Symmetry and Dominance Breaking for Pseudo-Boolean Optimization

Van Caudenberg, Daimy Stefanie; Bogaerts, Bart

Published in:
Artificial Intelligence and Machine Learning

DOI:
10.1007/978-3-031-39144-6_10

Publication date:
2023

Document Version:
Accepted author manuscript

Symmetry and Dominance Breaking for Pseudo-Boolean Optimization*

Daimy Van Caudenberg and Bart Bogaerts

Vrije Universiteit Brussel, Brussels, Belgium
daimy.stefanie.van.caudenberg@vub.be
bart.bogaerts@vub.be

Abstract. It is well-known that highly symmetric problems can often be challenging for combinatorial search and optimization solvers. One technique to avoid this problem is to introduce so-called symmetry breaking constraints, which eliminate some symmetric parts of the search space. In this paper, we focus on *pseudo-Boolean optimization problems*, which are specified by a set of 0–1 integer linear inequalities (also known as *pseudo-Boolean constraints*) and a linear objective. Symmetry breaking has already been studied in this context; however previous work could only deal with symmetries of the entire optimization problem. In this paper, we show how to handle *weak symmetries*: symmetries of the constraints that do not necessarily need to respect the objective. We show that weak symmetries induce a *dominance relation* and that pseudo-Boolean constraints are a natural target formalism to write the dominance breaking constraints in. We implemented these ideas on top of a state-of-the-art symmetry breaking tool for SAT, and in doing so also transfer modern symmetry breaking techniques to pseudo-Boolean optimization. We experimentally validate our approach on the latest pseudo-Boolean competition, as well as on hard combinatorial instances and conclude that the effect of breaking (weak) symmetries depends greatly on the type of solving algorithm used.

1 Introduction

Hard combinatorial decision and optimization problems often exhibit symmetries. It is well-known that when these symmetries are not properly taken into account, solvers can easily get stuck exploring many symmetric versions of the search space. To overcome this limitation, a variety of techniques has been developed; we here make the distinction between *static* (prior to the search) and *dynamic* (during search) symmetry handling techniques.

Static symmetry handling techniques all perform *symmetry breaking*: that is, they add new constraints that eliminate some, but not all symmetric assignments. One way to achieve this is by adding so-called *lex-leader constraints*,

* This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (FWO) (project G070521N); computational resources and services were provided by the VSC (Flemish Supercomputer Center), funded by FWO and the Flemish Government.

which discard all assignments that are lexicographically larger than their symmetric counterpart. This type of symmetry breaking (with lex-leader or other constraints) is used in various fields [11, 22, 23, 40, 3, 27, 30, 14, 16, 20, 13, 17].

Dynamic symmetry handling techniques come in different flavours. Most methods work on a set of symmetries of the input formula, but some also detect symmetries during the search [5]. Some methods actually *break* the symmetries (and hence potentially also remove satisfying assignments), e.g., by adopting a branching method that takes symmetries into account [36], or by lazily posting the symmetry breaking constraints that static techniques would also add [32, 28]. Other methods only eliminate unsatisfiable regions of the search space [7]; the most common such technique is *symmetric learning*, which allows a solver to learn symmetric versions of learned clauses or no-goods when constructing an unsatisfiability proof [26, 37, 6, 31, 18, 15].

The advantage of static techniques is that they are solver- and algorithm-agnostic: they can be combined with any type of search algorithm and hence are also compatible with future developments. The main drawback of static techniques is that it is difficult to know in advance which symmetry breaking constraints will be effective (will contribute to decreasing the search space), while adding all possible symmetry breaking constraints is too costly.

In this paper, we study static symmetry breaking techniques for *pseudo-Boolean optimization*, where we are given a set F of 0–1 integer linear inequalities and a linear term f (i.e. respectively the pseudo-Boolean constraints and the objective) and are asked to search for a solution to F that minimizes f. We are not the first to study symmetries in this setting: already in 2004, Aloul et al. [2] extended Shatter [3] (a symmetry breaking preprocessor for SAT) to pseudo-Boolean optimization. They detect symmetries σ of the entire optimization problem: permutations (of literals) σ such that $\sigma(F) = F$ and $\sigma(f) = f$, i.e. $\sigma(F)$ is syntactically equal to F and similarly for f. For such symmetries, they add (a clausal encoding of) lex-leader constraints: a set of clauses that expresses that a solution should be lexicographically smaller than its symmetric counterpart. In this paper, we will show that many of the techniques can be generalized to a more general setting. We will define a *weak symmetry* of (F, f) to be a symmetry of F, that not necessarily preserves f. Some care is needed when adding breaking constraints in this setting. We will show that weak symmetries give rise to a *dominance relation* [10] and will describe constraints that break weak symmetries in a sound way, roughly expressing that we are only interested in assignments that

- are at least as good (in terms of the objective) as their symmetric counterpart, and
- are lexicographically smaller than (or equal to) their symmetric counterpart, in case they have the same objective value.

As it turns out, this can easily be expressed as pseudo-Boolean constraints.

We implemented this idea on top of BreakID, a symmetry breaking preprocessor for SAT [16]. The resulting tool (which we call BreakIDPB) differs from Shatter in the following ways:

- first and foremost, it can handle weak symmetries,
- it integrates all the improvements of BreakID [16], such as detection of so-called *row-interchangeability*, and
- it can break symmetries with a native pseudo-Boolean encoding (rather than a clausal encoding).

We experimentally validate our techniques on the latest pseudo-Boolean competition, as well as on hard combinatorial instances. Our results are mixed: depending on which type of search algorithm is used to solve the resulting problem (after symmetry breaking), we see that (weak) symmetry breaking can have a very positive or a very negative impact. Moreover, we observed an interesting side-effect where weak symmetry breaking had a positive impact on instances that do not exhibit weak symmetries.

The rest of this paper is structured as follows. In Section 2 we present the necessary background for understanding our work. Section 3 describes the framework of dominance relations and how so-called *weak symmetries* fit within this framework. In Section 4 we explain how BreakID was extended into BreakIDPB and Section 5 contains the experimental results.

Publication History This paper is based on the bachelor thesis of the first author [38]. A short version of this paper was presented at the BNAIC/BeneLearn 2022 conference [39]. The current paper extends the short version with more examples, a precise description of the graph that is created, and more extensive experiments.

2 Preliminaries

In this section, we recall some standard definitions related to pseudo-Boolean optimization. A *literal* ℓ over a Boolean variable x is x itself or its negation $\overline{x} = 1 - x$, where variables take values 0 (false) or 1 (true). A *pseudo-Boolean (PB) constraint* C is a 0–1 linear inequality

$$\sum_i a_i \ell_i \geq A, \tag{1}$$

where a_i and A are integers. Without loss of generality, we will often assume that PB constraints are *normalized*; i.e., that all literals ℓ_i are over distinct variables and that the *coefficients* a_i and the *degree (of falsity)* A are non-negative, but most of the time we will not need this. A *pseudo-Boolean formula* F is a conjunction $\bigwedge_j C_j$ of PB constraints, which we can also think of as the set $\bigcup_j \{C_j\}$ of constraints in the formula, choosing whichever viewpoint seems most convenient. An *assignment* α is a function from a set of variables V to $\{0, 1\}$. An assignment is *complete* for F if it assigns a value to all the variables in F. Slightly abusing notation, we extend an assignment to literals in the natural way, by respecting negation where α is defined and being constant where α is not defined. I.e., if α is the assignment of the set of variables V we will write $\alpha(\overline{x})$ to mean $1 - \alpha(x)$ if $x \in V$ and to denote $\overline{x}$ if $x \notin V$. The (normalized)

constraint C in (1) is *satisfied* by α (denoted $\alpha \models C$) if $\sum_{\alpha(\ell_i)=1} a_i \geq A$. A PB formula is satisfied if all of its constraints are satisfied. A *pseudo-Boolean optimization problem*[1] is a tuple (F, f) with F a PB formula and f a linear objective $\sum_i w_i \ell_i$. We write f^α for the value of f in α, i.e., for $\sum_i w_i \alpha(\ell_i)$. An assignment α is an *(optimal) solution* to (F, f) if $\alpha \models F$ and $f^\alpha \leq f^\beta$ for each β that satisfies F.

