Improving Reduction Techniques in Pseudo-Boolean Conflict Analysis

Orestis Lomis
DTAI, KU Leuven, Belgium

Jo Devriendt
Nonfiction Software, Leuven, Belgium

Hendrik Bierlee
DTAI, KU Leuven, Belgium

Tias Guns
DTAI, KU Leuven, Belgium

Abstract

Recent pseudo-Boolean (PB) solvers leverage the cutting planes proof system to perform SAT-style conflict analysis during search. This process learns implied PB constraints, which can prune later parts of the search tree and is crucial to a PB solver's performance. A key step in PB conflict analysis is the *reduction* of a reason constraint, which caused a variable propagation that contributed to the conflict. While necessary, reduction generally makes the reason constraint less strong. Consequently, different approaches to reduction have been proposed, broadly categorised as division- or saturation-based, with the aim of preserving the strength of the reason constraint as much as possible.

This paper proposes two novel techniques in each reduction category. We theoretically show how each technique yields reason constraints which are at least as strong as those obtained from existing reduction methods. We then evaluate the empirical effectiveness of the reduction techniques on hard knapsack instances and the most recent PB'24 competition benchmarks.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, Pseudo-Boolean Reasoning, Conflict Analysis

Digital Object Identifier 10.4230/LIPIcs.SAT.2025.21

Supplementary Material *Dataset (Experimental Results)*: `https://github.com/ML-KULeuven/SAT25_PB_reductions_experiments` [19]
archived at `swh:1:dir:650c2f9b37926504a59a11a6dbf319620b311dc3`

Funding This research was partly funded by the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt).

1 Problem setting

Although the Boolean satisfiability (SAT) problem is NP-complete [7], in recent decades *conflict-driven clause learning (CDCL)* solvers [20, 1] routinely solve problems with up to millions of variables. However, the resolution proof system on which they are based may require exponential solve-time for some problems, such as the pigeonhole problem [14]. For this reason, higher-level proof systems pose a promising alternative, such as the cutting planes proof system [8]. Cutting planes form the underlying proof system of conflict-driven *pseudo-Boolean (PB)* constraint solvers, where we especially focus on PB constraints that are (weighted) linear inequalities over Boolean variables. The cutting planes proof system generalises the resolution proof system by using such PB constraints instead of clauses, which allow for more powerful reasoning. In theory this means that PB solvers can solve problems like the pigeonhole problem in polynomial time, where CDCL-based solvers would need exponential time. The effectiveness of good PB reasoning has been shown in the most recent PB competition of 2024 [24], where PB solvers such as *RoundingSat* [12] and its offspring *Exact* [11] show top performance with their native implementation of the cutting planes proof system. *Exact* and *RoundingSat* are not only used as stand-alone solvers, but also as PB oracles in other PB solvers such as Hybrid-CASHWMaxSATDisjCadSP+Exact [22], mixed-bag [17] and IPBHS [16].

However, there still lies a difficulty for PB solvers to use the improved reasoning power of the cutting planes proof system effectively. In CDCL, the resolution of two conflicting clauses can simply be done by taking the union of two conflicting clauses and leaving out the propagated variable. While resolution can be generalised, PB solvers require an additional so-called *reduction step* to ensure the eventual learned constraint prevents the original conflict [5]. Additionally, the stronger the learned constraint the more pruning it will be able to do, hence the need for reduction techniques that keep the reason constraint as strong as possible. Consequently, various types of reduction have been investigated [12, 18, 21]. In this work we theoretically motivate and investigate four new reduction methods that are at least as strong as existing reduction methods.

Our contributions. First, we propose two novel variants of division-based reduction, namely Weaken Superfluous (WS) and Anti-Weakening of non-falsifieds (AW). Both exploit the rounding behaviour of the division operation in division-based reduction. We prove how the reduced constraints using these techniques are at least as strong as the ones obtained from using standard division-based reduction. Furthermore, we show how after applying the WS technique, the division operation is equivalent to the Mixed-Integer Rounding (MIR) operation, even though generally MIR dominates division [21]. Secondly, we propose two novel variants of a saturation-based reduction technique known as Multiply and Weaken (MW) [18], which we call Multiply and Weaken Direct (MWD) and Multiply and Weaken Indirect (MWI). These variants hybridize saturation- and division-based reduction by applying the former when favourable, using the latter as fallback. Then, we show that MWD with MWI produces constraints at least as strong as those from MWD alone. Finally, experiments empirically evaluate how the theoretically stronger reduction techniques impact solver performance on crafted and competition PB problems.

The organisation of the paper is as follows. In Section 2 we review the basics of PB solving, starting from CDCL and working our way to the current state-of-the-art division-based reduction method in *RoundingSat*. In Section 3 and Section 4 we introduce our novel techniques for division- and saturation-based reduction methods respectively. Section 5 contains the results from our experiments. In Section 6 we conclude our finding and discuss future work.

2 Preliminaries

We use notation and terminology from [10]. The term *pseudo-Boolean (PB) constraint* refers to a 0-1 linear inequality. We identify 1 with *true* and 0 with *false*. A *literal* ℓ denotes either a variable x or its negation $\bar{x} = 1 - x$. We assume w.l.o.g. that all constraints $\sum_i c_i \ell_i \geq \delta$ are written in *normalized form*, where literals ℓ_i are over pairwise distinct variables, coefficients c_i are positive integers, and δ is a positive integer called the *degree*. For a constraint C, *lits*(C) denotes its set of literals and *coeff*(ℓ_i, C) denotes the coefficient of literal ℓ_i. A PB constraint C with degree 1 is a *clause*.

The *(partial) assignment* ρ is an ordered set of literals over pairwise distinct variables. A literal ℓ is *assigned to true* by an assignment ρ if $\ell \in \rho$, *assigned to false* or *falsified* if $\bar{\ell} \in \rho$, and is *unassigned* otherwise. We define the *slack* σ of a constraint $C = \sum_i c_i \ell_i \geq \delta$ under a partial assignment ρ as:

$$\mathit{slack}(C, \rho) = \left(\sum_{\ell_i \in \mathit{lits}(C), \bar{\ell}_i \notin \rho} c_i \right) - \delta \tag{1}$$

In other words, the slack measures how far ρ is from falsifying the constraint. Then, we say that ρ *falsifies* C if $\mathit{slack}(C, \rho) < 0$. A *pseudo-Boolean formula* φ is a set of PB constraints. An assignment ρ is a *solution* to φ if ρ satisfies all constraints in φ. A formula is *satisfiable* if it has a solution.

2.1 Conflict-Driven Pseudo-Boolean Search

Conflict-driven PB solving generalizes the CDCL algorithm for SAT, but uses PB constraints instead of clauses. The state of a PB solver can be represented by a ψ and ρ, where ψ is a set of constraints called the *constraint database*. Initially, ψ is the input formula φ and ρ is the empty set $\{\}$.

