BreakIDGlucose
On the importance of row interchangeability in SAT

Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe

University of Leuven
{jo.devriendt,bart.bogaerts,maurice.bruynooghe}@cs.kuleuven.be

Abstract. Symmetry is a topic studied by both the Satisfiability (SAT) and the Constraint Programming (CP) community. However, few attempts at transferring results between both communities have been made. This paper makes the link by investigating how symmetries from Constraint Satisfaction Problems (CSPs) transfer to symmetries of their SAT encodings. We point out that important symmetry groups studied in CP correspond to piecewise row symmetries of their SAT encoding, which -using a CP result- can be broken efficiently. We then show that the standard static symmetry breaking techniques for SAT fail to break these symmetries efficiently, and investigate how this can be mitigated. We also identify the Piecewise Row Interchangeability Detection problem, and we design a first automatic row interchangeability detection scheme for SAT. We implemented hese ideas in the BreakIDGlucose SAT solver, which detects and completely breaks interchangeable row symmetry. BreakIDGlucose's first place in the hard combinatorial track of the 2013 SAT competition shows the potential of row symmetry exploitation.

1 Introduction

Symmetry is a well-explored topic in many fields concerned with combinatorial search and optimisation. For Constraint Programming (CP) and Satisfiability (SAT) solving, many different forms of symmetry have been identified, and many different kinds of symmetry exploitation techniques have been devised.

Typically, each symmetry exploitation technique consists of two phases: the *symmetry detection* step, to identify which symmetries are present in the problem, and the *symmetry breaking* step, in which symmetry is used to reduce the search space. Often, symmetry detection is done prior to solving [1, 2], but runtime symmetry detection approaches exist as well [3]. The same holds for the symmetry breaking step, where *static symmetry breaking* [4] adds constraints to the problem that cut away a set of symmetrical solution candidates prior to solving, while *dynamic symmetry breaking* [5, 6] avoids symmetrical parts of the search tree at runtime by adjusting the solving mechanism.

In this paper, our focus is identifying symmetries in SAT problems, and constructing strong static symmetry breaking constraints for these. A well-known result for symmetry in Constraint Satisfaction Problems (CSP) is that complete symmetry breaking constraints exist for CSPs with *interchangeable row* symmetry, which are CSPs whose variables form a matrix for which the order of the rows is irrelevant. To the best of our knowledge however, no symmetry breaking approaches exist for SAT problems that both detect interchangeable row symmetry and construct complete symmetry breaking constraints for it.

We formally show in this paper that value and variable interchangeable CSPs can be encoded as SAT problems with interchangeable row symmetry. We propose a first rudimentary attempt to detect and completely break interchangeable row symmetry present in a CNF theory. We implemented the resulting techniques in a symmetry breaking preprocessor for CNF theories, and coupled this with the Glucose SAT solver [7]. The resulting system, BreakIDGlucose, went on to obtain the gold medal of the SAT+UNSAT crafted track of the 2013 SAT competition [8], indicating that state-of-the-art SAT solvers can be improved significantly by detecting and breaking row interchangeability symmetry.

The rest of this paper is structured as follows. In Section 2, we introduce CP and SAT symmetry terminology, thereby identifying different forms of symmetry, and explaining how these reduce to a CSP's SAT encoding. Section 3 explains how row interchangeability can be detected and completely broken in SAT. Section 4 explains implementation details of BreakIDGlucose and discusses experimental results, while Section 5 concludes this paper.

2 Symmetry in CP and SAT

2.1 CP and SAT terminology

A CSP is a triple (V, D, C) where V denotes a set of variables, D denotes the domain of possible values for these variables, and C is a set of constraints. An *assignment* to a CSP $\Pi = (V, D, C)$ is a function $\alpha : V \to D$. We abstract the set of constraints C as a function $C : (V \to D) \to \mathbb{B}$, denoting which assignments satisfy the constraints. We use $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$ to denote the set of boolean values. An assignment α is a *solution* if it satisfies the constraints in C, i.e., if $C(\alpha) = \mathbf{t}$. We call the set of all assignments to a CSP its *assignment space* and the set of solutions its *solution space*.

As running example throughout this paper, we use the pigeonhole problem (PH), where given a set of pigeons and a set of holes, an assignment of pigeons to holes must be found such that at most one pigeon is assigned to each hole. It is provably hard for a SAT-solver to solve unsatisfiable PH instances in a straightforward encoding [9], but exploiting the symmetry present in PH can make this problem trivial to solve.