Let π be a permutation of the set of literals over variables in F (i.e., a bijection on the set of literals) that respects negation. We extend π to various semantic objects in the expected way:

- pseudo-Boolean Constraints $\pi(\sum_i a_i l_i \geq A) = \sum_i a_i \pi(l_i) \geq A$,
- to pseudo-Boolean formulas: $\pi(C_1 \wedge \ldots \wedge C_n) = \pi(C_1) \wedge \ldots \wedge \pi(C_n)$,
- to linear terms: $\pi(f)(\sum_i w_i \ell_i) = \sum_i w_i \pi(\ell_i)$, and
- to assignments: $\pi(\alpha) = \alpha \circ \pi^{-1}$,

where the last, seemingly strange, condition guarantees for instance that $\alpha \models F$ if and only if $\pi(\alpha) \models \pi(F)$. We call π a *(syntactic) symmetry*[2] of F if $\pi(F) = F$ and a *strong symmetry* of (F, f) if additionally $\pi(f) = f$. In that case, α is an optimal solution of (F, f) if and only if $\pi(\alpha)$ is.

3 Weak Symmetries

So far, in the literature, only symmetries of pseudo-Boolean optimization problems that preserve the formula F and objective function f have been considered. However, we know that in many problems, symmetries of F show up that do not preserve f. Take for instance a nurse scheduling problem where large sets of nurses are interchangeable (given that some basic features, e.g. their degrees, are equal), but their preferences are not (the soft constraints, e.g. whether they prefer working during the weekends or night shifts). This gives rise to the following definition.

Definition 1. *Every symmetry ω of F is called a* weak symmetry *of the optimization problem (F, f).*

The strategy for breaking *strong* symmetries, as employed, e.g., by Aloul et al. [1] is to add *lex-leader* constraints. That is, given an order on the variables, for each detected[3] strong symmetry σ, they add a set LL_σ of pseudo-Boolean constraints such that for each total assignment α, α can be extended to a model of LL_σ if and only if α is lexicographically smaller than (or equal to) $\sigma(\alpha)$. When working with *weak symmetries*, this strategy is no longer applicable. Indeed, it might be the case that $\sigma(\alpha)$ is lexicographically smaller than α, but also has a worse objective than α. In that case, we do not want to derive constraints

[1] Decision problems are a special case where $F = 0$.

[2] More general definitions of symmetry exist, but all practical use cases of symmetry detection and elimination use syntactic symmetries.

[3] In practice, this is typically a set of generators of the group of strong symmetries.

that exclude α. In the rest of this section, we will show in detail how to handle weak symmetries. As a theoretic framework, we will use *dominance relations* as studied in the field of constraint programming [10]. In the rest of this section, we describe how to use this framework (in the context of PB formulas) to generate breaking constraints for weak symmetries, how to detect these symmetries, and how BreakID can be adapted to handle weak symmetries.

3.1 Dominance Relations

A dominance relation describes a pair of assignments, where one *dominates* the other when that assignment is better than the other in some "suitable sense". That "suitable sense" can be comparing the objective of the problem, or it can be something more complex. Similar to symmetries, these relations can be used to prune the search tree. Dominance relations can be exploited to add formulas that prevent the search from exploring non-dominant assignments, which we will call *Static Dominance Breaking*.

Definition 2. *A* dominance relation $\preceq$ (for (F, f)) *is a pre-order (a reflexive and transitive binary relation) on the set of assignments (that assign values to all variables in F and f) such that whenever $\alpha \preceq \beta$ it holds that*

- $f^\alpha \leq f^\beta$
- *if $\beta \models F$, then also $\alpha \models F$.*

I.e., the dominance relation should respect the problem specified by the couple (F, f) in the sense that non-models of F cannot dominate models of F and whenever one assignment dominates another, it should be at least as good as the other assignment in terms of the objective function. As usual, our preorder $\preceq$ induces a partial order $\prec$ (an irreflexive, transitive and asymmetric binary relation) where $\alpha \prec \beta$ if and only if $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$.

Example 1. Assume $\mathcal{S}$ is a set of strong symmetries of (F, f). In that case, the relation $\preceq_{\mathcal{S}}$ defined as

$$\{(\sigma(\alpha), \alpha) \mid \sigma \in \langle \mathcal{S} \rangle \wedge \sigma(\alpha) \leq_{lex} \alpha\}, \tag{2}$$

where $\langle \mathcal{S} \rangle$ denotes the group generated by $\mathcal{S}$, is a dominance relation for (F, f).

We now define what we mean by a dominance breaking formula. As we will see later in Proposition 1, soundness of a dominance breaking formula means that we can safely add it to F, without changing the objective value. One observation regarding this definition is that in the context of SAT and PB solving, contrary to the CP setting where our definition of dominance relation was borrowed from, formulas to break symmetries or dominance relations will typically contain fresh variables (sometimes also called *extension variables*). This is precisely why the definition that follows refers to *extensions* of an assignment α, which are assignments α' that agree with α wherever α is defined.

Definition 3. *Let $\preceq$ be a dominance relation. A set D of pseudo-Boolean constraints is called a* sound dominance breaking formula *for $\preceq$ if for each assignment α that cannot be extended to a model of D, there exists some assignment β with $\beta \prec \alpha$.*

D is called complete *if additionally whenever $\beta \prec \alpha$, α cannot be extended to a model of D.*

Rephrased, a sound dominance breaking formula is a formula that does not eliminate $\preceq$-minimal assignments; it is complete if it eliminates all non-$\preceq$-minimal assignments.

Proposition 1. *If D is a sound dominance breaking formula, then at least one optimal solution of (F, f) can be extended to an optimal solution of $(F \cup D, f)$. Hence,*

$$(F, f) \quad \textit{and} \quad (F \cup D, f)$$

have the same optimal objective value.

To see why this proposition holds, note that whenever α cannot be extended to satisfy D, this means that there exists another assignment that is *strictly* better. As a result, D only prunes assignments that are strictly dominated, and it is safe to add D to the pseudo-Boolean formula F in the sense that we are guaranteed not to prune all optimal solutions.

Example 2. If, as before, LL_σ is a set of lex-leader constraints, then

$$\bigcup_{\sigma \in \mathcal{S}} LL_\sigma$$

is a sound dominance breaking formula for the dominance relation $\preceq_{\mathcal{S}}$ from Example 1.

3.2 Weak Symmetries and Dominance Relations

The constraints in the previous example are precisely what Shatter [1] adds for breaking strong symmetries. We now turn our attention to weak symmetries by showing that they indeed also define a dominance relation and by constructing a sound dominance breaking formula for them.

