Static 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 \[8, 19, 20, 32, 22, 23, 11, 13, 17, 10, 14\].

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 \[4\]. 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 \[29\], or by lazily posting the symmetry breaking constraints that static techniques would also add \[25\]. Other methods only eliminate unsatisfiable regions of the search space \[6\]; 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 \[21, 30, 5, 24, 15, 12\].

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 \(\sigma\) of the entire optimization problem: permutations (of literals) \(\sigma\) 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 \[7\] 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 as their symmetric counterpart, and
- in case they are equally good, lexicographically smaller than (or equal to) their symmetric counterpart.

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 \[13\]. The resulting tool (which we call BreakIDPB) differs from Shatter in the following ways:

- are at least as good as their symmetric counterpart, and
- in case they are equally good, lexicographically smaller than (or equal to) their symmetric counterpart.
– first and foremost, it can handle weak symmetries,
– it integrates all the improvements of BreakID [13], 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 observe a symmetry between the optimizations of BreakID (compared to Shatter) and the idea of breaking weak symmetries, where the gains (or losses) obtained by combining these two ideas are significantly stronger than the individual effects of only using one technique.

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.

2 Preliminaries

In this section, we recall some standard definitions related to pseudo-Boolean optimization. A literal \( \ell \) 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

\[
\sum_i a_i \ell_i \geq A,
\]

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 \( \ell_i \) 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 \( \alpha \) 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 \( \alpha \) is defined and being constant where \( \alpha \) is not defined. I.e., if \( \alpha \) 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 \( \alpha \) (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

\(^1\) Decision problems are a special case where \( F = 0 \).
objective $\sum_i w_i \ell_i$. We write $f^\alpha$ for the value of $f$ in $\alpha$, i.e., for $\sum_i w_i \alpha(\ell_i)$. An assignment $\alpha$ is an \textit{(optimal) solution} to $(F, f)$ if $\alpha \models F$ and $f^\alpha \leq f^\beta$ for each $\beta$ that satisfies $F$.

Let $\pi$ 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 $\pi$ to various semantic objects in the expected way:

- \textbf{pseudo-Boolean Constraints:} $\pi(\sum_i a_i \ell_i \geq A) = \sum_i a_i \pi(\ell_i) \geq A$,
- \textbf{to pseudo-Boolean formulas:} $\pi(C_1 \land \ldots \land C_n) = \pi(C_1) \land \ldots \land \pi(C_n)$,
- \textbf{to linear terms:} $\pi(f)(\sum_i w_i \ell_i) = \sum_i w_i \pi(\ell_i)$, and
- \textbf{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 $\pi$ a \textit{(syntactic) symmetry}\footnote{More general definitions of symmetry exist, but all practical use cases of symmetry detection and elimination use syntactic symmetries.} of $F$ if $\pi(F) = F$ and a \textit{strong symmetry} of $(F, f)$ if additionally $\pi(f) = f$. In that case, $\alpha$ 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 $\omega$ of $F$ is called a weak symmetry of the optimization problem $(F, f)$.

The strategy for breaking \textit{strong} symmetries, as employed, e.g., by Aloul et al. [1] is to add \textit{lex-leader} constraints. That is, given an order on the variables, for each detected\footnote{In practice, this is typically a set of generators of the group of strong symmetries.} strong symmetry $\sigma$, they add a set $LL_\sigma$ of pseudo-Boolean constraints such that for each total assignment $\alpha$, $\alpha$ can be extended to a model of $LL_\sigma$ if and only if $\alpha$ is lexicographically smaller than (or equal to) $\sigma(\alpha)$. When working with \textit{weak symmetries}, this strategy is no longer applicable. Indeed, it might be the case that $\sigma(\alpha)$ is lexicographically smaller than $\alpha$, but also has a worse objective than $\alpha$. In that case, we do not want to derive constraints that exclude $\alpha$. In the rest of this section, we will show in detail how to handle weak symmetries. As a theoretic framework, we will use \textit{dominance relations} as studied in the field of constraint programming [7]. 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 \textsc{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

\[
- f^\alpha \leq f^\beta \\
- \text{if } \beta \models F, \text{ then also } \alpha \models F.
\]

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 $\beta \notin \alpha$.