Given a solver state, the search loop starts with a *propagation* phase, which checks for any constraint $C \in \psi$ whether it is falsified:

$$\mathit{slack}(C, \rho) < 0, \tag{2}$$

or whether a literal ℓ_i in C with coefficient c_i, where ℓ_i has not yet been assigned by ρ, is implied by C under ρ:

$$\mathit{slack}(C, \rho) - c_i < 0 \text{ with } \ell_i \notin \rho, \bar{\ell}_i \notin \rho. \tag{3}$$

If condition (3) holds, C is falsified by $\rho \cup \bar{\ell}_i$, so ℓ_i is implied by C under ρ. For an assignment ρ we write ℓ_i/C to denote that C is the *reason* for the propagation of ℓ_i, and also use the notation $C = \mathit{reason}(\ell_i, \rho)$. Each propagation can enable new propagations, continuing the propagation phase until condition (3) does not hold for any constraint in the database ψ or until condition (2) holds for at least one. Note that unlike clauses, a single PB constraint can propagate multiple literals, even at different propagation phases.

If condition (2) holds for some constraint, it is considered a *conflict* and the constraint is denoted as the *conflict constraint.* On conflict, the solver enters a *conflict analysis* phase. During this phase, the solver derives a *learned constraint* which is a logical consequence of the current set of reason constraints combined with the conflict constraint. Crucially, the learned constraint must propagate a literal at some earlier search depth, hence preventing the current conflict from occurring again. Then, this learned constraint is added to ψ, after which the solver *backjumps* to a sufficient early search depth.

Alternatively, if no conflict is detected, the solver extends ρ by making a heuristic *decision* to assign some currently unassigned variable. If ℓ_i is a decision then it has no associated reason constraint, which we denote by $\ell_i/\cdot$.

The PB solver reports unsatisfiability whenever it learns a constraint equivalent to the trivial inconsistency $0 \geq 1$. If propagation does not lead to a conflict and all variables have been assigned, the solver reports that the input formula is satisfiable.

2.2 PB Conflict Analysis

In this subsection we will go into more detail about the conflict analysis phase of PB solvers, where the following operations are used [2, 4]:

Addition.

$$\frac{\Sigma c_i\ell_i \geq \delta \quad \Sigma c'_i\ell'_i \geq \delta'}{\Sigma(c_i\ell_i + c'_i\ell'_i) \geq (\delta + \delta')} \text{ Add} \tag{4}$$

Where we implicitly assume that the result is rewritten in normalised form.

▶ **Example 1.** Addition of the constraint $x+\bar{y} \geq 2$ with $y+z \geq 1$ yields $x+y+\bar{y}+z \geq 3$, which normalises to $x + z \geq 2$ by *cancelling* literals y and $\bar{y}$, where $y + \bar{y} = 1$.

Division.

$$\frac{\Sigma c_i\ell_i \geq \delta}{\Sigma\lceil \frac{c_i}{d} \rceil \ell_i \geq \lceil \frac{\delta}{d} \rceil} \text{ Div. by } d \in \mathbb{N}_0 \tag{5}$$

Multiplication.

$$\frac{\Sigma c_i\ell_i \geq \delta}{\Sigma \mu c_i\ell_i \geq \mu\delta} \text{ Mul. by } \mu \in \mathbb{N}_0 \tag{6}$$

Saturation.

$$\frac{\Sigma c_i\ell_i \geq \delta}{\Sigma\ min(c_i, \delta)\ell_i \geq \delta} \text{ Sat.} \tag{7}$$

Weakening.

$$\frac{c\ell + \Sigma c_i\ell_i \geq \delta}{(c-m)\ell + \Sigma c_i\ell_i \geq \delta - m} \text{ Wkn. } \ell \text{ by } m \in \mathbb{N}_0 \tag{8}$$

Weakening is *partial* when $m < c$, and *full* when $m = c$.

Anti-weakening

$$\frac{c\ell + \Sigma c_i\ell_i \geq \delta}{(c+m)\ell + \Sigma c_i\ell_i \geq \delta} \text{ Anti-Wkn. } \ell \text{ by } m \in \mathbb{N}_0 \tag{9}$$

Using these operations, PB solvers often implement different variants of conflict analysis, [6, 12, 3, 11]. We give a general outline in Algorithm 1 [21]. Conflict analysis starts from the conflict constraint C_{co}. We call the last literal of the current assignment, the reason literal ℓ_r. If it was not propagated, or $\bar{\ell_r} \notin \mathit{lits}(C_{co})$, then it did not contribute to the conflict, so it can be removed from the assignment ρ and we continue with the literal propagated just before it. If it is propagated and $\bar{\ell_r} \in \mathit{lits}(C_{co})$, then we should replace $\bar{\ell_r}$ with its reason constraint $C_{rsn} = \mathit{reason}(\ell_r, \rho)$ by addition of the two constraints [15, 4], which requires *reducing* the reason constraint as explained below. When the new C_{co} is propagating we have a learned constraint, and we can exit the conflict analysis phase.

Algorithm 1 analyzeConflict.

```
Input: Conflict constraint $C_{co}$, falsifying partial assignment $\rho$
Output: Learned constraint $C_l$
1 while $C_{co}$ is not propagating do
2     $\ell_r \leftarrow$ last literal of assignment $\rho$
3     if $\ell_r$ is propagated $\wedge \bar{\ell}_r \in lits(C_{co})$ then
4         $C_{rsn} \leftarrow reason(\ell_r, \rho)$
5         $(C_{red}, C_{co}) \leftarrow reduce(C_{rsn}, C_{co}, \ell_r, \rho)$
6         $C_{co} \leftarrow C_{red} + C_{co}$
7     $\rho \leftarrow \rho \setminus \{\ell_r\}$
8 return $C_{co}$
```

To preserve the conflict during addition of C_{rsn} and C_{co}, there are two key requirements which make conflict-driven learning non-trivial for PB solving: The first one is that (i) adding C_{rsn} to C_{co} should indeed eliminate $\bar{\ell}_r$. Secondly, the learned constraint must propagate at an earlier stage, therefore (ii) it needs to remain conflicting under the current assignment. Since these requirements generally do not hold [4, Chapter 7], we need a *reduction step*, where $(C_{red}, C_{co}) = reduce(C_{rsn}, C_{co}, \ell_r, \rho)$ [13] such that the requirements are enforced. A sufficient condition for (i) is that the reduced reason has the same coefficient for $\bar{\ell}_r$ as the conflict constraint has for ℓ_r. A sufficient condition for (ii) is that after reduction, the reduced reason has slack 0 or less, since $slack(C_{co}, \rho) < 0$ by definition and slack is *subadditive* [12]: adding two constraints yields a constraint with a slack at most the sum of the slacks of the two original constraints. Hence formally we have the following requirements:

Cancelling Coefficients (Requirement 1) $coeff(\ell_r, C_{red}) = coeff(\bar{\ell}_r, C_{co})$
Negative Slack Condition (Requirement 2) $slack(C_{red}, \rho) \leq 0$

Our work focuses on this reduction step. To compare between two outcomes of a reduction, we say that C is *stronger* than C' when C implies C' (i.e. every satisfying variable assignment to C is also a satisfying assignment to C') but not the other way around. C is *rationally stronger* than C' when the former implies the latter and not the other way around, considering assignments of rational values between the closed interval $[0, 1]$ to the variables. We say reduction method A dominates reduction method B, with reduced constraints C^A_{red} and C^B_{red} respectively, when for any input C^A_{red} rationally implies C^B_{red}.