Proposition 2. *Let $\mathcal{W}$ be a set of weak symmetries of (F, f). The relation $\preceq_{\mathcal{W}}$ defined as*

$$\left\{ (\omega(\alpha), \alpha) \;\middle|\; \begin{array}{l} \omega \in \langle \mathcal{W} \rangle \\ \wedge\, f^{\omega(\alpha)} \leq f^{\alpha} \\ \wedge\, (f^{\omega(\alpha)} = f^{\alpha} \Rightarrow \omega(\alpha) \leq_{lex} \alpha) \end{array} \right\}$$

is a dominance relation for (F, f).

Proof. Reflexivity follows since *id* (the identity function) is in $\langle \mathcal{W} \rangle$ (it is its neutral element). Transitivity holds since composition is internal in $\langle \mathcal{W} \rangle$. □

Now we construct a dominance breaking formula for this dominance relation. We will again assume that for each weak symmetry ω, we have PB formula LL_ω such that α can be extended to a model of LL_ω if and only if α is lexicographically smaller than (or equal to) $\omega(\alpha)$. Slightly abusing notation, we will write

$$(f = \omega(f)) \Rightarrow LL_\omega$$

to denote a PB formula that is satisfied precisely by those assignments α for which $f^\alpha \neq \omega(f)^\alpha$ or for which $\alpha \models LL_\omega$. We will later show how to construct such a formula.

The idea of our dominance breaking formula is to discard candidate solutions whose symmetric variant has a strictly better solution (according to the objective function), as well as candidate solutions whose symmetric variants have an equally good objective function value, but that are lexicographically larger than their symmetric counterpart.

Proposition 3. *Let $\mathcal{W}$ be a set of weak symmetries of (F, f). For every ω in $\mathcal{W}$, the PB formula BF_ω defined as*

$$\left\{ \begin{array}{c} f \leq \omega(f) \\ (f = \omega(f)) \Rightarrow LL_\omega \end{array} \right\} \tag{3}$$

forms a sound dominance breaking formula for $\preceq_{\mathcal{W}}$.

Intuitively, the first constraint discards candidate solutions whose symmetric variant has a strictly better solution and the second constraint discards candidate solutions whose symmetric variants have an equally good objective function value, but that are lexicographically larger than their symmetric counterpart.

Proof (Proof of Proposition 3). We only have to show soundness. Hence, assume α cannot be extended to an assignment that satisfies BF_ω. We claim that in that case $\omega(\alpha) \prec_{\mathcal{W}} \alpha$. The fact that $\omega(\alpha) \preceq_{\mathcal{W}} \alpha$ is immediate: since α cannot be extended to satisfy BF_ω, there are two possible situations:

- either $\alpha \not\models f \leq \omega(f)$, but in this case clearly $(\omega(\alpha), \alpha) \in \preceq_{\mathcal{W}}$,
- or $\alpha \models f = \omega(f)$, and α cannot be extended to satisfy LL_ω, but in this case $\alpha \not\leq_{lex} \omega(\alpha)$ and hence $\omega(\alpha) \leq_{lex} \alpha$ and again $(\omega(\alpha), \alpha) \in \preceq_{\mathcal{W}}$.

We also need to show that $\alpha \not\preceq_{\mathcal{W}} \omega(\alpha)$. Assume towards contradiction that $\alpha \preceq_{\mathcal{W}} \omega(\alpha)$. In that case, by the definition of $\preceq_{\mathcal{W}}$, there must be some $\omega' \in \mathcal{W}$ such that

- $\omega'(\omega(\alpha)) = \alpha$,
- $f^{\omega'(\omega(\alpha))} \leq f^{\omega(\alpha)}$, and
- $f^{\omega'(\omega(\alpha))} = f^{\omega(\alpha)} \Rightarrow \omega'(\omega(\alpha)) \leq_{lex} \omega(\alpha)$

Hence we find that

$$f^\alpha = f^{\omega'(\omega(\alpha))} \leq f^{\omega(\alpha)} \leq f^\alpha,$$

and $f^\alpha = f^{\omega(\alpha)}$. But using the last item, we then find that $\alpha \leq_{lex} \omega(\alpha)$, which contradicts our earlier assumption that α cannot be extended to satisfy BF_ω. □

3.3 Encoding the Symmetry Breaking Formula

What remains to be explained is how the pseudo-Boolean encoding of (3) is constructed. In order to explain this, we first recall that BreakID's encoding of LL_ω consists of the clauses (written here as pseudo-Boolean formulas)

$$y_0 \geq 1, \tag{4a}$$
$$\overline{y}_{j-1} + \overline{x}_{i_j} + \omega(x_{i_j}) \geq 1, \qquad 1 \leq j \leq n \tag{4b}$$
$$\overline{y}_j + y_{j-1} \geq 1, \qquad 1 \leq j < n \tag{4c}$$
$$\overline{y}_j + \overline{\omega(x_{i_j})} + x_{i_j} \geq 1, \qquad 1 \leq j < n \tag{4d}$$
$$y_j + \overline{y}_{j-1} + \overline{x}_{i_j} \geq 1, \text{ and} \qquad 1 \leq j < n \tag{4e}$$
$$y_j + \overline{y}_{j-1} + \omega(x_{i_j}) \geq 1, \qquad 1 \leq j < n \tag{4f}$$

where $\{x_{i_1}, \ldots, x_{i_n}\}$ is the *support* of ω (i.e., all variables x such that $\omega(x) \neq x$), ordered so that $i_j \leq i_k$ if and only if $j \leq k$. In this formula, each y_j is a fresh variable representing that up to x_{i_j}, α and $\omega(\alpha)$ are equal. Equation (4b) does the actual breaking.

What is important to note in this formula is that all the actual breaking is conditional on y_0 being true (as specified in (4a)). Indeed, if we would instead replace (4a) by $\overline{y}_0 \geq 1$, then (4c) would imply that all y-variables are false, and all constraints in this formula are trivially satisfied. In other words, if y_0 is false, adding constraints (4b–4f) has no impact on the models of F. Thus, we can encode (3) as

$$f \leq \omega(f), \tag{5a}$$
$$M \cdot \overline{y}_0 + f - \omega(f) \geq 0, \tag{5b}$$
$$M \cdot y_0 + \omega(f) - f \geq 1, \text{ and} \tag{5c}$$
$$\text{Equations (4b–4f)}, \tag{5d}$$

with M a sufficiently large integer. Here, constraints (5b) and (5c) encode that y_0 holds if and only if $f \geq \omega(f)$ (in combination with the first constraint, this means that f and $\omega(f)$ are in fact equal). To see this, note for instance that in case y_0 is false, (5b) is trivially satisfied. If y_0 is true on the other hand, (5b) expresses that $f \geq \omega(f)$. Similarly, (5c) is trivially satisfied when y_0 is true and otherwise expresses that $f \not\geq \omega(f)$.

4 Extending BreakID

In this section, we discuss some implementation details of our extension of BreakID (the symmetry breaker for SAT) into BreakIDPB (the symmetry breaker for pseudo-Boolean optimization problems), with support for breaking both weak and strong symmetries. In our implementation, we did not use arbitrary precision arithmetic; hence we only support integer coefficients of limited size. Such a restriction is not uncommon in the pseudo-Boolean world: in the

competitions, *SMALLINT* tracks (where the sum of all coefficients in a constraint does not surpass 2^{20}) are common.