Example 1. PH can be modelled as a CSP $PH_{CSP} = (V, D, C)$, where V represents the set of pigeons and D the set of holes. The constraints C are satisfied iff at most one pigeon is assigned to each hole.

A *SAT-problem* is a special case of a CSP, where $D = \mathbb{B}$. To distinguish a SAT-problem from a CSP, we represent it as a tuple (W, T), where W is a set of Boolean variables, and T is a set of clauses, also known as a CNF theory, representing the constraints. A clause is a set of literals, while a literal l is a variable $w \in W$ or its negation $\neg w$. In line with our CSP terminology, a solution to a SAT-problem is an assignment α for which $T(\alpha) = \mathbf{t}$.

Example 2. A PH with a set of pigeons P and a set of holes H can also be modelled as a SAT-problem $PH_{SAT} = (W, T)$. W then consists of a Boolean variable w_{ph} for each $(p, h) \in P \times H$, while the CNF-theory T requires that exactly one w_{ph} is assigned $\mathbf{t}$ for each $p \in P$, and at most one w_{ph} is assigned $\mathbf{t}$ for each $h \in H$.

CSPs can be encoded as SAT problems. Several common encoding procedures are based on the fact that an assignment α to a CSP $\Pi = (V, D, C)$ is a subset of $V \times D$, and hence finding an α can be seen as finding a function $\alpha' : V \times D \to \mathbb{B}$. More formally:

Definition 1. *Let (V, D) be a pair with V a set of variables and D a set of values and let W be a set of Boolean variables. An* encoding *of (V, D) to W is a bijection $Enc : V \times D \to W$.*

Definition 2. *Let α be an assignment to a CSP $\Pi = (V, D, C)$ and Enc an encoding. Then the* encoded assignment *of α is the unique assigment α^{enc} such that $\alpha(v) = d \Leftrightarrow \alpha^{enc}(Enc(v, d)) = \mathbf{t}$.*

For symmetry purposes, we are not interested in how the constraints are encoded exactly, but only how variables, values, assignments and symmetries are encoded. As such, we refer to any encoding procedure that makes use of an encoding bijection as defined above as a *standard encoding*. Examples of standard encoding procedures are *direct encoding* [10] and *support encoding* [11].

Definition 3. *Let $\Pi = (V, D, C)$ be a CSP. The* standard encoding *of Π is the SAT-problem $\Pi^{enc} = (W, T)$ such that there exists an encoding Enc and*

- *$T(\alpha^{enc}) = C(\alpha)$ for each assignment α to Π*
- *$T(\beta) = \mathbf{f}$ if β is not the encoded assignment of some α*

We denote the variable $Enc(v, d)$ by w_{vd}. Assignments β to Π^{enc} that are not encoded assignments of some assignment α to Π are those β that do not correspond to a function $V \to D$.

Example 3. It is clear from our definitions in Examples 1 and 2 that PH_{SAT} is a standard encoding of PH_{CSP}, with $Enc(p, h) = w_{ph}$.

2.2 Symmetry

This section introduces some terminology and results about symmetry from the field of constraint programming. Firstly, we define a symmetry as a permutation on the assignment space which preserves satisfaction to the constraints[12]:

Definition 4. *A* symmetry *S for a CSP (V, D, C) is a permutation on the assignment space $S : (V \to D) \to (V \to D)$ such that $C(\alpha) = C(S(\alpha))$.*

It follows directly from the definition that symmetries of a CSP Π form algebraic groups under the composition relation. Given a set of symmetries Σ, a *symmetry group* G_Σ is the set of symmetries generated by Σ. A *symmetry class* SC is a set of assignments that is closed under the symmetries in G_Σ, i.e., for each $\alpha \in SC$ and $S \in G_\Sigma$, also $S(\alpha) \in SC$. Note that the symmetry classes of two different assignments are either disjoint or identical, hence, the symmetry classes of a symmetry group partition the assignment space. Since symmetries preserve satisfaction of the constraints, either all assignments in a symmetry class are solutions or none is a solution.

Example 4. In PH, it does not matter which holes are assigned to which pigeons. So, a permutation on the assignment space which swaps the holes to which two pigeons are assigned is a symmetry.

Note that by Definition 4, every permutation on the assignment space of an unsatisfiable CSP is a symmetry. This makes Definition 4 a very general notion of symmetry, and in practice, only particular types of symmetry are used to speed up search. We now identify some common types of symmetry, using [2] as reference. One such symmetry type is that of *variable symmetry*, which is induced by a permutation on the set of variables of a CSP.

