Symbolic Conflict Analysis in Pseudo-Boolean Optimization

Robert Nieuwenhuis
Barcelogic.com, Barcelona, Spain

Albert Oliveras
Technical University of Catalonia, Barcelona, Spain

Enric Rodríguez-Carbonell
Technical University of Catalonia, Barcelona, Spain

Rui Zhao
Technical University of Catalonia, Barcelona, Spain

Abstract

In the the last two decades, a lot of effort has been devoted to the development of satisfiability-checking tools for a variety of SAT-related problems. However, most of these tools lack optimization capabilities. That is, instead of finding any solution, one is sometimes interested in a solution that is best according to some criterion.

Pseudo-Boolean solvers can be used to deal with optimization by successively solving a series of problems that contain an additional pseudo-Boolean constraint expressing that a better solution is required. A key point for the success of this simple approach is that lemmas that are learned for one problem can be reused for subsequent ones.

In this paper we go one step further and show how, by using a simple symbolic conflict analysis procedure, not only can lemmas be reused between problems but also strengthened, thus further pruning the search space traversal. In addition, we show how this technique automatically allows one to infer upper bounds in maximization problems, thus giving an estimation of how far the solver is from finding an optimal solution. Experimental results with our PB solver reveal that (i) this technique is indeed effective in practice, providing important speedups in problems where several solutions are found and (ii) on problems with very few solutions, where the impact of our technique is limited, its overhead is negligible.

2012 ACM Subject Classification Theory of computation

Keywords and phrases SAT, Pseudo-Boolean Optimization, Conflict Analysis

Digital Object Identifier 10.4230/LIPIcs.SAT.2025.23

Supplementary Material
Software (Source Code): `https://github.com/dearzhaorui/symbolic-conflict-analysis`
archived at `swh:1:dir:f5894275ca84f010f2626a78762667102cefb9f4`

Funding All authors are supported by grant PID2021-122830OB-C43, funded by MCIN/AEI/10.13039/501100011033 and by "ERDF: A way of making Europe". Second and third authors are supported by Barcelogic through research grants C-11423 and C-11422, respectively.

1 Introduction

SAT solvers are nowadays routinely used in an increasing number of applications areas. Despite success stories initially emerged in the area of verification [4], they are now common in security [10, 35], cryptography [34] and even mathematics [18]. In parallel to these practical developments, theoretical works have shown that CDCL [23], the most successful procedure for SAT, cannot solve certain type of problems (e.g. the pigeon-hole principle [17]) in polynomial time [2, 28].

Pseudo-Boolean (PB) solving, also known as 0-1 Integer Linear Programming, has established itself as a promising alternative to SAT. First of all, 0-1 linear constraints are more general than clauses, and allow one to encode problems in a more compact way. Secondly, cutting planes [7], the underlying proof system of CDCL-based PB solvers [31] is exponentially more powerful than resolution [30], the proof system which CDCL-based SAT solvers rely on. Thus, PB solvers are, at least from the theoretical point of view, exponentially more powerful than SAT solvers.

Last, but not least, sometimes satisfiability checking is not enough. In several applications [16, 1, 26], one wants to find an optimal solution according to some criterion. Unlike what happens with SAT solvers, it is very easy to turn a PB solver into a PB optimizer, which finds a solution to a set of 0-1 linear constraints that maximizes a given objective 0-1 linear function. The solver is initially run on the set of constraints, ignoring the objective function, in order to obtain a first solution. If the objective function value for this solution is C, a constraint expressing that only solutions with objective function value strictly larger than C is added and the solver is executed again. The process is repeated until the solver is not able to find further solutions, concluding that the last solution found is optimal.

A noteworthy aspect of the previous approach, known as *Linear Search*, is that any lemma learned by the solver when looking for a solution larger than C can still be used at any future point when a solution larger than $C' > C$ is sought. This well-known fact [25] is crucial for the performance of this method.

Contributions. In this paper, we go one step further and show how constraints learned when looking for a solution larger than C can not only be reused, but also automatically *strengthened* so that a stronger version of them is available during the search for solutions larger than $C' > C$. This is accomplished by considering C as a symbolic variable and adapting conflict analysis to take care of this feature. As a by-product of this symbolic way of treating constraints, one can automatically extract upper bounds from them, thus giving an estimation of how far the solver is from reaching an optimal solution. Experimental results show that the overhead of this symbolic procedure is negligible, and that its ability to further prune the search space results in important runtime improvements.

Related Work. Pseudo-Boolean optimization is a well-studied problem for which three main methods exist: Linear Search, Core-guided Search [13, 24, 9], and Implicit Hitting Set approaches [33, 32]. They are considered to be complementary in the type of instances for which they work best and solvers might even use combinations of them (e.g. RoundingSat combines Linear with Core-guided Search). Linear Search, which is the topic of this paper, has also been applied to the MaxSAT problem. Two prominent contributions in this direction are Pacose [27] and MaxCDCL solver [21, 22], which combines a branch and bound search with a lower-bounding procedure to prune the search space. As far as we know, no previous work uses a symbolic treatment of the objective function and no other linear-search method can compute upper bounds with no additional cost. Hence, we believe the community will benefit from the introduction of this novel technique and will develop additional ways to further exploit it.

This paper is organized as follows. After some preliminaries in Section 2, we revisit the conflict analysis procedure of PB solvers in Section 3. Section 4 presents the main contributions of this paper: the symbolic conflict analysis procedure and its additional ability to compute upper bounds. Experimental evidence of their positive impact on a solver and a careful analysis of the data collected is reported in Section 5. We conclude in Section 6, where we also outline future research directions.

2 Preliminaries

Pseudo-Boolean Constraints. Let $\mathcal{X}$ be a set of propositional variables. A *literal* is either a variable (x) or the negation of one ($\overline{x}$). We will assume that $\overline{\overline{x}} = x$. A *(pseudo-Boolean* or *PB) constraint* is a 0-1 linear inequality $\sum_i c_i l_i \geq d$ where the l_i's are literals and, without loss of generality, the c_i's (*coefficients*) and d (*degree*) are positive integers. When all coefficients are 1 we say that the constraint is a *cardinality constraint* and if, in addition, d is also 1 we say that it is a *clause*. A *formula* is a set of constraints.

Satisfiability and Logical Consequence. An *assignment* ρ is a set of non-contradictory literals. It is *total* if for any $x \in \mathcal{X}$ either $x \in \rho$ or $\overline{x} \in \rho$, and *partial* otherwise. A literal l is *true* in ρ if $l \in \rho$, is *false* if $\overline{l} \in \rho$ and is *undefined* otherwise. Given a constraint C of the form $\sum_i c_i l_i \geq d$, an assignment ρ *satisfies* it if $\sum_{i:l_i \in \rho} c_i \geq d$, and *falsifies* it if no extension of ρ can satisfy it. If we define $slack(C, \rho) = (\sum_{i:\overline{l_i} \notin \rho} c_i) - d$, it can be seen that ρ falsifies C if and only if $slack(C, \rho) < 0$. Note the slack expression sums the coefficients of all non-false literals, and that is the maximum value that the left hand side of the constraint can reach. If even that does not exceed d, no extension of ρ will do. An assignment that satisfies all constraints of a formula is called a *model*. A formula F is a *logical consequence* of G if any model of G is also a model of F.