First, we discuss how symmetries are detected in BreakIDPB, next we explain how BreakIDPB breaks weak and strong symmetries for pseudo-Boolean optimization functions and finally we discuss the compatibility of these extensions with the existing optimizations of BreakID.

4.1 Detecting (Weak) Symmetries

To detect symmetries, BreakIDPB first transforms the input problem into a graph that represents a simplified version of its syntax tree. This transformation guarantees a one-to-one correspondence between (syntactic) symmetries of the problem at hand and automorphisms (i.e. symmetries) of the graph. Next, an algorithm for searching graph automorphisms is employed [27]. This is a common strategy for symmetry detection [20, 3].

Remark 1. The graph automorphism problem is not known to be solvable in polynomial time nor is it NP-complete [29, 9]. The best currently accepted algorithm [4] runs in quasi-polynomial time and was introduced in 2016 (and corrected in 2017). For several specific classes of graphs, polynomial time algorithms are known [24, 25]. In practice, several graph isomorphism solvers (including the one that BreakID uses) exist that perform very well on inputs without particular combinatorial structure.

BreakID uses the following technique to generate a graph corresponding to a given pseudo-Boolean problem:

- The nodes are organized as follows:
  - for each literal ℓ_i the graph has a node with colour "1". These nodes represent the literals.
  - For each term $w_i\ell_i$ that occurs in constraint $\sum_i w_i\ell_i \geq n$, there is a node with colour "w_i". These nodes represent the terms. For the special case where $w_i = 1$, no new node is created since the literal equals the term in this case.
  - For each constraint $\sum_i w_i\ell_i \geq n$, there is a node with colour "n" distinct from all term-colours, uniquely determined by the degree of the constraint. These nodes represent the constraints.
- The edges are organized as follows:
  - each literal is connected with its negation (i.e., there is an edge $(x, \overline{x})$).
  - Each term $w_i\ell_i$ is connected to ℓ_i.
  - Each constraint node $\sum_i w_i\ell_i \geq n$ is connected to all terms $w_i\ell_i$ (which happens to be the node for ℓ_i in case $w_i = 1$).

If the graph needs to contain the objective function f as well it is extended as follows:

- An single extra node with a unique colour is added. This node represents the objective. Moreover, for each term $w_i\ell_i$ that occurs in the objective $f = \sum_i w_i\ell_i$, there is a node with colour "w_i".

$$F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$
$$f = 2x_2 + 3\overline{x}_3$$
$$C_1 = x_1 + x_2 + \overline{x}_3 \geq 1$$
$$C_2 = x_3 + \overline{x}_1 + \overline{x}_2 \geq 1$$
$$C_3 = 2x_1 + x_2 + x_3 \geq 2$$
$$C_4 = x_1 + 2x_2 + x_3 \geq 2$$

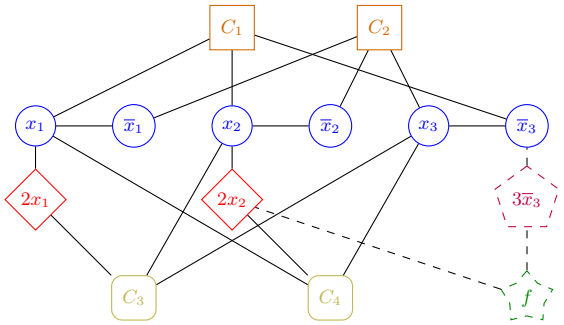


Fig. 1: Example of the graph constructed for a pseudo-Boolean optimization problem (F, f). The different vertex shapes represent different colours. The dashed edges and nodes are only required for detecting strong symmetries; in the case of weak symmetry detection, they are simply omitted.

– The objective node is connected to all terms occurring in it. As before, each term $w_i\ell_i$ is connected to ℓ_i.

In case we wish to detect *weak* symmetries instead of strong symmetries, the only modification needed is to remove the nodes that are only created for representing the objective; in Fig. 1, this means: dropping all dashed nodes and edges. Automorphisms of the resulting graph then correspond directly to weak symmetries.

The graph we construct differs slightly, but in a non-fundamental way from the graph used by Shatter [1]. The main difference is that we do not need a special treatment for clauses compared to general PB constraints. Next, the terms of (pseudo-Boolean) constraints are added differently.

Example 3. Fig. 1 contains an example of a pseudo-Boolean optimization problem and its corresponding graph. The optimization problem at hand has no strong symmetries, but exhibits the weak symmetry $(x_1, x_2)(\overline{x}_1, \overline{x}_2)$ (in disjoint cycle notation).

4.2 Breaking Weak Symmetries

To break symmetries, BreakIDPB generates the formulas described in Equations (5a–5d). For strong symmetries, the standard symmetry breaking constraints of Equations (4a–4f) are used instead.

Next to this, our implementation also supports an alternative encoding of the dominance breaking formula. The difference is that instead of using clauses, we use general pseudo-Boolean constraints. As a consequence, fewer introduced variables and fewer constraints are needed, at the cost of using larger (exponentially growing) exponents. Preliminary experiments showed, however, that for the current solvers, it does not make a big difference which encoding is used.

Example 4 (Example 3, continued). The symmetry breaking formulas added for the weak symmetry $(x_1, x_2)(\overline{x_1}, \overline{x_2})$ are the following:

$$2x_1 - 2x_2 \geq 0, \qquad \text{application of (5a)} \qquad (6a)$$
$$-2x_1 + 2x_2 - 2y_0 \geq -2, \qquad \text{application of (5b)} \qquad (6b)$$
$$2x_1 - 2x_2 + 3y_0 \geq 1, \qquad \text{application of (5c)} \qquad (6c)$$
$$1x_1 - 1x_2 - 1y_0 \geq -1. \qquad \text{application of (4b)–(4f)} \qquad (6d)$$

Constraints (6b) and (6c) encode that y_0 holds if and only if $2x_2+3\overline{x}_3 \geq 2x_1+3\overline{x}_3$ (in short, if $f \geq \omega(f)$). Together with (6a) this means that f and $\omega(f)$ are in fact equal. If y_0 is true, (6b) states that $f \geq \omega(f)$ and (6c) is trivially satisfied. Similarly if y_0 is false, (6b) is trivially satisfied. In this case (6c) expresses that $f \not\geq \omega(f)$. The last constraint (6d) performs the actual symmetry breaking.

4.3 Compatibility with previous optimizations

Compared to Shatter, BreakID introduced three optimizations for static symmetry breaking for SAT problems. The three introduced optimizations are a compact encoding of the lex-leader constraints, the exploitation of row interchangeability, and the generation of binary symmetry breaking clauses.

The first optimization is a *compact encoding of lex-leader constraints*. A more compact encoding of lex-leader constraints reduces the overhead introduced by adding the constraints to the problem. This more compact encoding is exactly the encoding used for LL_ω in the dominance breaking formulas generated by BreakIDPB as described in Equation 4.

The second optimization is the *exploitation of row interchangeability*. This is a type of symmetry present when a subset of variables can be structured as a two-dimensional matrix where each permutation of the rows induces a symmetry. If such a structure of symmetries can be found, the group of symmetries that forms this structure can be broken as a whole. This can also be applied to pseudo-Boolean optimization problems since the implementation detects this kind of symmetries based on symmetries detected in the generated graph. The detected row-symmetries are then broken using the implemented symmetry breaking constraints. For problems where there are weak symmetries present in the structure, dominance breaking formulas are used instead of symmetry breaking constraints to break those symmetries.