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

\[
\{(\sigma(\alpha), \alpha) \mid \sigma \in \langle S \rangle \land \sigma(\alpha) \leq_{lex} \alpha\},
\]

where $\langle S \rangle$ denotes the group generated by $S$, 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 $\alpha$, which are assignments $\alpha'$ that agree with $\alpha$ wherever $\alpha$ 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 $\alpha$ that cannot be extended to a model of $D$, there exists some assignment $\beta$ with $\beta \prec \alpha$.

$D$ is called complete if additionally whenever $\beta \prec \alpha$, $\alpha$ 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, \((F, f)\) and \((F \cup D, f)\) have the same optimal objective value.

To see why this proposition holds, note that whenever \( \alpha \) 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, \( LL_\sigma \) is a set of lex-leader constraints, then \( \bigcup_{\sigma \in S} LL_\sigma \) is a sound dominance breaking formula for the dominance relation \( \preceq_S \) 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 \( W \) be a set of weak symmetries of \((F, f)\). The relation \( \preceq_W \) defined as

\[
\{ (\omega(\alpha), \alpha) \mid \omega \in \langle W \rangle, f^{\omega(\alpha)} \leq f^\alpha \land (f^{\omega(\alpha)} = f^\alpha \Rightarrow \omega(\alpha) \leq_{lex} \alpha) \}
\]

is a dominance relation for \((F, f)\).

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

Now we construct a dominance breaking formula for this dominance relation. We will again assume that for each weak symmetry \( \omega \), we have PB formula \( LL_\omega \) such that \( \alpha \) can be extended to a model of \( LL_\omega \) if and only if \( \alpha \) 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 $\alpha$ 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 $W$ be a set of weak symmetries of $(F, f)$. The PB formula $BF_\omega$ defined as

$$\begin{cases} f \leq \omega(f) \\ (f = \omega(f)) \Rightarrow LL_\omega \end{cases}$$

forms a sound dominance breaking formula for $\preceq_W$.

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 $\alpha$ cannot be extended to an assignment that satisfies $BF_\omega$. We claim that in that case $\omega(\alpha) \prec_W \alpha$. The fact that $\omega(\alpha) \preceq_W \alpha$ is immediate: since $\alpha$ cannot be extended to satisfy $BF_\omega$, there are two possible situations:

- either $\alpha \nmid f \leq \omega(f)$, but in this case clearly $(\omega(\alpha), \alpha) \in \preceq_\omega$,
- or $\alpha \models f = \omega(f)$, and $\alpha$ cannot be extended to satisfy $LL_\omega$, but in this case $\alpha \npreceq_{lex} \omega(\alpha)$ and hence $\omega(\alpha) \preceq_{lex} \alpha$ and again $(\omega(\alpha), \alpha) \in \preceq_\omega$.

We also need to show that $\alpha \not\preceq_W \omega(\alpha)$. Assume towards contradiction that $\alpha \preceq_W \omega(\alpha)$. In that case, by the definition of $\preceq_W$, there must be some $\omega' \in W$ 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 $\alpha \preceq_{lex} \omega(\alpha)$, which contradicts our earlier assumption that $\alpha$ cannot be extended to satisfy $BF_\omega$. \qed

### 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
8 D. Van Caudenberg, B. Bogaerts

of $LL_\omega$ consists of the clauses (written here as pseudo-Boolean formulas)

\begin{align*}
y_0 & \geq 1 \tag{4a} \\
y_{j-1} + x_{ij} + \omega(x_{ij}) & \geq 1 & 1 \leq j \leq n \tag{4b} \\
y_{j} + y_{j-1} & \geq 1 & 1 \leq j < n \tag{4c} \\
y_{j} + \omega(x_{ij}) + x_{ij} & \geq 1 & 1 \leq j < n \tag{4d} \\
y_{j} + y_{j-1} + \omega(x_{ij}) & \geq 1 & 1 \leq j < n \tag{4e} \\
y_{j} + y_{j-1} + \omega(x_{ij}) & \geq 1 & 1 \leq j < n \tag{4f}
\end{align*}

where $\{x_{i1}, \ldots, x_{in}\}$ is the support of $\omega$ (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 $x_{ij}, \alpha$ 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 $y_0 \geq 1$, 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

\begin{align*}
f & \leq \omega(f) \tag{5a} \\
M \cdot y_0 + f - \omega(f) & \geq 0 \tag{5b} \\
M \cdot y_0 + \omega(f) - f & \geq 1 \tag{5c} \\
\text{Equations (4b–4f)} \tag{5d}
\end{align*}