We already mentioned that slack is subadditive under addition. Additionally, slack remains unchanged when weakening a non-falsified literal or anti-weakening a falsified literal; slack increases when weakening a falsified literal or anti-weakening a non-falsified literal; slack is multiplied by m when multiplying a constraint by m. Hence, to decrease the slack of an individual non-conflicting reason constraint, at least one division or saturation step is necessary. Saturation-based reduction was the first successful implementation of cutting planes for PB solving [3], however most state-of-the-art solvers opt for division-based reduction [12, 17, 11, 22]. We will now look more in depth at division-based reduction.

2.3 *RoundingSat*-style Reduction

We describe the division-based reduction approach proposed by *RoundingSat* [12], which consists of three steps.

Weaken Non-Divisible Non-Falsifieds (Step 1) In the first step, we weaken all non-falsified literals that have a coefficient not divisible by c_{rsn}, the reason literal ℓ_r's coefficient, so that all non-falsified literals become divisible by c_{rsn}. In the original *RoundingSat* paper, these literals were fully weakened, but in the most recent implementation of *RoundingSat*[1], non-falsified literals with coefficient c_i are partially weakened by $c_i \bmod c_{rsn}$, i.e. to the largest multiple of c_{rsn} smaller than c_i. This is a less aggressive version of weakening also discussed in [18].

Divide (Step 2) In the second step, we divide the resulting constraint by c_{rsn} and round up all coefficients as per Equation (5). Since the previous weakening guaranteed that all non-falsified literals are divisible by c_{rsn}, we have $\mathit{coeff}(\ell_r, C_{rsn}) = 1$ and no non-falsified literals are rounded up during division, which would have increased the slack. Furthermore, the authors have proven after this division step the slack is at most 0, because the divisor is larger than the slack [12, Proposition 3.1], hence satisfying Requirement 2. From their work it follows that:

▶ **Corollary 2.** *Given a constraint C_{rsn} with slack σ and a divisor $d \in \mathbb{N}_0$, if the coefficients of all non-falsified literals are divisible by d, then the slack after division is $\lfloor \frac{\sigma}{d} \rfloor$.*

Multiply (Step 3) In the third step, given that ℓ_r's coefficient is now 1, Requirement 1 can be satisfied by multiplying the reason constraint by the coefficient of the negated reason literal in the conflict c_{co}.

▶ **Example 3.** Given the reason constraint $C_{rsn} = x + 3y + 3z + 5w \geq 6$, a conflict constraint $C_{co} = 4y + 4\bar{w} \geq 4$, an assignment $\rho = \{\bar{x}/\cdot, \bar{y}/\cdot, z/C_{rsn}, w/C_{rsn}\}$, reason literal $\ell_r = w$ and divisor $d = \mathit{coeff}(\ell_r, C_{rsn}) = 5$.

Step 1 of this method is to weaken the non-divisible non-falsified literals. So z is weakened by 3. Then in Step 2, the constraint is divided by 5:

$$\cfrac{\cfrac{x + 3y + 3z + 5w \geq 6}{x + 3y + 5w \geq 3}\text{ Wkn. } z \text{ by } 3}{x + y + w \geq 1}\text{ Div. by } 5 \tag{10}$$

Note that Requirement 2 is now already satisfied. To satisfy Requirement 1 we need to perform Step 3 by multiplying $1x + 1y + 1w \geq 1$ by $\mathit{coeff}(\bar{\ell}_r, C_{co}) = 4$ to get the reduced reason $C_{red} = 4x + 4y + 4w \geq 4$, which satisfies Requirements 1 and 2. To obtain a new learned constraint we then add the reduced reason to C_{co} and get $C_l = 4x + 4y \geq 4$. This learned constraint will propagate y as soon as $\bar{x}$ is decided, thereby preventing the conflict, and possibly preventing similar conflicts in later search.

3 Division-Based Reduction Variants

In this section we propose two new variants of division-based reduction, based on the above. Notice that since the reduced constraint C_{red} must be implied by the reason constraint C_{rsn}, C_{red} is at best as strong as C_{rsn}. Hence, our goal is to design a reduction method such that C_{red} remains as strong as possible.

In Sections 3.1 and 3.2, we will exploit Corollary 2 and the behaviour of rounding during division-based reduction in two novel techniques. For each technique we show that Requirements 1 and 2 still hold and that each technique dominates division-based reduction without the technique. In Section 3.3, we show how the techniques interact and can be combined.

[1] This recent version also participated in the last PB competition [24].

3.1 Weaken Superfluous (WS)

When dividing a constraint with degree δ by divisor d, the post-division degree is $\delta' = \lceil \frac{\delta}{d} \rceil$. Due to the upward rounding, we can often lower δ by some amount θ without changing δ'. Specifically, $\delta' = \lceil \frac{\delta}{d} \rceil = \lceil \frac{\delta-\theta}{d} \rceil$ for any $\theta \leq (\delta - 1) \bmod d$. After Step 1 (weakening of non-divisible non-falsifieds) of the above division-based reduction, we set θ maximally to get $\theta = (\delta - 1) \bmod d$. We define a *superfluous* literal as:

▶ **Definition 4** (Superfluous Literal). *When dividing a constraint C by d, a literal ℓ_s with coefficient c_s is superfluous when $0 < c_s \bmod d \leq \theta$ with $\theta = (\delta - 1) \bmod d$.*

Hence, we can weaken a superfluous literal ℓ_s by $c_s \bmod d$ without altering the degree δ' obtained after Step 2. But now, ℓ_s is rounded down in Step 2, which makes the reduced constraint stronger. Thus we propose the WS reduction where we iteratively weaken superfluous literals until there are no more left. Since the non-falsifieds have already been weakened to be divisible, the superfluous literals are always falsified. Therefore the reason literal ℓ_r is never superfluous and its coefficient will not change compared to *RoundingSat*-style reduction. Thus Requirement 1 will still be satisfied after Step 3 as when applying this reduction.