Lastly, it is known that posting lex-leader constraints for all detected symmetries in a symmetry group can be infeasible. Hence, often lex-leader constraints are only added for the generators of the symmetry group. BreakID introduced an alternative; the generation of *binary symmetry breaking clauses*. These are short symmetry breaking clauses, equivalent to posting symmetry breaking constraints in the compact encoding given in Equations 4b–4f with $j = 1$. These very short clauses are then posted for a large set of symmetries in the symmetry group. Since we know that the effect of the first constraints added is higher, this is a cost-effective way to break symmetries. For pseudo-Boolean optimization

Table 1: This table contains for each benchmark family the:

- total number of instances (#Inst.),
- number of instances that exhibit strong ($\#\mathcal{S}$) or weak ($\#\mathcal{W}$) symmetries,
- number of instances solved without symmetry breaking (#Solved plain),
- number of extra instances solved (+) or no longer solved (−) by enabling certain symmetry breaking techniques:
  - "Effect $\mathcal{S}$" breaks strong symmetries (compared to no breaking)
  - "Effect $\mathcal{W}$" breaks weak symmetries (compared to "Effect $\mathcal{S}$")
  - "Effect Opt. X" adds BreakID's optimizations to configuration X

				#Solved	Effect $\mathcal{S}$		Effect Opt. $\mathcal{S}$		Effect $\mathcal{W}$		Effect Opt. $\mathcal{W}$	
	#Inst.	$\#\mathcal{S}$	$\#\mathcal{W}$	plain	+	-	+	-	+	-	+	-
RoundingSAT (without LP integration) with core-guided optimization												
Knapsack	783	660	361	329	**7**	6	13	**24**	**10**	6	13	**14**
MiplibOpt	291	156	66	77	2	**3**	2	**4**	0	0	**3**	2
PbCompOpt	1600	982	574	967	14	**15**	11	**40**	10	**14**	12	**23**
Crafted	1514	1349	216	1165	30	**208**	48	**174**	11	**37**	47	**62**
RoundingSAT (without LP integration) with linear SAT–UNSAT search												
Knapsack	783	660	361	336	**13**	3	8	**20**	5	**6**	13	**15**
MiplibOpt	291	171	76	75	**2**	**2**	2	**4**	**1**	0	3	**4**
PbCompOpt	1600	982	574	870	**19**	16	8	**36**	5	**12**	9	**27**
Crafted	1514	1349	216	344	**105**	59	**153**	48	12	**31**	**507**	44
RoundingSAT (with LP integration) with with core-guided optimization												
Knapsack	783	660	361	710	12	**33**	16	**54**	7	**66**	22	**31**
MiplibOpt	291	168	75	96	**5**	2	3	**8**	0	**2**	4	**5**
PbCompOpt	1600	982	570	958	**38**	15	16	**64**	9	**21**	12	**45**
Crafted	1514	1343	216	849	80	**332**	**100**	67	20	**54**	**359**	36

problems, this option is available as well. The decision was made however to only generate binary breaking clauses for strong symmetries of pseudo-Boolean problems. Since generating binary breaking clauses is done to prevent the overhead of posting many long clauses, adding the longer dominance breaking constraints for each weak symmetry used defeats the purpose.

5 Experiments

We implemented our techniques on top of BreakID bundled with Saucy [27] for detection of graph automorphisms; our implementation is available online at `https://bitbucket.org/krr/breakid/branch/pb_optimization` (commit `46cc058` was used for the experiments). As a back-end solver we used RoundingSAT [21] (commit `b5de84db`). The resources available to solve and break instances were 16GB of memory and 2600s on an Intel(R) Xeon(R) Gold 6148 CPU running CentOS 7 with Linux kernel 3.10.

We tested our tool with three configurations of RoundingSAT as a back-end:

- two configurations compiled without LP integration:
  - the default configuration, which uses core-guided optimization [19]
  - and a configuration that uses linear SAT–UNSAT search.
- One configuration with LP integration using linear SAT–UNSAT search.

As benchmarks, we used all collections used in the work that introduced this core-guided optimization in RoundingSAT:

- **PbCompOpt**: The linear fixed–precision pseudo-boolean optimization problems of the latest *pseudo-Boolean competition* [35]. (1600 instances)
- **Knapsack**: A set of knapsack instances from the paper *Where Are the Hard Knapsack Problems?* [34] converted to PB. (783 instances)
- **MiplibOpt**: A collection of 0–1 ILP optimization instances from the benchmark sets MIPLIB 2, 3, 2003, 2010 and 2017 to PB. Some instances were rescaled to make them suitable for fixed–precision solvers. [12] (291 instances)
- **Crafted**: A set of crafted combinatorial benchmarks inspired by proof complexity. [33] (1514 instances)

Cactus plots, showing how many instances could be solved up to optimality within a certain time, for all BreakIDPB configurations and all benchmark sets can be found in Figures 2, 3 and 4. Table 1 contains for each benchmark set the number of instances, the number of those that exhibit strong and/or weak symmetries, as well as for different configurations how many instances can be solved by modifying one dimension (e.g., including weak symmetries) that could not be solved before (and dually, how many can no longer be solved now).

When analyzing the results using *core-guided search without LP integration* as a back-end (Fig. 2), the results are discouraging. The effect of adding symmetry breaking is only very small, and on the **Crafted** benchmark set, symmetry breaking even deteriorates performance. The same holds for adding weak symmetries and/or the optimizations BreakID implements compared to Shatter: the effect on most benchmark sets is small, while on **Crafted** we only observe negative effects.

On the other hand, when using *linear SAT–UNSAT without LP integration* search as a back-end (Fig. 3), we notice the opposite effect: the impact of adding symmetry breaking techniques is generally positive and on the **Crafted** benchmark set, BreakID's optimizations and *weak* symmetry breaking reinforce one another.

Interestingly, and seemingly contradictory, on this **Crafted** benchmark set, adding weak symmetry breaking also has a positive effect on many instances that do not exhibit weak symmetries. The explanation of why this is possible relates to the *graph* used for detecting the symmetries. When performing weak symmetry breaking, this graph contains no node for the objective function and is generally simpler and smaller (see Fig. 1 for an example). Upon manual inspection of several instances, we noticed that with this simpler graph, it is easier for BreakIDPB to find the underlying *structure* of the symmetry group (in particular, detecting row-interchangeability), and hence it makes it easier to break the symmetry group completely.

The same effect can be observed when using *linear SAT–UNSAT search with LP integration* as a back-end (Fig. 4). Symmetry breaking has some inpact on the number of solved instances and for the **Crafted** benchmarkt set we notice once again that the optimizations reinforce *weak* symmetry breaking.