Inference Rules. In order to determine the satisfiability of a set of constraints, one can use the *cutting planes* proof system, which consists of axioms $l \geq 0$ for all literals l, and the following two inference rules:

$$\text{Linear combination:} \quad \frac{\sum_i a_i l_i \geq A \quad \sum_i b_i l_i \geq B}{\sum_i (\alpha a_i + \beta b_i) l_i \geq \alpha A + \beta B} \qquad \alpha, \beta \in \mathbb{N}^+$$

$$\text{Division:} \quad \frac{\sum_i a_i l_i \geq A}{\sum_i \lceil a_i/\alpha \rceil l_i \geq \lceil A/\alpha \rceil} \quad \alpha \in \mathbb{N}^+$$

It is well known [19] that a set of constraints is unsatisfiable if and only if $0 \geq 1$ can be derived using these rules. We note that in the application of linear combination we implicitly assume that any constraint is simplified by using that fact that $l + \overline{l} = 1$. For example, the constraint $2x + 3\overline{x} + 5y \geq 7$ can be rewritten as $2(x + \overline{x}) + \overline{x} + 5y \geq 7$ and is hence equivalent to $\overline{x} + 5y \geq 5$. Another well-known rule, which is very useful in PB solving, limits coefficients to be at most equal to the degree:

$$\text{Saturation:} \quad \frac{\sum_i a_i l_i \geq A}{\sum_i min(a_i, A)\, l_i \geq A}$$

Unit Propagation. Given a constraint $C = \sum_i c_i l_i \geq d$ and an assignment ρ, we say that C *unit propagates* l_i under ρ if l_i is undefined in ρ, but l_i is true in any total assignment extending ρ that satisfies C. The latter is equivalent to checking whether $slack(C, \rho) < c_i$, i.e., if we do not set l_i to true, the constraint becomes falsified. Given a formula F and an assignment ρ, *unit propagation* of F under ρ is the outcome of applying the following two rules until a fixpoint is reached: (i) if ρ falsifies a constraint $C \in F$, a *conflict* is found with *conflicting constraint* C and we stop, (ii) if ρ unit propagates some literal l due to constraint C, extend $\rho := \rho \cup \{l\}$ with *reason* C.

Conflict-Driven Pseudo-Boolean Solving. A generalization of the well-known CDCL [23] algorithm for SAT can be applied to the pseudo-Boolean case [31]. The algorithm starts with an empty assignment ρ and proceeds as follows: (1) Apply unit propagation, possibly extending ρ. (2) If a conflict is found, a conflict analysis procedure uses the aforementioned inference rules to derive a constraint C (called *lemma*) that can be safely added to the formula. If C is the constraint $0 \geq 1$, the formula is unsatisfiable, otherwise it is guaranteed that after removing some literals from ρ in a last-in first-out way (*backjumping*), C allows some literal to be unit propagated. Hence, we go to step 1. (3) If no conflict is found, and ρ is total, it is a model of the formula. Otherwise, an undefined literal l (*decision literal*) is added to ρ and we go to step 1. If we see ρ as a sequence, any literal appearing in the sequence between the k-th decision literal (included) and before the $(k+1)$-th one is said to belong to *decision level* k.

Pseudo-Boolean Optimization. Given a set S of PB-constraint and an *objective* function $\sum_i c_i l_i$, the *Pseudo-Boolean Optimization* problem consists of finding, among all assignments that satisfy S, one that maximizes the value of the objective function. A very simple approach for this problem is to first run a PB solver to determine the satisfiability of S. If a solution is found, for which the objective function has value C, a constraint $\sum_i c_i l_i \geq C + 1$ is added to S and the same solver is launched again. Eventually, the solver will report the unsatisfiability of the augmented set of constraints, and the last found solution is an optimal one. This is sometimes known as the *Linear Search* approach to PB optimization.

3 Conflict Analysis in CDCL-Based Pseudo-Boolean Solvers

In CDCL-based PB solvers, when unit propagation finds a conflicting constraint, a procedure is launched whose ultimate goal is to derive a new constraint that (i) is a logical consequence of the current formula and hence can safely be added to it, and (ii) allows the solver to backjump to some previous decision level and propagate a literal at that point. This mimics conflict analysis in SAT solvers, where the 1-UIP scheme [36] has established itself as the dominant approach and, after twenty years, only small variants have been added to it (e.g. [12]).

Conflict analysis in PB solvers is a more complex task than in SAT and one can find different variants in state-of-the-art CDCL-based PB solvers. However, we believe that Algorithm 1 is an adequate abstraction of most of them. For the purpose of this paper, this simplified presentation is detailed enough so that we can introduce our procedure in Section 4. The overall idea of Algorithm 1 is that a series of linear combination steps are applied between the conflicting constraint and the reasons for certain literals in the trail ρ until we obtain a constraint that propagates some literal at some previous decision level. This is essentially what conflict analysis does in CDCL-based SAT solvers, where resolution can be seen as a concrete case of linear combination.

There are three differences we would like to remark. The first difference can be found in the loop in lines 4-6. In a SAT solver, this loop looks for the topmost literal $l \in \rho$ whose negation appears in $confCtr$. But the actual property we want about l is that the addition of l to ρ is the one that caused $confCtr$ to be false. In a PB constraint, the literal l with this property need not be the topmost in ρ whose negation appears in $confCtr$. For example, if $\rho = (\overline{x}, \overline{y}, \overline{z})$, the conflicting constraint $2x + y + z \geq 3$ already became false when $\overline{x}$ was added to ρ, although the topmost literal whose negation appears in the constraint is $\overline{z}$.

Algorithm 1 CDCL-Based PB solver Conflict Analysis.

```
1  Function Conflict-Analysis(Assignment ρ, Constraint confCtr):
2      while true do
3          int slck := slack(confCtr, ρ)  // slck < 0 because confCtr is a conflict.
4          while slck < 0 do
5              Literal l := ρ.pop() // ρ is a stack
6              if l̄ ∈ confCtr then slck := slck + coef(l̄, confCtr)
7          Constraint reasonCtr := reason(l, ρ)  // l ∈ reasonCtr and l̄ ∈ confCtr
8          weakenConstraints(l, confCtr, reasonCtr)
9          α := coef(l̄, confCtr)
10         β := coef(l, reasonCtr)
11         confCtr := linearComb(β, confCtr, α, reasonCtr)  // confCtr is false in ρ
12         if confCtr propagates at previous decision level then return confCtr
```

The second, and probably the most important difference, is that we want the application of a linear combination to preserve the fact that the resulting constraint is false in ρ. This property, which trivially holds in SAT solvers, is not true when using linear combinations in general.

▶ **Example 1.** Let us consider the two constraints $5v+2\overline{x}+3y+3z \geq 6$ and $5\overline{v}+5x+3y+4z \geq 6$. Assume that we decide on $\overline{x}$ and later on $\overline{y}$. The second constraint propagates z and $\overline{v}$ and hence we have the assignment $\rho = (\overline{x}, \overline{y}, z, \overline{v})$. But now the first constraint is conflicting. After the loop in lines 4-6, l is $\overline{v}$ and, if line 8 were not present, $\alpha = \beta = 5$ and hence the linear combination[1] would be $15x + 30y + 35z \geq 25$ (note that the right hand side is $30 + 30 - 25 - 10$, where the negative numbers come from canceling 25 units of variable v and 10 units of variable x). One can now check that the derived constraint is no longer false in the new assignment $(\overline{x}, \overline{y}, z)$.