▶ **Example 5.** Continuing with Example 3, after Step 1, we can weaken by a total of $\theta = (\delta - 1) \bmod d = (3 - 1) \bmod 5 = 2$ before Step 2. Thus x is a superfluous literal, so we weaken it as well.

$$\cfrac{\cfrac{\cfrac{x + 3y + 3z + 5w \geq 6}{x + 3y + 5w \geq 3}\text{Wkn. } z \text{ by } 3}{3y + 5w \geq 2}\text{Wkn. } x \text{ by } 1}{y + w \geq 1}\text{Div. by } 5 \tag{11}$$

The final constraints from Equations (10) and (11) now both satisfy Requirement 2 and after multiplication both will also satisfy Requirement 1, but the latter is stronger than the former.

We now prove division-based reduction with WS dominates division-based reduction without WS.

▶ **Proposition 6.** *Let $d \in \mathbb{N}_0$ be some divisor. Let $C = c_s\ell_s + \sum c_i\ell_i \geq \delta$ be a constraint with ℓ_s a superfluous literal, so $c_s \bmod d \leq \theta$. Let C' be the constraint after division by d. Let C^{WS} be the constraint after weakening the superfluous literal ℓ_s to the nearest divisible integer (so by $c_s \bmod d$), and then division by d.*

C^{WS} implies C' and the slack of C^{WS} is at most that of C'.

Proof. We know that after division $C' = \lceil \frac{c_s}{d} \rceil \ell_s + \sum \lceil \frac{c_i}{d} \rceil \ell_i \geq \lceil \frac{\delta}{d} \rceil$ and $C^{WS} = \lceil \frac{c_s - c_s \bmod d}{d} \rceil \ell_s + \sum \lceil \frac{c_i}{d} \rceil \ell_i \geq \lceil \frac{\delta - c_s \bmod d}{d} \rceil$. Note that $\lceil \frac{c_s - c_s \bmod d}{d} \rceil = \lceil \frac{c_s}{d} \rceil - 1$ and that $\lceil \frac{\delta - c_s \bmod d}{d} \rceil = \lceil \frac{\delta}{d} \rceil$ (since $c_s \bmod d \leq \theta$). Hence, C^{WS} only differs from C' in that the coefficient of ℓ_s is rounded down. This means *antiweaken*$(C^{WS}, \ell_s, 1) = C'$ and thus C^{WS} implies C'. ◀

From the proof, *antiweaken*$(C^{WS}, \ell_s, 1) = C'$, so after division, the slack of the constraint where a superfluous literal is weakened is at most that of the non-weakened variant. So weakening superfluous literals preserves whether Requirement 2 is satisfied after division.

3.1.1 Link to Mixed Integer Rounding (MIR)

It has been proposed to replace the division operation (Step 2) in division-based reduction by the Mixed Integer Rounding (MIR) operation [21]:

Mixed Integer Rounding (MIR).

$$\frac{\Sigma_i c_i \ell_i \geq \delta}{\sum_{\ell \in I_1} \lceil \frac{c_i}{d} \rceil \ell_i + \sum_{\ell_j \in I_2} (\lfloor \frac{c_j}{d} \rfloor + \frac{c_j \bmod d}{(\delta - 1) \bmod d + 1}) \ell_j \geq \lceil \frac{\delta}{d} \rceil} \text{ MIR by } d \in \mathbb{N}_0 \tag{12}$$

with

$$I_1 = \{\ell_i \mid c_i \bmod d > (\delta - 1) \bmod d \vee c_i \bmod d = 0\},$$

$$I_2 = \{\ell_j \mid 0 < c_j \bmod d \leq (\delta - 1) \bmod d\},$$

To obtain normalised PB constraints with integer coefficients, MIR is followed by multiplication by $((\delta - 1) \bmod d) + 1$.

It was shown that division-based reduction using the MIR operation in Step 2 dominates division-based reduction with the division operation [21]. However, we can show that if there are no superfluous literals, the two reduction variants are equivalent. Consequently, after weakening superfluous literals, using MIR in Step 2 provides no advantage anymore compared to using division.

▶ **Proposition 7.** *Let $C = \sum c_i \ell_i \geq \delta$ be a constraint and $d \in \mathbb{N}_0$ a divisor such that none of the literals ℓ_i are superfluous in C. Let C^{DIV} and C^{MIR} be the constraints obtained by applying the division and the MIR operation with d, respectively. Then $C^{DIV} = C^{MIR}$.*

Proof. As no literals are superfluous, for all literals it holds by Definition 4 that $c_i \bmod d \geq \theta$ or $c_i \bmod d = 0$. Hence, $I_2 = \emptyset$. In that case, Equation (12) simplifies to Equation (5), so $C^{DIV} = C^{MIR}$. ◀

This means that, when there are no superfluous literals (e.g. after removing them with WS) division and MIR are equivalent.[2] This is not the case when there are superfluous literals in the constraint. E.g. WS followed by division on constraint $x + 2y \geq 2$ with divisor 2 is stronger than just MIR on the same constraint. The opposite can also be true, e.g. for constraint $x + y + 2z \geq 2$ with divisor 2. There could be other cases where just the MIR operation, without doing WS first, may be stronger than WS combined with the division operation. Further analysis is left for future work.

3.2 Anti-Weaken Anti-Superfluous (AW)

When dividing a constraint with slack σ by divisor d, the post-division slack is $\lfloor \frac{\sigma}{d} \rfloor$ according to Corollary 2. Due to downward rounding, we can often increase σ by some amount κ without changing $\lfloor \frac{\sigma}{d} \rfloor$. Specifically, $\lfloor \frac{\sigma}{d} \rfloor = \lfloor \frac{\sigma + \kappa}{d} \rfloor$ for any $0 \leq \kappa \leq (d - \sigma - 1) \bmod d$. During Step 1 we set κ maximally to get $\kappa = (d - \sigma - 1) \bmod d$. We define an *anti-superfluous* literal as:

▶ **Definition 8** (Anti-Superfluous Literal). *When dividing a constraint C with slack σ by d, a non-falsified literal ℓ_{aw} with coefficient c_{aw} is anti-superfluous when $0 < d - (c_{aw} \bmod d) \leq \kappa$ with $\kappa = (d - \sigma - 1) \bmod d$.*

[2] Up to multiplication by a constant factor, which is needed for MIR.

During the weakening of non-divisible non-falsifieds, we can then anti-weaken ℓ_{aw} by $d - (c_{aw} \bmod d)$. This makes the literal divisible without it being weakened, while still not increasing the slack as part of division by d. Note that we can repeat this step until there are no more anti-superfluous literals left.