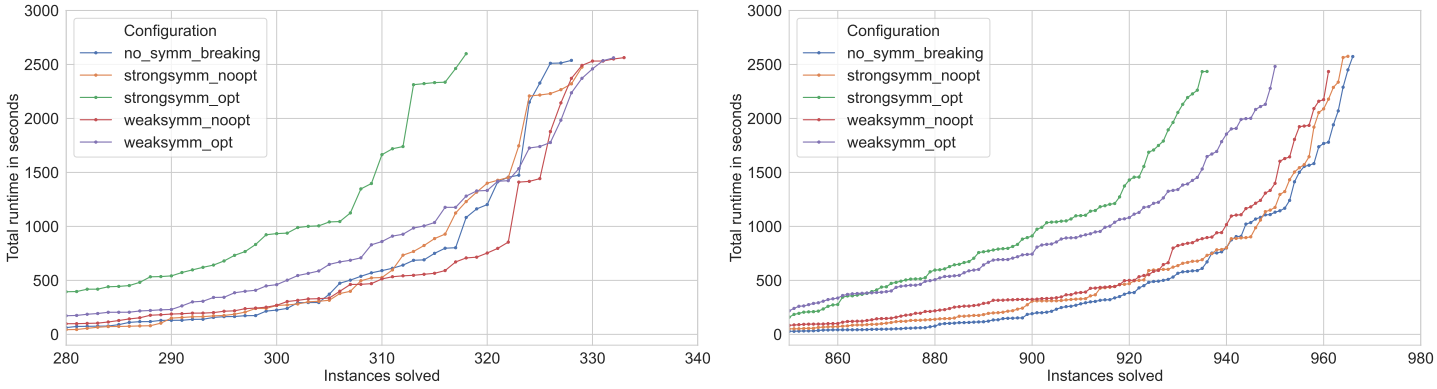
6 Conclusion

In this paper, we have studied static symmetry breaking for pseudo-Boolean optimization problems. We defined the novel notion of *weak symmetries*, which do not necessarily respect the objective function. We developed the theory showing how to statically break weak symmetries and implemented a tool that performs static symmetry breaking for both strong and weak symmetries. Our experimental validation shows that the effect of breaking (weak) symmetries depends greatly on the type of solving algorithm used, where we generally observed negative effects with core-guided optimization and positive effects with linear SAT–UNSAT search when not using LP integration. As a surprising side-effect of this investigation, we discovered that even for optimization problems where all symmetries are strong, doing the detection as if we are searching for weak symmetries can have a large positive impact on the breaking power. Developing a clear understanding of when precisely this is the case, and potentially, how to exploit this further, are challenges for future work. Another avenue for future work is extending our tool with support for *proof logging*, which the standard version of BreakID has recently obtained [8].

Acknowledgements We are grateful to Jakob Nordström for the interesting discussions on pseudo-Boolean search and optimization, as well as for providing details on the different configurations of RoundingSAT and where to find the benchmarks used in this paper.

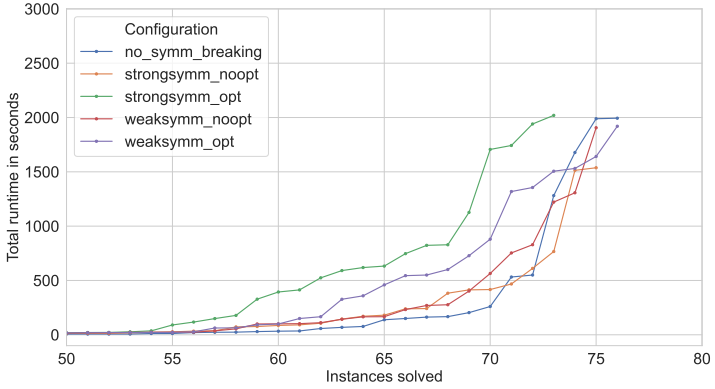
References

1. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Symmetry breaking for pseudo-boolean formulas. Journal of Experimental Algorithmics (JEA) **12**, 1–14 (2008)
2. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Shatterpb: symmetry-breaking for pseudo-boolean formulas. In: ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753). pp. 884–887 (2004). https://doi.org/10.1109/ASPDAC.2004.1337720
3. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for Boolean satisfiability. IEEE Transactions on Computers **55**(5), 549–558 (2006). https://doi.org/10.1109/TC.2006.75
4. Babai, L.: Graph isomorphism in quasipolynomial time. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing. pp. 684–697 (2016)
5. Benhamou, B., Nabhani, T., Ostrowski, R., Saïdi, M.R.: Dynamic symmetry breaking in the satisfiability problem. In: Proceedings of the 16$^{\text{th}}$ international conference on Logic for Programming, Artificial intelligence, and Reasoning. LPAR-16, Dakar, Senegal (April 25 - may 1, 2010)

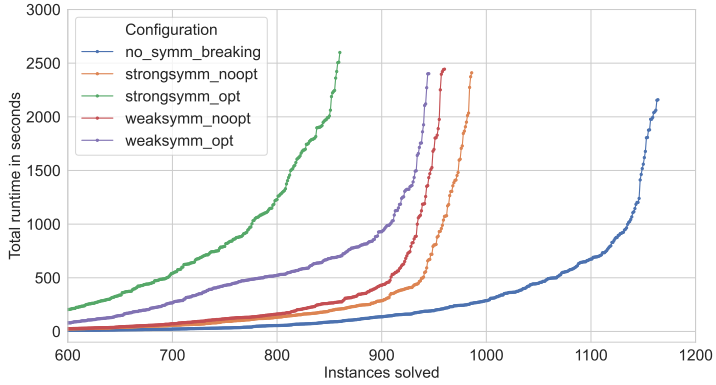


(a) The **Knapsack** benchmark set.

(b) The **PbCompOpt** benchmark set.

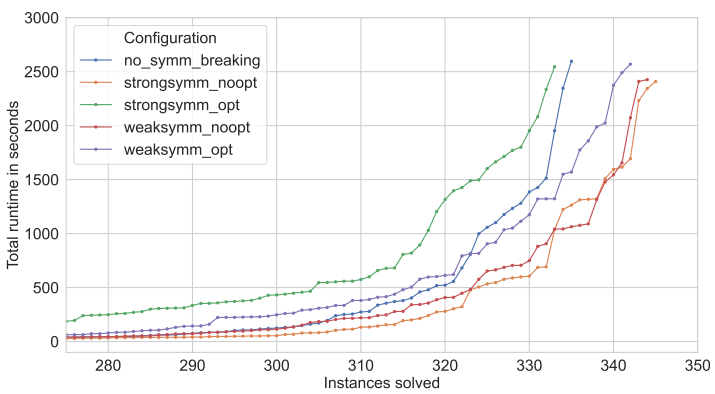


(c) The **MiplibOpt** benchmark set.

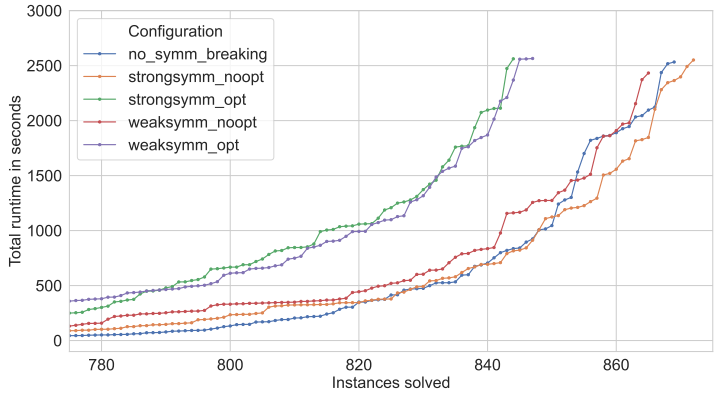


(d) The **Crafted** benchmark set.