To avoid this problem, which is well-known in the PB [6, 11, 3] and the ILP communities [20, 29], function *weakenConstraints* replaces the constraints by weaker versions, that is, logical consequences. Probably the easiest solution is to replace the reason of a literal by the clause that expresses the implication. In the previous example we could convert the reason for v, which was $5\overline{v} + 5x + 3y + 4z \geq 6$, to $\overline{v} + x + y \geq 1$, expressing that if both x and y are false, then $\overline{v}$ needs to be true. Now $\alpha = 5$ and $\beta = 1$ and the linear combination that consists of adding 5 times the new reason clause with the conflicting constraint is $3x + 8y + 3z \geq 4$, which is now false in the assignment $(\overline{x}, \overline{y}, z)$. Other possibilities for weakening the reason constraint exist [6, 11, 3] but they are out of the scope of this paper.

The last difference in the procedure refers to the termination of the procedure. In SAT solving, conflict analysis terminates when the 1-UIP is found, which guarantees that the current $confCtr$ propagates at some previous decision level. In PB solving, that constraint might be propagating even before the 1-UIP is found and hence it makes sense to check, after every linear combination step, whether $confCtr$ propagates at some previous level. In order to alleviate the computational effort of this check, we use the fact that if $confCtr$ and $reasonCtr$ do not have any contradictory literal other than l, then if $confCtr$ does not propagate at some previous decision level, neither does the constraint resulting of the linear combination between $confCtr$ and $reasonCtr$ computed in line 11 of Algorithm 1. We prove this in the following.

[1] Note that in Algorithm 1 one can use the least common multiple to compute smaller α, β but we omit this detail in the presentation.

▶ **Lemma 2.** *Let $\mathcal{C} := al + C \geq k_c$ and $\mathcal{D} := b\bar{l} + D \geq k_d$ be two PB constraints such that their parts C and D have no contradictory literal and let $\mathcal{R}$ be the linear combination $b\mathcal{C} + a\mathcal{D}$. For any assignment ρ that assigns some value to l, if $\mathcal{R}$ is false in ρ then either $\mathcal{C}$ or $\mathcal{D}$ are also false in ρ.*

Proof. Given a constraint $X \geq d$, we denote by $SNF(X, \rho)$ the (S)um of the coefficients of all literals in X that are (N)on-(F)alse in ρ. It holds that $SNF(X, \rho) = slack(X \geq d, \rho) + d$.

Let us assume that neither $\mathcal{C}$ nor $\mathcal{D}$ are false in ρ and reach a contradiction. Since $\mathcal{R}$ is false in ρ, we have that $slack(\mathcal{R}, \rho) < 0$. Since we know that $\mathcal{R}$ is of the form $bC + aD \geq bk_c + ak_d - ab$, with no contradictory literals between C and D, we have that $slack(\mathcal{R}, \rho) = SNF(bC+aD, \rho) - bk_c - ak_d + ab = bSNF(C, \rho) + aSNF(D, \rho) - bk_c - ak_d + ab$. Note that the last equality holds because C and D have no contradictory literals, otherwise cancellation between those literals could make it invalid. All in all, we can conclude that $bSNF(C, \rho) + aSNF(D, \rho) < bk_c + ak_d - ab$.

Now, since $\mathcal{C}$ is not false, we have that $slack(\mathcal{C}, \rho) \geq 0$ and hence $SNF(al + C, \rho) - k_c \geq 0$ and also $aSNF(l, \rho) + SNF(C, \rho) - k_c \geq 0$ and $abSNF(l, \rho) + bSNF(C, \rho) - bk_c \geq 0$. Similarly $abSNF(\bar{l}, \rho) + aSNF(D, \rho) - ak_d \geq 0$. If we add these last two inequalities, and considering that $SNF(l, \rho) + SNF(\bar{l}, \rho) = 1$ because l is assigned in ρ, we have that $ab + bSNF(C, \rho) + aSNF(D, \rho) - bk_c - ak_d \geq 0$, which contradicts the last inequality of the previous paragraph. ◀

▶ **Corollary 3.** *Let $\mathcal{C} := al + C \geq k_c$ and $\mathcal{D} := b\bar{l} + D \geq k_d$ be two PB constraints such that their parts C and D have no contradictory literal and let $\mathcal{R}$ be the linear combination $b\mathcal{C} + a\mathcal{D}$. For any assignment ρ that assigns some value to l, if unit propagation of $\mathcal{R}$ under ρ propagates a literal l_p, then either $\mathcal{C}$ or $\mathcal{D}$ also propagate l_p under ρ.*

Proof. Apply Lemma 2 with assignment $\rho \cup \{\bar{l}_p\}$ and use the fact that a constraint $\mathcal{C}$ is false in $\rho \cup \{\bar{l}_p\}$ if and only if $\mathcal{C}$ propagates a literal l_p under ρ. ◀

We conclude this section with an example showing the execution of a CDCL-based PB solver that uses a conflict analysis method based on Algorithm 1. This allow us to show how our procedure of Section 4 is able to improve it. For simplicity, we have chosen an example for which the application of *weakenConstraints* is never necessary.

▶ **Example 4.**

$$
\begin{array}{lllllllllll}
Max & & & x & +2y & +3z & +4u & +5v & +6w & & \\
 & \overline{a} & & +\overline{x} & +\overline{y} & & & & & \geq 2 & (C_1) \\
 & & \overline{b} & & & +\overline{z} & +\overline{u} & & & \geq 2 & (C_2) \\
 & & & \overline{c} & & & & +\overline{v} & +\overline{w} & \geq 2 & (C_3)
\end{array}
$$

First solution. Since no constraint is propagating, the solver decides on a literal, say a, which propagates $\overline{x}$ and $\overline{y}$ due to C_1. Next decision is b, which propagates $\overline{z}$ and $\overline{u}$ due to C_2. Next decision is c, which propagates $\overline{v}$ and $\overline{w}$ due to C_3. All constraints are satisfied and we have found a solution for which the objective function has value 0.

Second solution. Since we have to look for a better solution, constraint $C_4 := x + 2y + 3z + 4u + 5v + 6w \geq 1$ is added. The solver makes the same decisions and propagations as before, but the assignment $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2, c^d, \overline{v}_3, \overline{w}_3)$, where a superscript d indicates a decision and a subscript i represents that constraint C_i is the reason for that propagation, now falsifies constraint C_4. Conflict analysis is started, popping literal $\overline{w}$ from ρ, making *slck*

be 5 and the loop in lines $4-6$ terminate. Since C_3 is the reason for $\overline{w}$, and $\alpha = 6, \beta = 1$, the linear combination is $1 \cdot C_4 + 6 \cdot C_3$, which is $C_5 := 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 2$. This constraint propagates $\overline{c}$ at decision level 2, and hence the solver backjumps to produce assignment $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2, \overline{c}_5)$. No other constraint propagates and the solver now decides on v, which propagates $\overline{w}$ due to C_3 and ρ is a solution with value 5.

Third solution. Since a better solution is needed we replace C_4 by $x + 2y + 3z + 4u + 5v + 6w \geq 6$. The solver restarts and makes the same decisions and propagations to produce $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2)$. However, now C_4 propagates w and, after that, C_3 propagates $\overline{v}$ and $\overline{c}$, which produces a solution with value 6.