▶ **Example 9.** We can again look at the conflict from Example 3, but now apply AW. Instead of weakening z by 3, we can anti-weaken it by 2 and then divide by 5.

$$\dfrac{\dfrac{x + 3y + 3z + 5w \geq 6}{x + 3y + 5z + 5w \geq 6}\text{ Anti-Wkn. } z \text{ by } 2}{x + y + z + w \geq 2}\text{ Div. by } 5 \tag{13}$$

The final constraints from Equations (10) and (13) now both satisfy Requirement 2 and after multiplication will also satisfy Requirement 1, but the latter is stronger than the former.

We now prove that division-based reduction with anti-weakening non-falsifieds dominates division-based reduction without.

▶ **Proposition 10.** *Let d be some divisor, ρ a partial assignment, $C = c_{aw}\ell_{aw} + \sum_i c_i\ell_i \geq \delta$ a constraint with slack σ, and ℓ_{aw} an anti-superfluous literal, so $0 < d - (c_{aw} \bmod d) \leq \kappa$.*

Let C' be the constraint after Steps 1 and 2. Let C^{AW} be the constraint after Steps 1 and 2 where during Step 1 ℓ_{aw} is anti-weakened by $d - (c_{aw} \bmod d)$ instead of weakened by $c_{aw} \bmod d$.

C^{AW} implies C' and the slack of C^{AW} is equal to that of C'.

Proof. Since the total amount that is anti-weakened is at most κ, we know that the slacks of C' and C^{AW} are still equal after Step 2. Then, since ℓ_{aw} is anti-weakened for C^{AW} it holds that $\mathit{coeff}(\ell_{aw}, C') + 1 = \mathit{coeff}(\ell_{aw}, C^{AW})$. And since the slack of the two constraints is the same, the following equality holds for their respective degrees $\delta' + 1 = \delta^{AW}$. This means $\mathit{weaken}(C^{AW}, \ell_{aw}, 1) = C'$ and thus C^{AW} implies C'. ◀

Hence, we propose the AW reduction where we iteratively anti-weaken anti-superfluous literals until there are no more left. From Proposition 10 it follows that anti-weakening does not increase the slack (as it yields constraints that are at least as strong), preserving Requirement 2. And since in *RoundingSat*-style reduction the divisor is the reason literal's coefficient, the reason literal is never anti-superfluous and will thus be unchanged, satisfying Requirement 1.

3.3 Combining WS and AW reduction

There is an interesting dynamic between the WS and AW reduction methods. Both exploit Corollary 2 in a similar way, by temporarily increasing the slack, while ensuring Requirement 2 is not violated after division. In the context of division-based reduction, WS and AW are two sides of the same coin, where WS applies to falsified literals and AW to non-falsified literals.

▶ **Proposition 11.** *Let d be some divisor and ρ a partial assignment. Let $C = \sum c_i\ell_i \geq \delta$ be some constraint with slack σ, before Step 1. Let $\kappa = (d - \sigma - 1) \bmod d$. Let C' be that same constraint, with degree δ' after Step 1. Let $\theta = (\delta' - 1) \bmod d$. Then $\theta = \kappa$.*

Proof. Before Step 1, $\kappa = (d - \sigma - 1) \bmod d$. Weakening non-falsified literals does not alter this value. So after Step 1, $\theta = (\delta' - 1) \bmod d$. By rearranging Equation (1) and replacing δ' and since all non-falsifieds are divisible by d we get $\theta = (\delta - 1) \bmod d = (\sum_{\bar{\ell}_i \notin \rho} c_i - \sigma - 1) \bmod d = (-\sigma - 1) \bmod d = (d - \sigma - 1) \bmod d$. Therefore, $\theta = \kappa$. ◀

So the amount θ we can (anti-)weaken by is shared between the two techniques, i.e. if we weaken superfluous literals by θ', then we only have $\theta - \theta'$ left to anti-weaken anti-superfluous literals.

We can easily combine AW and WS in one divsion-based reduction approach, which we present in Algorithm 2. We first apply AW in lines 6 to 8, then WS in lines 10 to 14.

Algorithm 2 reduceDivision.

```
Input: Reason $C_{rsn}$, conflict $C_{co}$, reason literal $\ell_r$, partial assignment $\rho$
Output: Tuple of reduced constraint $C_{rsn}$ and conflict constraint $C_{co}$
1  $d \leftarrow coeff(\ell_r, C_{rsn})$
2  $\theta \leftarrow (d - slack(C_{rsn}, \rho) - 1) \bmod d$
3  for $\ell_i \in lits(C_{rsn})$ do
4      $\alpha \leftarrow coeff(C_{rsn}, \ell_i) \bmod d$
5      if $\alpha \neq 0 \wedge \bar{\ell}_i \notin \rho$ then
6          if $\theta - d + \alpha \geq 1$ then
7              $\theta \leftarrow \theta - d + \alpha$
8              continue
9          $C_{rsn} \leftarrow weaken(C_{rsn}, \ell_i, \alpha)$
10 for $\ell_i \in lits(C_{rsn})$ do
11     $\alpha \leftarrow coeff(C_{rsn}, \ell_i) \bmod d$
12     if $0 < \alpha \leq \theta$ then
13         $\theta \leftarrow \theta - \alpha$
14         $C_{rsn} \leftarrow weaken(C_{rsn}, \ell_i, \alpha)$
15 $C_{rsn} \leftarrow divide(C_{rsn}, d)$
16 $C_{red} \leftarrow coeff(\bar{\ell}_r, C_{co}) \cdot C_{rsn}$
17 return $(C_{red}, C_{co})$
```

4 Saturation-Based Reduction Variants

In the previous section we focused on division-based reduction. In this section we will focus on the other family of reduction techniques: saturation-based reduction. We investigate a saturation-based method called *Multiply and Weaken* (MW) [18]. The main idea we use from MW is to multiply the reason and/or the conflict constraint in order to bring the coefficients of the reason literal in the reason c_{rsn} and conflict c_{co} close to each other, with $c_{rsn} \geq c_{co}$. We develop two reduction variants, both of which use weakening in a different manner to satisfy Requirement 1. As for Requirement 2, the MW reduction variants use the necessary, but less strict, requirement:

Weak Negative Slack Condition (Requirement 3) $slack(C_{red}, \rho) + slack(C_{co}, \rho) < 0$