Definition 5. *Let P be a permutation on the set of variables V. Then $S_P : \alpha \mapsto \alpha \circ P$ is the permutation* induced *by P on the assignment space. If S_P is a symmetry, it is called a* variable symmetry*.*

Similarly, a *value symmetry* is induced by a permutation on the set of values:

Definition 6. *Let Q be a permutation on the set of values D. Then $S_Q : \alpha \mapsto Q \circ \alpha$ is a permutation* induced *by Q on the assignment space. If S_Q is a symmetry, it is called a* value symmetry*.*

CSPs where all permutations on V induce variable symmetries are *variable interchangeable*, while CSPs where all permutations on D induce value symmetries are *value interchangeable* [2].

Example 5. In PH_{CSP}, any permutation of the pigeons induces a variable symmetry, while any permutation on the holes induces a value symmetry. So PH_{CSP} is both variable and value interchangeable.

Other interesting symmetries are those derived from matrix-structured variable sets:

Definition 7. *Given a CSP $\Pi = (V, D, C)$, a* variable matrix *of Π is a bijection $M : Ro \times Co \to V$, where Ro and Co are index sets. We typically denote $M(r, c)$ as v_{rc}. A* row *of a variable matrix is the set of variables $\{v_{rc_1}, \ldots, v_{rc_n}\}$ which share the same first index $r \in Ro$. A* column *of a variable matrix is the set of variables $\{v_{r_1c}, \ldots, v_{r_mc}\}$ which share the same second index $c \in Co$.*

Note that the bijection Enc used for directly encoding a CSP Π, is also a variable matrix of Π^{enc}. Even stronger: there is a bijection between variables of Π and rows of Enc and between values of Π and columns of Enc.

Example 6. The encoding $Enc : P \times H \rightarrow W : (p, h) \mapsto w_{ph}$ is a variable matrix for PH_{SAT}, which in turn is a standard encoding of PH_{CSP}. A row of Enc is the set of $w_{ph} \in W$ that share the same pigeon, while a column is the set of w_{ph} that share the same hole.

The symmetries occurring on matrix-structured variable sets are induced by row and column permutations:

Definition 8. *Given is a CSP $\Pi = (V, D, C)$ and a variable matrix M of Π. An M-*row permutation *P is a variable permutation such that $P(v_{rc}) = v_{P'(r)c}$ for some permutation P' on M's Ro indices. An M-*row symmetry *is a symmetry induced by an M-row permutation.*
*An M-*column permutation *Q is a variable permutation such that $Q(v_{rc}) = v_{rQ'(c)}$ for some permutation Q' on M's Co indices. An M-*column symmetry *is a symmetry induced by an M-column permutation.*

A CSP is *row interchangeable* for some variable matrix M if all M-row permutations induce a symmetry. A CSP is *column interchangeable* for some variable matrix M if all M-column permutations induce a symmetry [2]. With M^{tr} the transpose of M, M-column symmetries are M^{tr}-row symmetries, and vice versa.

Example 7. PH_{SAT} is both row interchangeable and column interchangeable for Enc.

2.3 Symmetry Breaking Constraints

As mentioned in the introduction, one way to exploit symmetry is by posting symmetry breaking constraints which reduce the set of symmetrical solutions of a CSP:

Definition 9. *A set of constraints B is symmetry breaking for a symmetry group G of a CSP Π if for each symmetry class SC of G there exists an assignment $\alpha \in SC$ such that $B(\alpha) = \mathbf{t}$, and if there exists an assignment α in some symmetry class SC of G such that $B(\alpha) = \mathbf{f}$.*

This definition guarantees that not all solutions are cut away. Nonetheless, we want symmetry breaking constraints to cut away as much of the search tree as possible. Ideally, we want symmetry breaking constraints to be *complete*:

Definition 10. *A set of symmetry breaking constraints B is* complete *for some symmetry group of a CSP Π if for each symmetry class SC, there exists exactly one assignment $\alpha \in SC$ such that $B(\alpha) = \mathbf{t}$.*

This notion of complete symmetry breaking constraints corresponds to the classic notion of *sound and complete* symmetry breaking constraints [4], but since unsound symmetry breaking constraints are self-defeating, we rather use the above terminology.