Fourth solution. C_4 now becomes $x + 2y + 3z + 4u + 5v + 6w \geq 7$. The solver restarts and proceeds as before to reach $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2)$. Now C_4 propagates v and w and the resulting assignment $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2, v_4, w_4)$ falsifies constraint C_3 The loop at lines $4-6$ pops w from ρ, making *slck* to be 0. Since the coefficient of w in the reason of w is $\beta = 6$ and the coefficient of $\overline{w}$ in C_3 is $\alpha = 1$ the linear combination computed is $6 \cdot C_3 + 1 \cdot C_4$, which is $C_{tmp} := 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 8$. Since this constraint does not propagate at any previous decision level, one more iteration of conflict analysis is required. The loop at lines 4-6 pops literals v and $\overline{u}$, after which the *slck* is 3 and the loop terminates. The procedure will now try to eliminate $\overline{u}$ from C_{tmp} by performing an appropriate linear combination with the reason for $\overline{u}$, which is C_2. This linear combination is $1 \cdot C_{tmp} + 4 \cdot C_2$, which results in $C_6 := 4\overline{b} + 6\overline{c} + x + 2y + \overline{z} + \overline{v} \geq 9$. This constraint is learned since it allows the solver to backjump to decision level 1 and propagate $\overline{c}$ and $\overline{b}$. The assignment is now $\rho = (a^d, \overline{x}_1, \overline{y}_1, \overline{c}_6, \overline{b}_6)$. No other constraint propagates and the solver decides on v, which propagates $\overline{w}$ due to C_3. Next decision is on u, which propagates $\overline{z}$ due to C_2. The current assignment ρ is a solution with value 9.

Next step. C_4 now becomes $x + 2y + 3z + 4u + 5v + 6w \geq 10$. Let us now remark that the solver, apart from the original constraints, only has the lemmas:

$$
\begin{array}{ll}
x + 2y + 3z + 4u + 5v + 6w \geq 10 & (C_4) \\
6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 2 & (C_5) \\
4\overline{b} + 6\overline{c} + x + 2y + \overline{z} + \overline{v} \geq 9 & (C_6)
\end{array}
$$

The solver restarts and, since no constraint propagates at decision level zero, it will decide on a. However, as we will show in the next section, this can be done much better.

4 Symbolic Conflict Analysis in CDCL-Based Pseudo-Boolean Solvers

This section starts with the presentation of our procedure. Then, in Section 4.2 we explain how to obtain upper bounds and how to use it inside a binary-search solving procedure. Next, we explain in Section 4.3 how the procedure interacts with other inference rules commonly used in PB solvers.

4.1 The Symbolic Conflict Analysis Procedure

Let us consider again the maximization problem in Example 4. After every solution found, constraint C_4, which states that only better solutions are accepted, is strengthened. However, we know that lemma C_5 was obtained via the linear combination $1 \cdot C_4 + 6 \cdot C_3$, and lemma

C_6 was $1 \cdot C_4 + 6 \cdot C_3 + 4 \cdot C_2$. What we show in this section is how, after every solution is found, not only can we strengthen C_4 but also strengthen every lemma that was derived by using C_4. Moreover, we can do it in a simple and efficient way that only requires minor modifications to the solver. Let us introduce our *symbolic conflict analysis* procedure with the following example.

▶ **Example 5.** Let us consider the same maximization problem as in Example 4.

First solution. We proceed as before in order to find the first solution, with value 0.

Second solution. Now, instead of adding constraint $C_4 := x + 2y + 3z + 4u + 5v + 6w \geq 1$, we modify its degree so that it contains a symbolic representation of it. We consider a symbolic variable δ and instead add constraint $C_4 := x + 2y + 3z + 4u + 5v + 6w \geq 1 \; [\![1 + \delta]\!]$. The symbolic part $[\![1 + \delta]\!]$ intuitively expresses that the objective function has to be at least 1 unit larger than the value of the best solution found so far, which is symbolically represented by a variable δ. In terms of propagation, this symbolic expression will be ignored in all constraints.

Hence, we have the same constraints as in the previous example and the solver constructs $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2, c^d, \overline{v}_3, \overline{w}_3)$. Our symbolic conflict analysis is started, proceeding as usual except when performing linear combinations, where the symbolic part will be taken into account. As before, we have to compute $1 \cdot C_4 + 6 \cdot C_3$. Note that input constraints like C_3 also have a symbolic representation of the degree. However, in those cases it will be equal to the degree. That is, the linear combination is:

$1 \cdot C_4$:		x	$+2y$	$+3z$	$+4u$	$+5v$		$+6w$	$\geq$	1	$[\![1 + \delta]\!]$
$6 \cdot C_3$:	$6\overline{c}$					$+6\overline{v}$		$+6\overline{w}$	$\geq$	12	$[\![12]\!]$
C_5 :	$6\overline{c}$	$+x$	$+2y$	$+3z$	$+4u$	$+(\overline{v} + 5)$		$+6$	$\geq$	13	$[\![13 + \delta]\!]$

A simple arithmetic manipulation, also on the symbolic part, results in the final learned lemma $C_5 : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 2 \; [\![2 + \delta]\!]$. Since the symbolic representation is ignored when propagating, the solver proceeds as in the previous example to obtain solution $\rho = (a^d, \overline{x}_1, \overline{y}_1, b^d, \overline{z}_2, \overline{u}_2, \overline{c}_5, v^d, \overline{w}_3)$ with cost 5.

Third solution. This is where our procedure makes the solver behave differently. Since we are looking for a solution of cost larger than 5, we strengthen C_4 to become $x + 2y + 3z + 4u + 5v + 6w \geq 6 \; [\![1 + \delta]\!]$ but we can also automatically strengthen C_5 to become $6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 7 \; [\![2 + \delta]\!]$. More explicitly, the new degree of a constraint is equal to symbolic part where variable δ is replaced by the value of the best solution found so far. We can observe that C_5 allows for more propagations than its initial non-strengthened version. However none of these propagations apply here and the solver finds the same solution as before, with value 6.

Fourth solution. Constraints C_4 and C_5 are strengthened again to become now C_4 : $x + 2y + 3z + 4u + 5v + 6w \geq 7 \; [\![1 + \delta]\!]$ and $C_5 : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 8 \; [\![2 + \delta]\!]$. By making the same decisions, the solver finds again the same conflict and the first linear combination to be computed is again $6 \cdot C_3 + 1 \cdot C_4$, which gives $C_{tmp} : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 8 \; [\![2 + \delta]\!]$.

This constraint does not propagate at any previous decision level and one additional iteration is needed. The following linear combination is $1 \cdot C_{tmp} + 4 \cdot C_2$, which results in $C_6 : 4\overline{b} + 6\overline{c} + x + 2y + \overline{z} + \overline{v} \geq 9 \; [\![3 + \delta]\!]$. It propagates $\overline{c}$ and $\overline{b}$ at decision level 1 and the solver proceeds identically to obtain a solution of value 9.

Fifth solution. Constraints C_4, C_5 and C_6 are strengthened again and become

$$\begin{array}{ll} C_4 : x + 2y + 3z + 4u + 5v + 6w \geq 10 & [\![1 + \delta]\!] \\ C_5 : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 11 & [\![2 + \delta]\!] \\ C_6 : 4\overline{b} + 6\overline{c} + x + 2y + \overline{z} + \overline{v} \geq 12 & [\![3 + \delta]\!] \end{array}$$

These constraints are now much stronger than in the previous example due to symbolic conflict analysis. This clearly pays off now: C_6 propagates $\overline{b}$ and $\overline{c}$ at decision level zero, something that was not possible in Example 4, where our symbolic conflict analysis procedure was not used. The solver now decides on a, which propagates $\overline{x}$ and $\overline{y}$ due to C_1. Since we now have the strengthened version of C_5, we can propagate u. Unit propagation ends up finding a solution $\rho = (\overline{b}_6, \overline{c}_6, a^d, \overline{x}_1, \overline{y}_1, u_5, \overline{z}_2, w_4, \overline{v}_3)$ with value 10.

