP conflict analysis currently used in SCIP in terms of actual running time. In our opinion, this demonstrates convincingly that pseudo-Boolean conflict analysis in MIP is a research direction that should be worth pursuing further, and that similar proof-of-concept studies could also be relevant to investigate for other combinatorial solving paradigms such as constraint programming.

As already noted above, an obvious direction of future work is to provide a more carefully engineered version of pseudo-Boolean conflict analysis that could deliver more fully on the potential for improved performance identified by our experiments. In addition to optimizing the existing code, however, it would be valuable to develop a better understanding of how and why the conflict analysis works and of ways in which the reasoning could be improved.

Pseudo-Boolean conflict analysis alternates between weakening constraints (to eliminate seemingly less relevant variables) and strenghtening them by applying cut rules (to get tighter propagation on the variables that remain). The interplay between these two operations is quite poorly understood even for pseudo-Boolean solvers, and so both PB solvers and MIP solvers could gain from a careful study of how to strike the right balance. Since PB conflict analysis can be performed with several different reduction methods, and since different reduction methods can be employed independently in consecutive steps in one and the same conflict analysis, it would also be good to be able to assess the quality of constraints derived during conflict analysis, so as to select the most promising candidate at each step to pass on to the next step in the conflict analysis.

Arguably the most interesting research question, though, is whether pseudo-Boolean conflict analysis could be extended beyond 0–1 ILPs to 0–1 mixed linear problems, and/or to general integer linear programs. It is worth noting that the latter has been attempted in [32, 41], but so far with quite limited success. It is clear that the algorithms presented in this paper *cannot* work for 0–1 mixed LPs or general ILPs if generalized in the obvious, naive way, and so additional, new ideas will be needed.

References

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

2 Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. Available at `https://opus4.kobv.de/opus4-zib/files/1112/Achterberg_Constraint_Integer_Programming.pdf`.

3 Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

4 Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.

5 Timo Berthold, Stefan Heinz, and Marc Pfetsch. Solving Pseudo-Boolean problems with SCIP, 2008.

6 Timo Berthold, Stefan Heinz, and Marc E Pfetsch. Nonlinear Pseudo-Boolean optimization: relaxation or propagation? In *Theory and Applications of Satisfiability Testing-SAT 2009: 12th International Conference, SAT 2009, Swansea, UK, June 30-July 3, 2009. Proceedings 12*, pages 441–446. Springer, 2009.

7 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The SCIP Optimization Suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.

8 Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.

9 Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

10 AL Brearley, Gautam Mitra, and H Paul Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical programming*, 8:54–83, 1975.

11 Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 233–350. IOS Press, 2nd edition, February 2021.

12 Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, March 2005. Preliminary version in *DAC '03*.

13 Vasek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973.

14 William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

15 Gérard Cornuéjols and Yanjun Li. Elementary closures for integer programs. *Operations Research Letters*, 28(1):1–8, 2001. `doi:10.1016/S0167-6377(00)00067-5`.

16 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

17 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

18 Theodorus Jozef Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

19 Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1–4):26–55, October 2021. Preliminary version in *CPAIOR '20*.

20 Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3750–3758, February 2021.

21 Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.

22 Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström, and Marc Vinyals. Using combinatorial benchmarks to probe the reasoning power of pseudo-Boolean solvers. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 75–93. Springer, July 2018.

23 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.

24 Matthew L Ginsberg. Dynamic backtracking. *Journal of artificial intelligence research*, 1:25–46, 1993.

25 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*, 13:443–490, 2021. `doi:10.1007/s12532-020-00194-3`.

26 Stephan Gocht, Jakob Nordström, and Amir Yehudayoff. On division versus saturation in pseudo-Boolean solving. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*, pages 1711–1718, August 2019.

27 Ralph E. Gomory. An algorithm for the mixed integer problem. Technical Report P-1885, The RAND Corporation, June 1960.

28 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

29 John N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12(1):217–239, December 1988.

30 John N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6(1):271–286, March 1992.

31 Yuejun Jiang, Thomas Richards, and Barry Richards. Nogood backmarking with min-conflict repair in constraint satisfaction and optimization. In *Principles and Practice of Constraint Programming: Second International Workshop, PPCP'94 Rosario, Orcas Island, WA, USA, May 2–4, 1994 Proceedings*, pages 21–39. Springer, 1994.

32 Dejan Jovanovic and Leonardo de Moura. Cutting to the chase solving linear integer arithmetic. *Journal of Automated Reasoning*, 51(1):79–108, June 2013. Preliminary version in *CADE-23*.

33 Daniel Le Berre, Pierre Marquis, and Romain Wallon. On weakening strategies for PB solvers. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, July 2020.

34 Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, July 2010.

35 Vincent Liew, Paul Beame, Jo Devriendt, Jan Elffers, and Jakob Nordström. Verifying properties of bit-vector multiplication using cutting planes reasoning. In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD '20)*, pages 194–204, September 2020.

36 Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory driven by influential applications*, pages 1–12. INFORMS, 2013.

37 Hugues Marchand and Laurence A Wolsey. Aggregation and mixed integer rounding to solve mips. *Operations research*, 49(3):363–371, 2001.

38 João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

39 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, July 2014.

40 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

41 Robert Nieuwenhuis. The IntSat method for integer linear programming. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP '14)*, volume 8656 of *Lecture Notes in Computer Science*, pages 574–589. Springer, September 2014.

42 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

43 Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, 98-D(6):1121–1127, June 2015.

44 Tuomas Sandholm and Robert Shields. Nogood learning for mixed integer programming. In *Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization, Montréal*, volume 20, pages 21–22, 2006.

45 Buser Say, Jo Devriendt, Jakob Nordström, and Peter Stuckey. Theoretical and experimental results for planning with learned binarized neural network transition models. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 917–934. Springer, September 2020.

46 Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, March 2006. Preliminary version in *DATE '05*.

47 Richard M Stallman and Gerald J Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, 9(2):135–196, 1977.

48 Jakob Witzig, Timo Berthold, and Stefan Heinz. Computational aspects of infeasibility analysis in mixed integer programming. Technical Report 19-54, ZIB, Takustr. 7, 14195 Berlin, 2019.