4.1 Multiply and Weaken Direct (MWD)

The first MW reduction variant, *Multiply and Weaken Direct (MWD)*, applies the following operations. First, multiply C_{rsn} by $\left\lceil \frac{c_{co}}{c_{rsn}} \right\rceil$ and C_{co} by $\mu = \max(1, \left\lfloor \frac{c_{rsn}}{c_{co}} \right\rfloor)$. Then, weaken the reason literal ℓ_r in the multiplied reason constraint by the exact amount needed to satisfy Requirement 1, i.e. by $\left\lceil\frac{c_{co}}{c_{rsn}}\right\rceil \cdot c_{rsn} - \mu \cdot c_{co}$. However, these operations will only yield a reduced reason and conflict constraint (i.e. meeting Requirement 3) if the following condition holds:

$$\left\lceil\frac{c_{co}}{c_{rsn}}\right\rceil \cdot \mathit{slack}(C_{rsn}, \rho) + \mu \cdot \mathit{slack}(C_{co}, \rho) < 0 \tag{14}$$

If Equation (14) does hold, then Requirement 3 is guaranteed to be satisfied, since the multiplications of the slack are taken into account and weakening of non-falsified literals does not increase the slack. If Equation (14) does not hold, MWD reduction uses a fallback reduction method instead. In some cases, MWD yields stronger constraints than division-based reduction:

▶ **Example 12.** Given a reason constraint $C_{rsn} = x + 2y + 3z + 5w \geq 5$, a conflict constraint $C_{co} = 3u + 4\overline{w} + 5y \geq 7$, an assignment $\rho = \{\overline{x}/\cdot, \overline{y}/\cdot, w/C_{rsn}\}$ and a reason literal $\ell_r = w$.

We show Steps 1 and 2 for division-based reduction:

$$\cfrac{\cfrac{x + 2y + 3z + 5w \geq 5}{x + 2y + 5w \geq 2}\ \text{Wkn. } z \text{ by } 3}{x + y + w \geq 1}\ \text{Div. by } 5 \tag{15}$$

For MWD reduction, we first need to check if it is even possible to satisfy Requirement 3. Since $\left\lceil\frac{c_{co}}{c_{rsn}}\right\rceil \cdot \mathit{slack}(C_{rsn}, \rho) + \mu \cdot \mathit{slack}(C_{co}, \rho) = \left\lceil\frac{4}{5}\right\rceil \cdot 3 + \max(1, \left\lfloor\frac{5}{4}\right\rfloor) \cdot (-4) = 3 - 4 = -1 < 0$ Requirement 3 will indeed be satisfied. If so, Requirement 3 will indeed be satisfied and we can continue with the MWD approach. Since no multiplication is needed all we need to do is weaken w by 1:

$$\cfrac{x + 2y + 3z + 5w \geq 5}{x + 2y + 3z + 4w \geq 4}\ \text{Wkn. } w \text{ by } 1 \tag{16}$$

We can see that the reduced reason from MWD in Equation (16) is stronger than the one from division-based reduction Equation (15).

4.2 Multiply and Weaken Indirect (MWI)

While MWD always directly weakens the reason literal ℓ_r, another approach is possible when ℓ_r is *saturated*, i.e. if its coefficient is at least as high as the degree of the constraint. Instead of weakening ℓ_r directly, we can lower the degree by weakening a *different* non-falsified literal, and then apply saturation. This lowers ℓ_r's coefficient to the degree of the constraint, effectively giving ℓ_r the same coefficient as if it was weakened. In this case, we weaken two literals "for the price of one". Other than the different weakening approach followed by saturation of the reason constraint, MWI applies the same operations as MWD. We continue Example 12:

▶ **Example 13.** Instead of weakening w directly by 1, we can instead weaken the non-falsified literal z by 1 and then saturate C_{rsn} so that w gets the desired coefficient 4.

$$\cfrac{\cfrac{x + 2y + 3z + 5w \geq 5}{x + 2y + 2z + 5w \geq 4}\ \text{Wkn. } z \text{ by } 1}{x + 2y + 2z + 4w \geq 4}\ \text{Sat.} \tag{17}$$

Clearly, since the coefficient of z is lower, the reduced reason in Equation (17) is stronger than the one obtained by MWD in Equation (16).

Note that constraints obtained by MWI imply those obtained by MWD, as the only difference in both routines is that some coefficients are lowered for MWI, while the degree in the reduced reason remains exactly the same.

4.3 Combining MWD, MWI, and division-based reduction

We present the novel MW variants in Algorithm 3. On line 3 we check Equation (14) to see if Requirement 3 will hold after MWD. If it does, we multiply the constraint and calculate the amount we directly weaken ℓ_r by on line 17. Otherwise, we use a fallback reduction, in our case this is division-based reduction from Algorithm 2. Note that this allows us to combine the division-based reduction variants, AW and WS, with the saturation-based variants, MWD and MWI. Then from line 9 to 16, if the reason literal is saturated, we apply indirect weakening and saturation. We perform the final step of direct weakening and saturate again. In this fashion, we use MWI in combination with MWD since there is no guarantee that only MWI will always sufficiently reduce the reason coefficient.

Algorithm 3 reduceSaturation.

```
   Input: Reason $C_{rsn}$, conflict $C_{co}$, reason literal $\ell_r$, partial assignment $\rho$
   Output: Tuple of reduced $C_{rsn}$ and (potentially multiplied) $C_{co}$
 1 $c_{rsn} \leftarrow coeff(C_{rsn}, \ell_r)$
 2 $c_{co} \leftarrow coeff(C_{co}, \bar{\ell_r})$
 3 if $\left\lceil \frac{c_{co}}{c_{rsn}} \right\rceil \cdot slack(C_{rsn}, \rho) + \mu \cdot slack(C_{co}, \rho) < 0$ then
 4     $C_{rsn} \leftarrow \left\lceil \frac{c_{co}}{c_{rsn}} \right\rceil \cdot C_{rsn}$
 5     $C_{co} \leftarrow \mu \cdot C_{co}$
 6     $\alpha \leftarrow \left\lceil \frac{c_{co}}{c_{rsn}} \right\rceil \cdot c_{rsn} - \mu \cdot c_{co}$
 7 else
 8     return $reduceDivision(C_{rsn}, C_{co}, \ell_r, \rho)$
 9 if $c_{rsn} \geq \delta_{C_{rsn}}$ then
10     for $\ell_i \in C_{rsn}$ with coefficient $c_i \wedge \ell_i \notin \rho$ do
11         if $c_i > \alpha$ then
12             $C_{rsn} \leftarrow weaken(C_{rsn}, \ell_i, \alpha)$
13             $\alpha \leftarrow 0$
14         else
15             $C_{rsn} \leftarrow weaken(C_{rsn}, \ell_i, c_i)$
16             $\alpha \leftarrow \alpha - c_i$
17     $C_{rsn} \leftarrow saturate(C_{rsn})$
18 $C_{rsn} \leftarrow weaken(C_{rsn}, \ell_r, \alpha)$
19 return $(saturate(C_{rsn}), C_{co})$
```