Sixth solution. Constraints C_4, C_5 and C_6 are strengthened again and become now:

$$\begin{array}{ll} C_4 : x + 2y + 3z + 4u + 5v + 6w \geq 11 & [\![1 + \delta]\!] \\ C_5 : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 12 & [\![2 + \delta]\!] \\ C_6 : 4\overline{b} + 6\overline{c} + x + 2y + \overline{z} + \overline{v} \geq 13 & [\![3 + \delta]\!] \end{array}$$

After the propagation of $\overline{b}, \overline{c}$ at decision level zero, the solver decides on a and unit propagation produces the assignment $\rho = (\overline{b}_6, \overline{c}_6, a^d, \overline{x}_1, \overline{y}_1, u_5, z_5)$ with value 10. This now falsifies C_2 and conflict analysis is triggered. So z is popped from ρ, and the linear combination $3 \cdot C_2 + 1 \cdot C_5$ is performed, giving $C_{tmp} : 3\overline{b} + 6\overline{c} + x + 2y + u + \overline{v} \geq 12\ [\![2 + \delta]\!]$. Since it does not propagate at decision level zero, conflict analysis continues. Then u and $\overline{y}$ are popped and the linear combination $1 \cdot C_{tmp} + 2 \cdot C_1$ is performed, resulting in $C_7 : 2\overline{a} + 3\overline{b} + 6\overline{c} + \overline{x} + u + \overline{v} \geq 13\ [\![3 + \delta]\!]$, which propagates $\overline{a}$ at decision level zero. The solver backjumps to $\rho = (\overline{b}_6, \overline{c}_6, \overline{a}_7)$. Since nothing else propagates, the solver decides on v and unit propagation results in solution $\rho = (\overline{b}_6, \overline{c}_6, \overline{a}_7, v^d, \overline{w}_3, y_6, \overline{x}_1, u_4, \overline{z}_2)$ with value 11.

Seventh solution. Constraints C_4, C_5, C_6 and C_7 are strengthened again and become now:

$$\begin{array}{ll} C_4 : x + 2y + 3z + 4u + 5v + 6w \geq 12 & [\![1 + \delta]\!] \\ C_5 : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 13 & [\![2 + \delta]\!] \\ C_6 : 4\overline{b} + 6\overline{c} + x + 2y + \overline{z} + \overline{v} \geq 14 & [\![3 + \delta]\!] \\ C_7 : 2\overline{a} + 3\overline{b} + 6\overline{c} + \overline{x} + u + \overline{v} \geq 14 & [\![3 + \delta]\!] \end{array}$$

These strengthenings allow the solver to compute the optimal solution only by unit propagation $\rho = (\overline{b}_6, \overline{c}_6, \overline{a}_7, \overline{x}_7, u_7, \overline{v}_7, \overline{z}_2, y_4, w_4)$, which has value 12.

Summing up, there are only three differences with respect to the underlying algorithm of a CDCL-based PB solver. First of all, PB constraints now have a symbolic representation of their degree. Note that this representation will always be of the form $[\![p \cdot \delta + q]\!]$, where $p, q \in \mathbb{Q}$. We cannot guarantee that they are natural numbers because of the use of the division rule that we will explain later. This symbolic representation is updated accordingly when the linear combination rule is applied. The second difference is that, when the first solution is found (with value V) a new constraint with symbolic part $[\![\delta + 1]\!]$ and degree $V + 1$ is added. The third difference is that when a new solution is found with value V, the degree of all constraints is updated. More concretely, if a constraint has as symbolic part $[\![p \cdot \delta + q]\!]$ its degree will become $p \cdot V + q$. The following theorem guarantees the correctness of our procedure.

▶ **Theorem 6.** *Let us consider S a set of PB constraints and $\sum_i o_i l'_i$ a PB objective function to maximize. If the symbolic constraint $\sum_i c_i l_i \geq [\![p \cdot \delta + q]\!]$ can be derived by a series of symbolic linear combination steps from S and the symbolic constraint $\sum_i o_i l'_i \geq [\![\delta + 1]\!]$, then $\sum_i c_i l_i \geq p \cdot V + q$ can be derived by a series of linear combination steps from S and $\sum_i o_i l'_i \geq V + 1$.*

4.2 Additional Benefits of Symbolic Conflict Analysis

In the previous section we showed the main benefit of symbolic conflict analysis: the strengthened constraints have stronger unit propagation capabilities and hence allow for a more efficient traversal of the search space. We now report on additional benefits that we obtain by using our symbolic conflict analysis procedure.

Computation of Upper Bounds. Somewhat surprisingly, a by-product of the symbolic procedure we have introduced is that we can easily compute upper bounds on the value of the objective function we want to maximize.

Again, the idea is best illustrated by revisiting Example 5. Let us consider constraint C_5 in its original form $6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 2\ [\![2 + \delta]\!]$. Assume that a solution with value V is found. Our procedure would automatically update the degree of C_5 and it would become, removing its symbolic part for the sake of this reasoning, $6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 2 + V$. The left-hand side of this inequality can at most be 17 (the sum of its coefficients). This means that if $2 + V \geq 18$ (i.e. $V \geq 16$) this constraint cannot be satisfied and hence the solver would conclude that there is no solution strictly larger than V. Hence the best objective can at most be 16. Thus, after second solution (with value 5) is found in Example 5, we can guarantee that the optimal solution is in the interval $[5, 16]$.

▶ **Theorem 7.** *Let us consider $\mathcal{P}$ a PB maximization problem consisting of a set S of PB constraints and an objective function $\sum_i o_i l'_i$. If we can derive by a series of symbolic linear combination steps a symbolic constraint $\sum_i c_i l_i \geq [\![p \cdot \delta + q]\!]$ from S and $\sum_i o_i l'_i \geq [\![\delta + 1]\!]$, then the value of the optimal solution to $\mathcal{P}$ is at most $\left\lfloor \frac{(\sum_i c_i) - q}{p} + 1 \right\rfloor$.*

Proof. Let O be the value of the optimal solution to $\mathcal{P}$. Let us consider the set of constraints $S' = S \cup \{\sum_i o_i l'_i \geq O\}$, which we know is satisfiable. Theorem 6 and the premises of this theorem guarantee that a series of linear combination steps exists that allow us to derive from S' the constraint $\sum_i c_i l_i \geq p \cdot (O - 1) + q$. Since linear combination produces logical consequences, the latter constraint must be satisfiable. Hence O must be such that $\sum_i c_i \geq p \cdot (O - 1) + q$, which is the same as $O \leq \frac{(\sum_i c_i) - q}{p} + 1$. Since O must be an integer value, we can take the floor of the right-hand side to obtain the desired bound. ◀

In Example 5, the previous theorem allows the symbolic conflict analysis procedure to produce an upper bound of 13 due constraint C_6, and a better upper bound of 12 due to C_7.

An interesting question is whether the derivation of upper bounds can trigger an early termination of the procedure because the best solution found matches the computed upper bound. We now prove that, in this situation, the constraint from which the upper bound is computed becomes trivially unsatisfiable after the corresponding strengthening step.