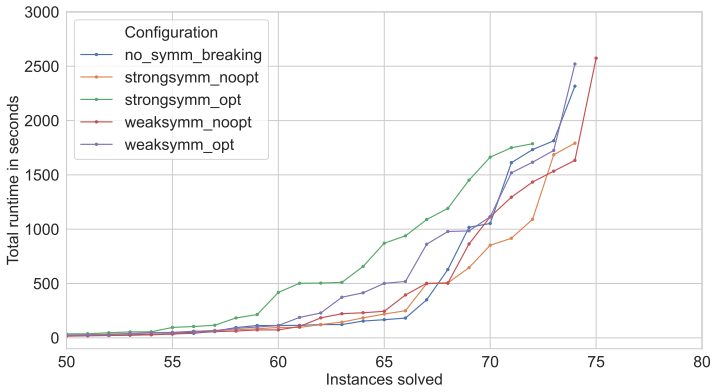
Fig. 2: Runtime (Y-axis) needed to solve a number (X-axis) of optimization problems to optimality for the different BreakIDPB configurations with the core-guided optimization configuration of RoundingSAT.



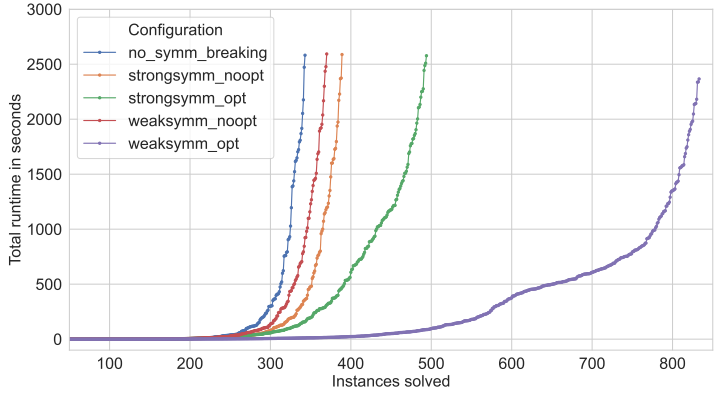
(a) The **Knapsack** benchmark set.



(b) The **PbCompOpt** benchmark set.

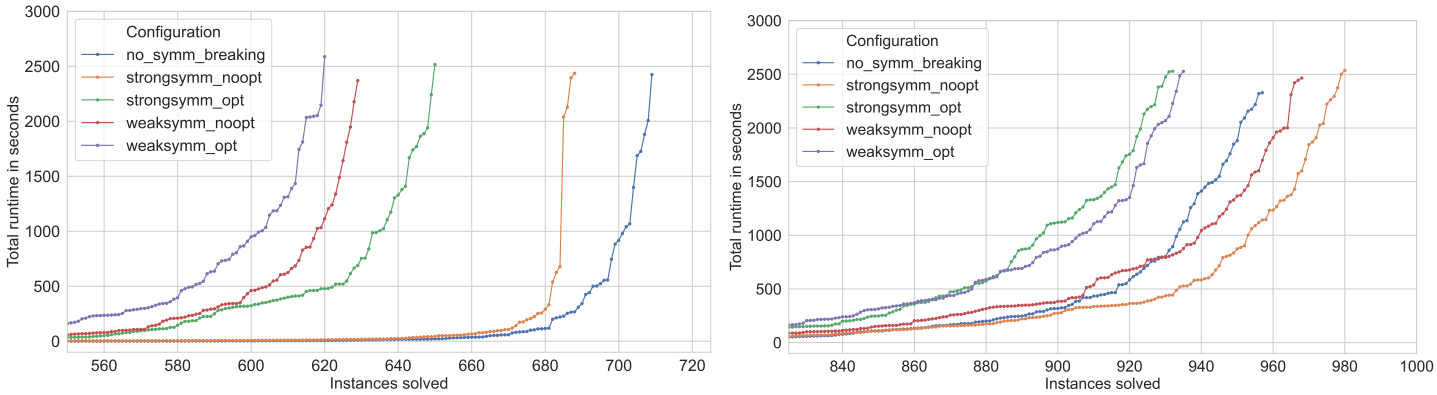


(c) The **MiplibOpt** benchmark set.



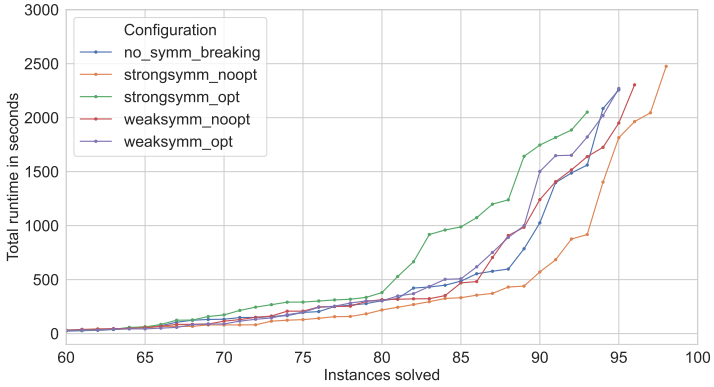
(d) The **Crafted** benchmark set.

Fig. 3: Runtime (Y-axis) needed to solve a number (X-axis) of optimization problems to optimality for the different BreakIDPB configurations with the linear SAT–UNSAT search configuration of RoundingSAT.

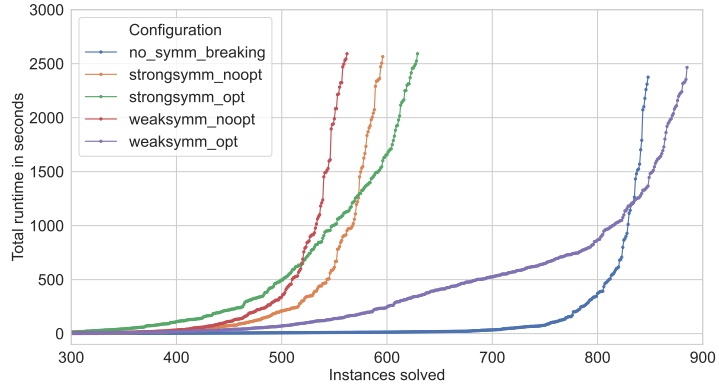


(a) The **Knapsack** benchmark set.

(b) The **PbCompOpt** benchmark set.



(c) The **MiplibOpt** benchmark set.



(d) The **Crafted** benchmark set.

Fig. 4: Runtime (Y-axis) needed to solve a number (X-axis) of optimization problems to optimality for the different BreakIDPB configurations with the linear SAT–UNSAT search configuration of RoundingSAT with linear solver integration.