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
static symmetry and dominance breaking for pseudo-boolean optimization

\[ F = C_1 \land C_2 \land C_3 \land C_4 \]
\[ f = 2x_2 + 3x_3 \]
\[ C_1 = x_1 + x_2 + x_3 \geq 1 \]
\[ C_2 = x_3 + x_1 + x_2 \geq 1 \]
\[ C_3 = 2x_1 + x_2 + x_3 \geq 2 \]
\[ C_4 = x_1 + 2x_2 + x_3 \geq 2 \]

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.

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) strong symmetries of the problem at hand and automorphisms of the graph. Next, an algorithm for searching graph automorphisms is employed [22]. This is a common strategy for symmetry detection [17, 3]. Fig. 1 contains an example of a pseudo-Boolean optimization problem and its corresponding graph. We refer the reader to the Bachelor thesis of the first author [31] for details on the precise construction, and the subtle ways in which it differs from Shatter’s graph. In order to detect weak 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.

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.
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 $LL_\omega$ 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 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 [22] 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 [18] (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.
(a) The Knapsack benchmark set. (b) The PbCompOpt 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.

We tested our tool with two configurations of RoundingSAT (compiled without LP integration) as a back-end: the default configuration, which uses core-guided optimization [16] and a configuration that uses 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* [28]. (1600 instances)
- **Knapsack**: A set of knapsack instances from the paper *Where Are the Hard Knapsack Problems?* [27] 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. [9] (291 instances)
- **Crafted**: A set of crafted combinatorial benchmarks inspired by proof complexity. [26] (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 and 3. 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 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 search as a back-end (Fig. 3), we notice the opposite effect: the impact of adding symmetry breaking techniques (and in particular breaking weak symmetries) 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
Table 1: This table contains for each benchmark family the:

- total number of instances (#Inst.),
- number of instances that exhibit strong (#S) or weak (#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 S” breaks strong symmetries (compared to no breaking)
  - “Effect W” breaks weak symmetries (compared to “Effect S”)
  - “Effect Opt. X” adds BreakID’s optimizations to configuration X

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RoundingSAT with core-guided optimization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knapsack</td>
<td>783</td>
<td>660</td>
<td>361</td>
<td>329</td>
<td>7</td>
<td>6</td>
<td>13</td>
<td>24</td>
</tr>
<tr>
<td>MiplibOpt</td>
<td>291</td>
<td>156</td>
<td>66</td>
<td>77</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>PbCompOpt</td>
<td>1600</td>
<td>982</td>
<td>574</td>
<td>967</td>
<td>14</td>
<td>15</td>
<td>11</td>
<td>40</td>
</tr>
<tr>
<td>Crafted</td>
<td>1514</td>
<td>1349</td>
<td>216</td>
<td>1165</td>
<td>30</td>
<td>208</td>
<td>48</td>
<td>174</td>
</tr>
</tbody>
</table>

RoundingSAT with linear SAT–UNSAT search

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Knapsack</td>
<td>783</td>
<td>660</td>
<td>361</td>
<td>336</td>
<td>13</td>
<td>3</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>MiplibOpt</td>
<td>291</td>
<td>171</td>
<td>76</td>
<td>75</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>PbCompOpt</td>
<td>1600</td>
<td>982</td>
<td>574</td>
<td>870</td>
<td>19</td>
<td>16</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td>Crafted</td>
<td>1514</td>
<td>1349</td>
<td>216</td>
<td>344</td>
<td>105</td>
<td>59</td>
<td>153</td>
<td>48</td>
</tr>
</tbody>
</table>

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.

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. 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.
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 ROUNDSAT and where to find the benchmarks used in this paper.

References