▶ **Theorem 8.** *Let us consider the constraint $C := \sum_i c_i l_i \geq [\![p \cdot \delta + q]\!]$ from which the upper bound θ is derived. If a solution with value θ is obtained, the corresponding strenghtened version of C is unsatisfiable.*

Proof. Due to theorem 7, we know that $\theta = \left\lfloor \frac{(\sum_i c_i)-q}{p} + 1 \right\rfloor$. If we find a solution with value θ we can strengthen C to obtain $\sum_i c_i l_i \geq p \cdot \left\lfloor \frac{(\sum_i c_i)-q}{p} + 1 \right\rfloor + q$. We can manipulate the degree of the constraint as follows:

$$p\left\lfloor \frac{(\sum_i c_i) - q}{p} + 1 \right\rfloor + q = p\left\lfloor \frac{(\sum_i c_i) - q}{p} \right\rfloor + p + q = p\left(\frac{(\sum_i c_i) - q}{p} - \varepsilon \right) + p + q = \sum_i c_i + p(1-\varepsilon)$$

for some $0 \leq \varepsilon < 1$. Hence, the strengthened constraint is $\sum_i c_i l_i \geq \sum_i c_i + p(1-\varepsilon)$ which is clearly unsatisfiable. ◀

Use in a Binary-Search Approach. The standard CDCL-based PB procedure we revisited in Example 4 strengthens the degree of the objective function every time a solution is found in order to find a better one. As mentioned before, this is sometimes known as Linear Search. However, we can use the well-known binary-search approach in which we always have an interval $[LB, UB]$ that includes the value of the optimal solution. If $LB = UB$ the optimal has been found. Otherwise we pick $M = (LB + UB)/2$ and recursively solve two problems, the first one in which a solution is sought in $[LB, M]$, and a second problem in which a solution is sought in $[M + 1, UB]$.

One of the main drawbacks of this approach is that constraints learned when solving the $[M + 1, UB]$ problem cannot be reused for the $[LB, M]$ problem, because they might have been derived by using the constraint $obj \geq M + 1$ that expresses that a solution of value at least $M + 1$ is needed, which is a not a valid constraint in the $[LB, M]$ problem.

However, this problem can easily be overcome in our symbolic conflict analysis procedure. First of all, we can easily identify the constraints that depend on $obj \geq M + 1$: the ones with a non-zero coefficient on δ in the symbolic part. But, even more importantly, we can even reuse those ones for solving the $[LB, M]$ problem by substituting δ by LB in the symbolic part and changing the degree of the constraint to that value.

4.3 Interaction with Other Inference Rules

As we have mentioned, our presentation in Algorithm 1 of the conflict analysis procedure used by CDCL-based PB solvers omitted some details in order to (i) ease its understanding and (ii) do not bias it towards any concrete solver. One of the ingredients that our presentation abstracts away is the use of other rules apart from linear combination. In particular, as explained in [15], the use of division and saturation allows solvers to process some problems exponentially faster. Apart from this important effect, they also allow the solver to deal with smaller coefficients in the constraints, which may become very large after a few linear combinations are applied. The recurrent use of infinite-precision arithmetic in a solver might have remarkable negative effects on performance.

The Division Rule. This rule can be smoothly adapted to be used in our symbolic conflict analysis procedure. Whenever a constraint $\sum_i a_i l_i \geq A$ is divided by a positive natural number α to obtain $\sum_i \lceil a_i/\alpha \rceil l_i \geq \lceil A/\alpha \rceil$ its symbolic part $[\![p \cdot \delta + q]\!]$ is also divided by the same constant to become $[\![\frac{p}{\alpha} \cdot \delta + \frac{q}{\alpha}]\!]$. This is the reason why coefficients in the symbolic part might be rational numbers. When a new solution with value V is found and the constraint needs to be strengthened, its degree will become the ceiling $\lceil \frac{p}{\alpha} \cdot V + \frac{q}{\alpha} \rceil$.

The Saturation Rule. Another common rule in PB solvers is saturation, which forces all coefficients of a constraint to be at most equal to its degree. In general, the use of this rule is incompatible with our symbolic procedure. This is easy to see if we consider again Example 5. The first lemma that is learned is $C_5 : 6\overline{c} + x + 2y + 3z + 4u + \overline{v} \geq 2\ [\![2 + \delta]\!]$. At this point, we could apply saturation to obtain $C_5 : 2\overline{c} + x + 2y + 2z + 2u + \overline{v} \geq 2\ [\![2 + \delta]\!]$. However, after the fourth solution (with value 9) is found, the procedure would update the degree of this constraint to be $2 + 9 = 11$, which would make the constraint unsatisfiable. Hence, the solver would wrongly conclude that the value of the optimal solution is 9.

We want to remark that saturation is still applicable to all constraints where δ has zero coefficient on the symbolic part, i.e., the ones that do not depend on the objective function. Regarding the rest of the constraints, there are a few possibilities: (i) apply saturation but remove the symbolic part or (ii) do not apply saturation, or (iii) apply saturation but store the original coefficients of the constraints in order to be used later when a strengthening step is performed.

5 Experimental Evaluation

In order to empirically evaluate the impact of our symbolic conflict analysis procedure, we have run experiments on the set of benchmarks that were used in the OPT-LIN (optimization problems with linear constraints) track of the 2024 Pseudo-Boolean Competition[2]. Since our solver does not accept big integers on the constraints, we limit our analysis to the 385 benchmarks that only use 32-bit integers. All experiments[3] were done on 3.3Ghz 16GB Intel Xeon E-2124 machines, setting a time limit of one hour per benchmark. We report our findings in what follows.

Potential and Overhead of Symbolic Conflict Analysis. After every new solution is found, except for the first one, our symbolic conflict analysis procedure performs a strengthening step: it changes the degree of all symbolic constraints. Hence, having a systematic low number of strengthening steps on all benchmarks would indicate that our procedure is rarely applicable. In order to confirm that this pessimistic scenario is not common we computed, for all 385 benchmarks, how many such steps are done within one hour. This information is summarized in the leftmost cumulative plot of Figure 1.

A point (x, y) in the plot means that there exist $y\%$ of the benchmarks for which at least x strengthening steps are applied. For example, around 50% of the benchmarks have at least 8 steps, one third of them has at least 15 steps, and 20% of the benchmarks have at least 24 steps. Hence, the important message one has to infer from the plot is that it is very common that a non-negligible number of strengthening steps are used. There are some instances, left out of the plot for readability, with an unusually large number of steps: 4 benchmarks with between 100 and 150 steps, 2 benchmarks between 151 and 200, and 2 benchmarks with more than 201, being 255 the maximum one. There are also extreme cases on the opposite direction: for about 18% of the benchmarks no strengthening step was applied. This is not a huge number but we believe it is large enough so that it is not negligible: doing expensive computations to later realize that our technique is not applicable at all would definitely slow down the solver on around 20% of the instances.

[2] `https://www.cril.univ-artois.fr/PB24/`
[3] Additional material can be found in `https://github.com/dearzhaorui/symbolic-conflict-analysis`.

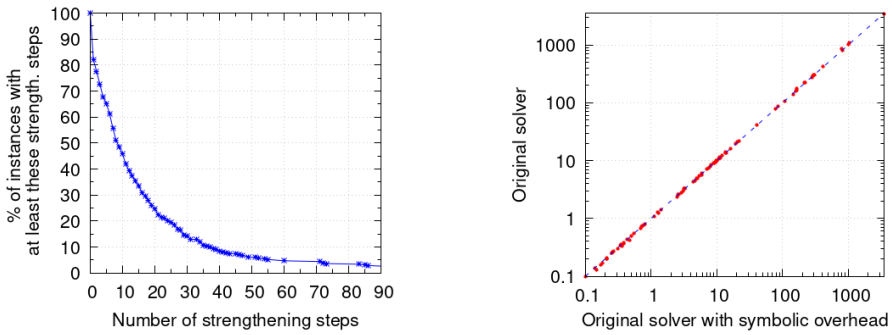

Figure 1 Left plot displays the percentage of benchmarks with at least a certain number of strengthening steps. The plot on the right shows the negligible overhead of the symbolic conflict analysis procedure.