5 Experimental Evaluation

In this section, we evaluate the impact of the proposed techniques on solver performance. We implemented our techniques into *Exact* 2.1.0 [11][3], which is a fork of *RoundingSat* [12]. We use three benchmark sets:

KNAP crafted knapsack (783 instances) [23, 9]
DEC-LIN the decision linear track of the PB'24 competition (398 instances)
OPT-LIN the optimisation linear track of the PB'24 competition (489 instances)

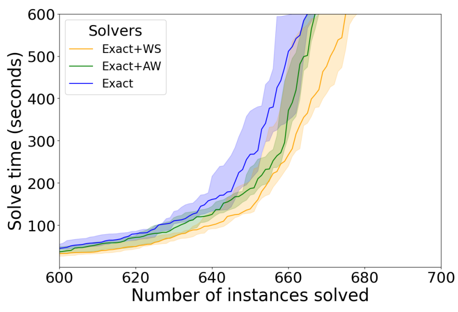
[3] The earlier version of *Exact* submitted to the PB'24 competition already incorporated these techniques

For KNAP the solver is given a memory limit 8GB of RAM and a timeout of 600 seconds. For DEC-LIN and OPT-LIN the solver is given a memory limit of 31GB of RAM and a timeout of 3600 seconds as in the PB'24 competition. The KNAP benchmark set is added because each instance consists of a single constraint, which puts a heavy emphasis on learning strong constraints. Our experiments were run on a cluster with 32 INTEL(R) XEON(R) SILVER 4514Y (2GHz) CPUs with 256GB of shared memory. We run each configuration of the solver with 5 seeds. Our figures are plotted with performance on the x-axis and solve time on the y-axis. Optimisation instances are considered solved if an optimal solution is found *and* it is proven that no better exists. We use the results from the five seeds to plot 95% confidence bands to visualize potential variance in solve-times, with a solid line for the median. The implementation and run logs are included as supplemental material.

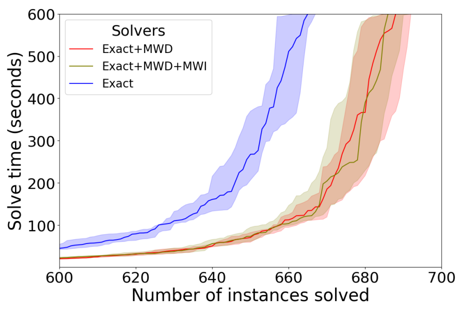
5.1 Individual techniques

Our first experiment compares each individual technique (i.e. the WS, AW, MWD, and MWD+MWI reduction methods) to *Exact* as a baseline. For the division-based techniques we see that on KNAP in Figure 1a, WS solves more instances and AW solves some instances faster. We also see positive results when it comes to the saturation-based techniques in Figure 1b. MWD and MWD+MWI both perform similarly, while outperforming the baseline. On DEC-LIN we see similar trends for the division-based techniques in Figure 2a. WS still solves more instances than the baseline and AW mostly helps solving some instances faster. For the saturation-based techniques we do see in Figure 2b a different trend compared to KNAP. While MWD still helps improve the performance of the solver, MWD+MWI shows a negative performance. This unexpected result is interesting, since we know MWD+MWI dominates MWD from a theoretical perspective. On OPT-LIN we do not see much of a difference either way for the division-based techniques in Figure 3a, but the performance marginally worsens for the saturation-based techniques in Figure 3b.

We believe the smaller impact of AW is due to how each technique changes the constraint. WS only lowers one coefficient while keeping everything else the same, while AW increases the degree as well as a coefficient. The increase in the degree could however lead to fewer saturation opportunities after addition with the conflict constraint, reducing the impact. On the other hand, lowering a coefficient via WS may lead to fewer variable cancellations and in turn less saturation after addition, though the odds for cancellation on non-propagated literals may be relatively smaller. In the end, both techniques show only moderate impact on the selected benchmarks.

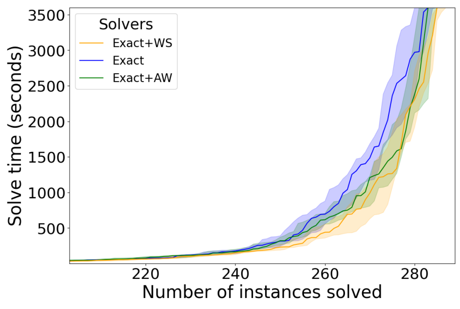


(a) KNAP.

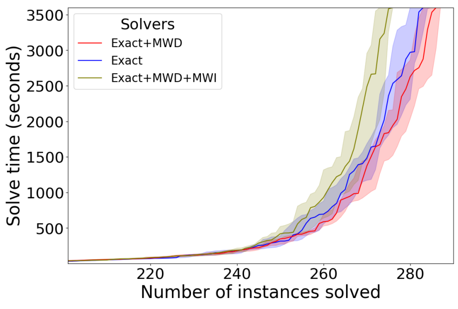


(b) KNAP.

Figure 1 Comparing individual division- and saturation-based techniques on KNAP.

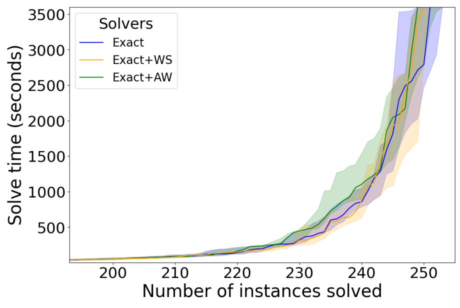


(a) DEC-LIN.

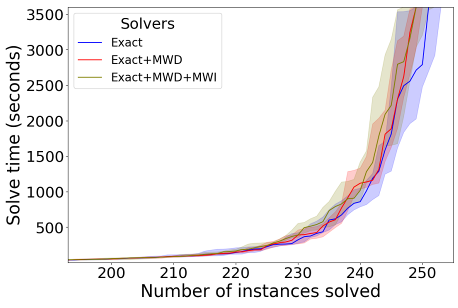


(b) DEC-LIN.

Figure 2 Comparing individual division- and saturation-based techniques on DEC-LIN.



(a) OPT-LIN.



(b) OPT-LIN.

Figure 3 Comparing individual division- and saturation-based techniques on OPT-LIN.

5.2 Combined techniques

In our second experiment we evaluate how combining the different techniques as shown in Section 4.3 impacts the empirical runtime of PB solvers. Additionally, we test an implementation of the combined division-based techniques, WS+AW, on another solver, namely the PB'24 competition version of *RoundingSat*, to see how it behaves in different solvers. Implementing the saturation-based techniques in *RoundingSat* would have required more extensive changes due to overflows after multiplication.