6. Benhamou, B., Nabhani, T., Ostrowski, R., Saïdi, M.R.: Enhancing clause learning by symmetry in SAT solvers. In: Proceedings of the 2010 22Nd IEEE International Conference on Tools with Artificial Intelligence - Volume 01. pp. 329–335. ICTAI '10, IEEE Computer Society, Washington, DC, USA (2010). https://doi.org/10.1109/ICTAI.2010.55, http://dx.doi.org/10.1109/ICTAI.2010.55
7. Benhamou, B., Saïs, L.: Tractability through symmetries in propositional calculus. Journal of Automated Reasoning **12**(1), 89–102 (1994). https://doi.org/10.1007/BF00881844, http://dx.doi.org/10.1007/BF00881844
8. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified symmetry and dominance breaking for combinatorial optimisation. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022. pp. 3698–3707. AAAI Press (2022), https://ojs.aaai.org/index.php/AAAI/article/view/20283
9. Booth, K.S., Colbourn, C.J.: Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, Dept. of Computer Science, Univ. Waterloo (1979)
10. Chu, G., Stuckey, P.J.: A generic method for identifying and exploiting dominance relations. In: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming. pp. 6–22. CP'12, Springer-Verlag, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7˙4, http://dx.doi.org/10.1007/978-3-642-33558-7_4
11. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Principles of Knowledge Representation and Reasoning. pp. 148–159. Morgan Kaufmann (1996)
12. Devriendt, J.: Miplib 0-1 instances in opb format (May 2020). https://doi.org/10.5281/zenodo.3870965, https://doi.org/10.5281/zenodo.3870965
13. Devriendt, J., Bogaerts, B.: BreakID: Static symmetry breaking for ASP (system description). In: Bogaerts, B., Harrison, A. (eds.) Ninth workshop on Answer Set Programmin and Other Computing Paradigms: Proceedings, 2016, New York City, New York, USA, October 16, 2016. pp. 25–39 (2016), https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxhc3BvY3AyMDE2fGd4OjU4NTA0YTc3N2VkYWYyMWM
14. Devriendt, J., Bogaerts, B., Bruynooghe, M.: BreakIDGlucose: On the importance of row symmetry in SAT. In: Proceedings of the Fourth International Workshop on the Cross-Fertilization Between CSP and SAT (CSPSAT) (2014), https://lirias.kuleuven.be/handle/123456789/456639
15. Devriendt, J., Bogaerts, B., Bruynooghe, M.: Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 83–100. Springer (2017). https://doi.org/10.1007/978-3-319-66263-3˙6, https://doi.org/10.1007/978-3-319-66263-3_6
16. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bor-

deaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 104–122. Springer (2016). https://doi.org/10.1007/978-3-319-40970-2"˙8, http://dx.doi.org/10.1007/978-3-319-40970-2_8
17. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: On local domain symmetry for model expansion. Theory and Practice of Logic Programming **16**(5-6), 636–652 (009 2016). https://doi.org/10.1017/S1471068416000508, https://www.cambridge.org/core/article/on-local-domain-symmetry-for-model-expansion/96E8AB07EB4C02D502B68687B23AC21C
18. Devriendt, J., Bogaerts, B., De Cat, B., Denecker, M., Mears, C.: Symmetry propagation: Improved dynamic symmetry breaking in SAT. In: IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012. pp. 49–56. IEEE Computer Society (2012). https://doi.org/10.1109/ICTAI.2012.16, http://dx.doi.org/10.1109/ICTAI.2012.16
19. Devriendt, J., Gocht, S., Demirović, E., Nordström, J., Stuckey, P.J.: Cutting to the core of pseudo-boolean optimization: combining core-guided search with cutting planes reasoning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 3750–3758 (2021)
20. Drescher, C., Tifrea, O., Walsh, T.: Symmetry-breaking answer set solving. AI Communications **24**(2), 177–194 (2011)
21. Elffers, J., Nordström, J.: Divide and conquer: Towards faster pseudo-Boolean solving. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 1291–1299. ijcai.org (2018). https://doi.org/10.24963/ijcai.2018/180, https://doi.org/10.24963/ijcai.2018/180
22. Flener, P., Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models. In: Hentenryck, P. (ed.) Principles and Practice of Constraint Programming - CP 2002, LNCS, vol. 2470, pp. 462–477. Springer Berlin Heidelberg (2002). https://doi.org/10.1007/3-540-46135-3˙31, http://dx.doi.org/10.1007/3-540-46135-3_31
23. Gent, I.P., Smith, B.M.: Symmetry breaking in constraint programming. In: Proceedings of ECAI-2000. pp. 599–603. IOS Press (2000)
24. Grohe, M.: Fixed-point definability and polynomial time on graphs with excluded minors. In: 2010 25th Annual IEEE Symposium on Logic in Computer Science. pp. 179–188 (2010). https://doi.org/10.1109/LICS.2010.22
25. Grohe, M.: Logical and structural approaches to the graph isomorphism problem. In: Chatterjee, K., Sgall, J. (eds.) Mathematical Foundations of Computer Science 2013. pp. 42–42. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
26. Heule, M., Keur, A., Maaren, H.V., Stevens, C., Voortman, M.: CNF symmetry breaking options in conflict driven SAT solving (2005)
27. Katebi, H., Sakallah, K.A., Markov, I.L.: Symmetry and satisfiability: An update. In: Strichman, O., Szeider, S. (eds.) SAT. LNCS, vol. 6175, pp. 113–127. Springer (2010)
28. Kirchweger, M., Szeider, S.: SAT modulo symmetries for graph generation. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021. LIPIcs, vol. 210, pp. 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CP.2021.34, https://doi.org/10.4230/LIPIcs.CP.2021.34

29. Lubiw, A.: Some NP-complete problems similar to graph isomorphism. SIAM Journal on Computing **10**(1), 11–21 (1981). https://doi.org/https://doi.org/10.1137/0210002, publisher: SIAM
30. Lynce, I., Silva, J.P.M.: Breaking symmetries in SAT matrix models. In: Marques-Silva, J., Sakallah, K.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4501, pp. 22–27. Springer (2007). https://doi.org/10.1007/978-3-540-72788-0˙6, http://dx.doi.org/10.1007/978-3-540-72788-0_6
31. Mears, C., García de la Banda, M., Demoen, B., Wallace, M.: Lightweight dynamic symmetry breaking. Constraints pp. 1–48 (2013). https://doi.org/10.1007/s10601-013-9154-2, http://dx.doi.org/10.1007/s10601-013-9154-2
32. Metin, H., Baarir, S., Colange, M., Kordon, F.: Cdclsym: Introducing effective symmetry breaking in SAT solving. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10805, pp. 99–114. Springer (2018). https://doi.org/10.1007/978-3-319-89960-2“˙6, https://doi.org/10.1007/978-3-319-89960-2_6
33. Nordström, J.: Supplementary material for cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning (2022), https://www.csc.kth.se/~jakobn/publications/CoreGuidedPB/
34. Pisinger, D.: Where are the hard knapsack problems? Computers & Operations Research **32**(9), 2271–2284 (2005). https://doi.org/https://doi.org/10.1016/j.cor.2004.03.002, https://www.sciencedirect.com/science/article/pii/S030505480400036X
35. Roussel, O.: Pseudo-boolean competition 2016 (2016), http://www.cril.univ-artois.fr/PB16/
36. Sabharwal, A.: SymChaff: Exploiting symmetry in a structure-aware satisfiability solver. Constraints **14**(4), 478–505 (2009). https://doi.org/10.1007/s10601-008-9060-1, http://dx.doi.org/10.1007/s10601-008-9060-1
37. Schaafsma, B., Heule, M., van Maaren, H.: Dynamic symmetry breaking by simulating Zykov contraction. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5584, pp. 223–236. Springer (2009). https://doi.org/10.1007/978-3-642-02777-2“˙22, http://dx.doi.org/10.1007/978-3-642-02777-2_22
38. Van Caudenberg, D.: Static symmetry and dominance handling for pseudo-Boolean optimization (2022), https://www.bartbogaerts.eu/MScBScStudents/2022-Daimy/BScThesisDaimy.pdf, bachelor thesis; Bogaerts, Bart (supervisor)
39. Van Caudenberg, D., Bogaerts, B.: Static symmetry and dominance breaking for pseudo-Boolean optimization. In: BNAIC/BeNeLearn 2022 (2022)
40. Walsh, T.: Symmetry breaking constraints: Recent results. CoRR **abs/1204.3348** (2012)