Fortunately, the rightmost scatter plot of Figure 1 reveals how surprisingly small the overhead of symbolic conflict analysis is. For precisely assessing this overhead, we implemented our symbolic procedure on top of our solver: all constraints have a symbolic part; linear combination, saturation and division are applied taking into account this symbolic part but, after a new solution is found, the strengthened degree of all symbolic constraints is computed but not updated. That is, we pay all the overhead of our procedure but, since the constraints are unchanged, the solver has the same search behavior as its baseline version. The scatter plot compares the runtime in seconds of this modified system and our original solver. It is entirely obvious that the overhead caused by the symbolic procedure is negligible. We want to remark that this is the case independently of the package we use for infinite-precision rational numbers on the symbolic part: we tried GMP [14], Boost [5] and a custom package developed in our research group and no differences were observed.

Runtime Improvement. Let us now show that implementing symbolic conflict analysis on top of our solver produces important performance gains. The CDF plot on the left of Figure 2 shows how the addition of the symbolic conflict analysis procedure makes the solver able to solve more problems within any given time limit. For a time limit of 1 hour the solver is able to solve 6 more benchmarks up to optimality. The scatter plot on the right contains two parallel green lines delimiting speedups of 2x. We can see that despite there are concrete instances for which the symbolic solver is slower, they are mostly easy instances solved within 30 seconds, and the slowdown is never larger than 2x. On the other hand, there are several benchmarks for which the symbolic solver outperforms the baseline by very large speedup factors.

Finally, we want to compare our system with the state of the art. For this purpose we chose the latest version of RoundingSat [11, 9, 8]. Note that, in the optimization linear constraints category of the 2024 Pseudo-Boolean Competition, 8 of the 10 best performing solvers used RoundingSat, either as an oracle or combined with additional techniques. We ran RoundingSat in linear-search mode. Despite its full version, which combines linear and core-guided search is more powerful, the purpose of this comparison is limited to linear search. The CDF plot on Figure 3 shows how our original solver is slower than RoundingSat. This might be due to maturity of the implementation, different search strategies and, more importantly, concrete details in conflict analysis related to different weakening procedures, different uses of saturation and division, and termination criteria of the main loop as we explained in Section 3. However, the addition of symbolic conflict analysis bridges this gap

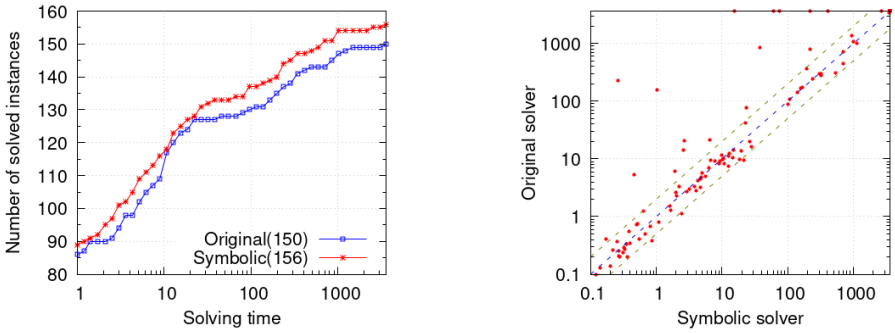

Figure 2 CDF and scatter plots comparing runtimes of our original solver with a version of it implementing symbolic conflict analysis.

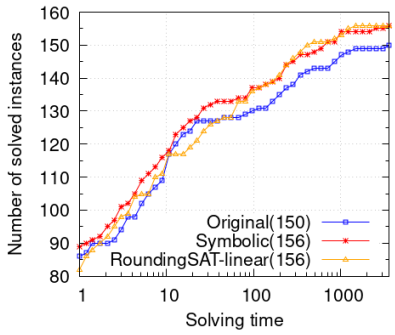

Figure 3 CDF plot comparing RoundingSat, our original solver and our implementation of symbolic conflict analysis on top of it.

and makes them behave more similarly. As we will mention, it is part of our future work to examine how symbolic conflict analysis can improve the linear-search component of other solvers, and RoundingSat is definitely a candidate.

In the optimization linear constraints category, 8 of the 10 best performing solvers did use RoundingSAT. That is, it was either used as an oracle or combined with additional techniques. Hence, it is at the core of almost every solver, except for SCIP.

Percentage of Symbolic Lemmas and Impact of Saturation. The next question we want to address is to determine how many lemmas have a non-constant symbolic part, that is, $[\![p \cdot \delta + q]\!]$ with $p \neq 0$. The left histogram in Figure 4 summarizes this information. For each of the 312 out of the overall 385 benchmarks that apply some strengthening step, whenever this step is performed we compute the percentage of PB constraints that have a non-constant symbolic part. Note that in our solver clauses have a particular treatment and they are not considered as PB constraints in this computation. When the execution finishes, we compute the average over all strengthening steps of these percentages. The histogram essentially represents this average. More concretely, the first bar indicates that for 34 benchmarks this average is zero. The second bar shows that for 65 benchmarks this average is in the interval $(0, 5]$; the third bar over the 10 label means that the average is in (5,10] for 13 benchmarks, and so on. As expected from the previous results on performance, the percentage of constraints with non-constant symbolic part is remarkable, making the strengthening step a powerful one.

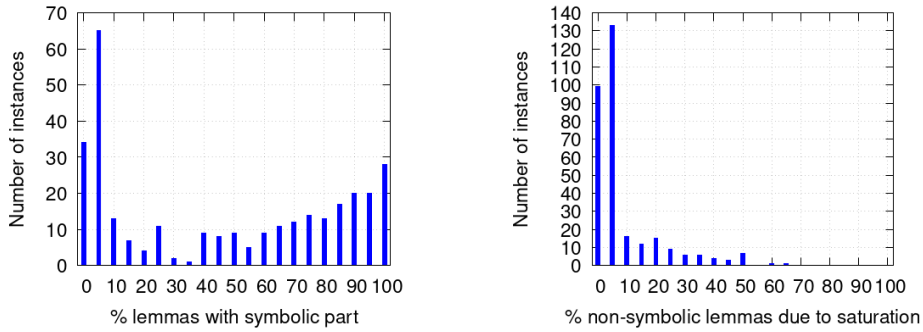

Figure 4 The left histogram shows the number of instances with a certain percentage of constraints with non-constant symbolic part. The right histogram show the percentage of lemmas that are non-symbolic due to an application of saturation.

Another question we wanted to answer is to discover the reasons why certain lemmas are non-symbolic (i.e. $p = 0$ in $[\![p \cdot \delta + q]\!]$). One obvious reason is that it might happen that the constraint bounding the objective function has not been used in the derivation of this lemma. Another possibility is the interaction between saturation and our symbolic procedure, as we explained in Section 4.3. In our implementation, whenever saturation is to be applied on a constraint with non-constant symbolic part, we apply it and remove its symbolic information. If this happens too often, it could hinder the power of our symbolic reasoning. The right histogram in Figure 4 answers this question in a positive way: this situation is not common at all. For example, the bar with height 15 over the 20 label means that there are 15 instances for which, if we only consider non-symbolic constraints, the percentage of them that are non-symbolic due to an application of saturation is in the interval $(15, 20]$. As we can see, the main reason for a constraint not being symbolic is not the application of saturation.