The results are summarised in Table 1. On KNAP, combining all the techniques has a compounding effect for both solvers. On DEC-LIN, the results are less clear. *Exact+WS+AW* does perform very well, which might initially seem surprising since according to Proposition 11 applying AW means less WS is possible. *RoundingSat+WS+AW*, on the other hand has minor impact on *RoundingSat* for DEC-LIN. Still, it seems both techniques can be combined effectively. The same cannot be said when mixing the division- and saturation-based techniques. We hypothesize that since WS+AW improves division-based reduction, a better heuristic than Equation (14) for the saturation-based techniques is necessary. It may be that WS+AW is most effective in many cases where MWD is actually possible, thus the improvements from WS+AW carry over to WS+AW+MWD. On OPT-LIN, the choice of configuration of the techniques still does not seem to have much impact, with similar performance to the configurations using an individual technique.

Table 1 Median number of solved instances for different solver configurations across benchmarks.

Solved	KNAP (783)	DEC-LIN (398)	OPT-LIN (489)
Exact	664	282	250
Exact+WS	674	284	249
Exact+AW	666	282	248
Exact+WS+AW	683	**287**	247
Exact+MWD	687	285	248
Exact+MWD+MWI	685	275	248
Exact+WS+AW+MWD	695	284	248
Exact+WS+AW+MWD+MWI	698	280	249
RoundingSat	691	281	**253**
RoundingSat+WS+AW	**709**	282	251

6 Conclusion and Future Work

We presented novel techniques to generate stronger reduced constraints in both division-based [12] and saturation-based [18] reduction methods. As in established work [21], we can indeed prove dominance relationships between the various reduction methods, which guarantee that reduced constraints obtained from one are at least as strong as those from another. The experiments show that stronger reduced constraints can improve the solver performance for different solvers and benchmarks, but not uniformly across all problems. While there are improvements on crafted knapsack benchmarks, and the competition decision benchmark for *Exact*, we observe little difference on competition optimisation benchmarks. Perhaps more surprisingly, in competition decision benchmarks we see a case of worsening performance for MWD+MWI compared to MWD, despite their dominance relationship.

Hence our theoretical results provide a better understanding of reduction methods and the freedom there is in reducing constraints before addition. Empirically there is a lesser understood relationship between the strength of the reduced reason constraint, the strength of the learned constraint after all iterations of constraint addition in the conflict analysis, and the effect of the learned constraints on solver performance. However, these relationships are complex, because they involve the reduction and resolution of multiple reason constraints with the conflict constraint. With the insights of the paper we also see avenues to strengthen the reduced constraints further. For example, in division-based reduction, relaxing Requirement 2 to Requirement 3 (as in saturation-based reduction) could lead to a smaller divisor or an increase in the amount of superfluous and anti-superfluous literals.

We also saw how some combinations of reduction techniques can be effective, but they use heuristics, e.g. to choose between division- and saturation-based reduction for specific constraints. These heuristics are much less studied and can have a big impact on empirical performance which deserves further study.

References

1 Gilles Audemard and Laurent Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, February 2018. `doi:10.1142/S0218213018400018`.

2 Danel Le Berre, Pierre Marquis, Stefan Mengel, and Romain Wallon. On Irrelevant Literals in Pseudo-Boolean Constraint Learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1148–1154, July 2020. `doi:10.24963/ijcai.2020/160`.

3 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *J. Satisf. Boolean Model. Comput.*, 7(2-3):59–6, 2010. doi:10.3233/sat190075.

4 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021. doi:10.3233/FAIA200990.

5 Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *Proceedings of the 40th Annual Design Automation Conference*, pages 830–835, Anaheim CA USA, June 2003. ACM. doi:10.1145/775832.776041.

6 Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24:305–317, 2005. doi:10.1109/TCAD.2004.842808.

7 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047.

8 W. Cook, C.R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, September 1987. doi:10.1016/0166-218X(87)90039-4.

9 Jo Devriendt. Pisinger's knapsack instances in opb format, July 2020. doi:10.5281/zenodo.3939055.

10 Jo Devriendt. Watched Propagation of 0-1 Integer Linear Constraints. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming*, pages 160–176, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-58475-7_10.

11 Jo Devriendt. Exact: Evaluating cutting-planes learning at the PB'24 competition. *Slides*, 2024.

12 Jan Elffers and Jakob Nordström. Divide and Conquer: Towards Faster Pseudo-Boolean Solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1291–1299, Stockholm, Sweden, July 2018. International Joint Conferences on Artificial Intelligence Organization. doi:10.24963/ijcai.2018/180.

13 Jan Elffers and Jakob Nordström. RoundingSat 2019: Recent work on PB solving. https://slides.com/jod/deck-26-29-32, July 2019.

14 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science. doi:10.1016/0304-3975(85)90144-6.

15 John N Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.

16 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Ipbhs in pb'24 competition. *Slides*, 2024.

17 Christoph Jabs, Jeremias Berg, and Matti Järvisalo. PB-OLL-RS and MIXED-BAG in Pseudo-Boolean Competition 2024. *PB Competition 2024*, 2024.

18 Daniel Le Berre, Pierre Marquis, and Romain Wallon. On Weakening Strategies for PB Solvers. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020*, pages 322–331, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-51825-7_23.

19 Orestis Lomis, Jo Devriendt, Hendrik Bierlee, and Tias Guns. ML-KULeuven/SAT25_PB_reductions_experiments. Dataset, This research was partly funded by the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt)., swhId: swh:1:dir:650c2f9b37926504a59a11a6dbf319620b311dc3 (visited on 2025-07-23). URL: https://github.com/ML-KULeuven/SAT25_PB_reductions_experiments, doi:10.4230/artifacts.24089.

20 J.P. Marques Silva and K.A. Sakallah. Grasp-a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227, 1996. doi:10.1109/ICCAD.1996.569607.

21 Gioni Mexi, Timo Berthold, Ambros Gleixner, and Jakob Nordström. Improving conflict analysis in mip solvers by pseudo-boolean reasoning. *arXiv preprint arXiv:2307.14166*, 2023. doi:10.48550/arXiv.2307.14166.

22 Shiwei Pan, Yiyuan Wang, Shaowei Cai, Jiangnan Li, Wenbo Zhu, and Minghao Yin. Cashwmaxsat-disjcad: Solver description. *MaxSAT Evaluation 2024*, 2024:25, 2024.

23 David Pisinger. Where are the hard knapsack problems? *Comput. Oper. Res.*, 32(9):2271–2284, 2005. doi:10.1016/j.cor.2004.03.002.

24 Roussel, Olivier. Pseudo-Boolean Competition of 2024. https://www.cril.univ-artois.fr/PB24/.