Upper Bounding. An interesting feature of our procedure is the computation of upper bounds. The number of times that our procedure improves the upper bound is significant for several benchmarks. In the left plot of Figure 5 we can see, for three concrete instances, the evolution of the upper and the lower bound. Since these instances are minimization problems, finding a new solution improves the upper bound, whereas our new bounding techniques improve the lower bound. We can see that both bounds are improved several times during the execution of the system. However, we want to note that there are of course instances for which the production of new bounds is not as frequent. This is observed in the right plot of Figure 5. We have considered all 87 benchmarks for which our solver timed out in one hour and such that after the last solution found, our bounding technique is able to reduce the size of the interval $[LB, UB]$. The plot displays the number of instances for which the size of the last interval produced by the solver, divided by the size of the interval after the last solution was found is equal to a certain number. A small number implies that our bounding technique is able to produce a significant reduction on the interval size since the last found solution. As we can see, although for half of the benchmarks only small reductions are achieved, for the other half it does have a beneficial effect, with different magnitudes depending on the benchmark. This shows that, even when the solver is not able to find further solutions, the bounding technique still makes progress in the estimation of the optimum.

Binary Search. It is well known that in PB optimization, binary search is generally outperformed by linear search. Unfortunately, according to our experiments, this is still the case even if one is able to reuse all lemmas from all problems, as we can do thanks to our symbolic procedure.

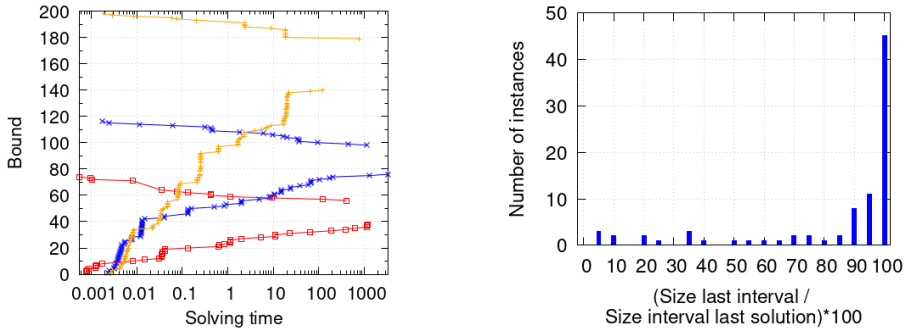

Figure 5 The left plot shows the evolution of the lower and upper bounds for three different instances. On the right, we can see the percentage of reduction of the interval $[LB, UB]$ after the last found solution.

6 Conclusions and Future Work

We have introduced a novel symbolic conflict analysis procedure for PB optimization. Despite being easy to implement and having no overhead, experimental results have shown that it is able to significantly improve the performance of our linear-search based solver. In addition, it has several benefits such as the computation of upper bounds and its use in binary search.

As future work, we plan to evaluate the impact of our technique on other PB solvers. Finding ways to improve our bound computation is also on our agenda, as well as developing proof-logging techniques to verify the correctness of the results given by this technique.

References

1 Elvira Albert, Pablo Gordillo, Alejandro Hernández-Cerezo, and Albert Rubio. A max-smt superoptimizer for EVM handling memory and storage. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 201–219. Springer, 2022. doi:10.1007/978-3-030-99524-9_11.

2 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. doi:10.1613/jair.3152.

3 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 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, 2020. doi:10.1007/978-3-030-51825-7_23.

4 Armin Biere and Daniel Kröning. Sat-based model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 277–303. Springer, 2018. doi:10.1007/978-3-319-10575-8_10.

5 The Boost C++ libraries. Available at https://www.boost.org/.

6 Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(3):305–317, 2005. doi:10.1109/TCAD.2004.842808.

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

8 Jo Devriendt, Ambros M. Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-boolean conflict-driven search. *Constraints An Int. J.*, 26(1):26–55, 2021. doi:10.1007/S10601-020-09318-X.

9 Jo Devriendt, Stephan Gocht, Emir Demirovic, Jakob Nordström, and Peter J. Stuckey. Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3750–3758. AAAI Press, 2021. doi:10.1609/AAAI.V35I5.16492.

10 Julian Dolby, Mandana Vaziri, and Frank Tip. Finding Bugs Efficiently With a SAT Solver. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 195–204, 2007. doi:10.1145/1287624.1287653.

11 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1291–1299. ijcai.org, 2018. doi:10.24963/ijcai.2018/180.

12 Nick Feng and Fahiem Bacchus. Clause size reduction with all-uip learning. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2020. doi:10.1007/978-3-030-51825-7_3.

13 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006. doi:10.1007/11814948_25.

14 The GNU MP Bignum Library. Available at https://gmplib.org/.

15 Stephan Gocht, Jakob Nordström, and Amir Yehudayoff. On division versus saturation in pseudo-boolean solving. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1711–1718. ijcai.org, 2019. doi:10.24963/IJCAI.2019/237.

16 Ana Graça, João Marques-Silva, Inês Lynce, and Arlindo L. Oliveira. Efficient haplotype inference with combined CP and OR techniques. In Laurent Perron and Michael A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008, Paris, France, May 20-23, 2008, Proceedings*, volume 5015 of *Lecture Notes in Computer Science*, pages 308–312. Springer, 2008. doi:10.1007/978-3-540-68155-7_28.

17 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2 & 3):297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

18 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

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

20 Dejan Jovanovic and Leonardo Mendonça de Moura. Cutting to the chase – solving linear integer arithmetic. *J. Autom. Reasoning*, 51(1):79–108, 2013. doi:10.1007/s10817-013-9281-x.

21 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for maxsat. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 38:1–38:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CP.2021.38.

22 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Boosting branch-and-bound maxsat solvers with clause learning. *AI Commun.*, 35(2):131–151, 2022. doi:10.3233/AIC-210178.

23 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. 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 133–182. IOS Press, 2021. doi:10.3233/FAIA200987.

24 António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints An Int. J.*, 18(4):478–534, 2013. doi:10.1007/S10601-013-9146-2.

25 Robert Nieuwenhuis and Albert Oliveras. On SAT Modulo Theories and Optimization Problems. In *Proceedings of SAT '06*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006. doi:10.1007/11814948_18.

26 Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Emma Rollon. Employee scheduling with sat-based pseudo-boolean constraint solving. *IEEE Access*, 9:142095–142104, 2021. doi:10.1109/ACCESS.2021.3120597.

27 Tobias Paxian and Bend Becker. Pacose: An Iterative SAT-based MaxSAT Solver. In Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian, editors, *MaxSAT Evaluation 2024 : Solver and Benchmark Descriptions*, Department of Computer Science Series of Publications B, page 26. University of Helsinki, 2024. URL: http://hdl.handle.net/10138/584878.

28 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

29 Albert Oliveras Robert Nieuwenhuis and Enric Rodríguez-Carbonell. Intsat: integer linear programming by conflict-driven constraint learning. *Optimization Methods and Software*, 2023. doi:10.1080/10556788.2023.2246167.

30 J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM, JACM*, 12(1):23–41, 1965. doi:10.1145/321250.321253.

31 Olivier Roussel and Vasco M. Manquinho. Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in AI and Applications*, pages 695–733. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-695.

32 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-boolean optimization by implicit hitting sets. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 51:1–51:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CP.2021.51.

33 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Improvements to the implicit hitting set approach to pseudo-boolean optimization. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SAT.2022.13.

34 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing – SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 – July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

35 Yichen Xie and Alexander Aiken. Saturn: A SAT-Based Tool for Bug Detection. In *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005*, pages 139–143, 2005. doi:10.1007/11513988_13.

36 L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *International Conference on Computer-Aided Design, ICCAD '01*, pages 279–285, 2001.