We recall three important symmetry breaking results obtained in CP:

Proposition 1. *[13] Let $\Pi = (V, D, C)$ be a CSP with interchangeable variables and corresponding symmetry group G_Σ. There exists a set of complete symmetry breaking constraints B for G_Σ that is polynomial in the size of $V \times D$.*

Proposition 2. *[13] Let $\Pi = (V, D, C)$ be a CSP with interchangeable values and corresponding symmetry group G_Σ. There exists a set of complete symmetry breaking constraints B for G_Σ that is polynomial in the size of $V \times D$.*

Proposition 3. *[14, 15] Let $\Pi = (V, D, C)$ be a CSP with interchangeable rows for some variable matrix M and corresponding symmetry group G_Σ. There exists a set of complete symmetry breaking constraints B for G_Σ that is polynomial in the size of $V \times D$.*

The constraints for Proposition 3 use a lexicographic order $\leq_{\text{lex}}$ on the rows of an assignment, and state that in each solution each two consecutive rows r_1 and r_2 are lexicographically increasing: $r_1 \leq_{\text{lex}} r_2$. This way, for every symmetry class of SC of G_Σ, only the assignment which has its rows ordered lexicographically can be a solution.

Note that Proposition 3 does not tell anything about the case where for some variable matrix both the rows are interchangeable and the columns are interchangeable. Currently, no polynomially sized symmetry breaking constraints are known to completely break the symmetry group arising from interchangeable rows and columns for the same variable matrix [15, 4].

2.4 Symmetry encodings

In this subsection we transfer symmetry properties from CSPs to their standard encodings in SAT. We first define what happens with variable and value permutations:

Definition 11. *Given is a CSP $\Pi = (V, D, C)$, Π's standard encoding $\Pi^{enc} = (W, T)$, a variable permutation P of Π and a value permutation Q of Π. The permutation $P^{enc} : W \rightarrow W : w_{vd} \mapsto w_{P(v)d}$ is the* encoded variable permutation *of P. The permutation $Q^{enc} : W \rightarrow W : w_{vd} \mapsto w_{vQ(d)}$ is the* encoded value permutation *of Q.*

Since there is a bijection between rows and columns of *Enc* and respectively variables and values of Π, an encoded variable permutation is an *Enc*-row permutation of Π^{enc} and an encoded value permutation is an *Enc*-column permutation of Π^{enc}.

Encoded variable permutations have the following useful property:

Lemma 1. *Let P be a variable permutation and α an assignment for some CSP, then the following equation holds:*

$$(\alpha \circ P)^{enc} = \alpha^{enc} \circ P^{enc}$$

Proof. First we write the lemma in a more verbose form:

$$\forall w_{vd} \in W : (\alpha \circ P)^{enc}(w_{vd}) = \mathbf{t} \Leftrightarrow (\alpha^{enc} \circ P^{enc})(w_{vd}) = \mathbf{t}$$

Using the definitions of standard encodings and encoded permutations, we can simply transform the left hand side to the right hand side of the equivalence:

$$\begin{aligned}(\alpha \circ P)^{enc}(w_{vd}) = \mathbf{t} &\Leftrightarrow (\alpha \circ P)(v) = d \\ &\Leftrightarrow \alpha(P(v)) = d \\ &\Leftrightarrow \alpha^{enc}(w_{P(v)d}) = \mathbf{t} \\ &\Leftrightarrow \alpha^{enc}(P^{enc}(w_{vd})) = \mathbf{t}\end{aligned}$$

□

A similar lemma with similar proof exists for value permutations:

Lemma 2. *Let Q be a value permutation and α an assignment for some CSP, then the following equation holds:*

$$(Q \circ \alpha)^{enc} = \alpha^{enc} \circ (Q^{-1})^{enc}$$

We can now show that each standard encoding of a CSP with variable symmetry exhibits again variable symmetry:

Proposition 4. *Given a CSP $\Pi = (V, D, C)$, Π's standard encoding $\Pi^{enc} = (W, T)$ and a variable permutation P, then P induces a symmetry for Π if and only if P^{enc} induces a symmetry for Π^{enc}.*

Proof. We know from Definitions 4 and 5 that P induces a symmetry for Π if and only if for every assignment α, $C(\alpha) = C(\alpha \circ P)$.
Similarly, P^{enc} induces a symmetry for Π^{enc} if and only if for every assignment β, $T(\beta) = T(\beta \circ P^{enc})$. Also, since P is a variable permutation, β represents a function iff $\beta \circ P^{enc}$ represents a function. So $T(\beta) = \mathbf{f} = T(\beta \circ P^{enc})$ for every β that does not represent a function. As a result, P^{enc} induces a symmetry if and only if for every encoded assignment α^{enc}, $T(\alpha^{enc}) = T(\alpha^{enc} \circ P^{enc})$.

We know from Definition 3 that $C(\alpha) = T(\alpha^{enc})$ and $C(\alpha \circ P) = T((\alpha \circ P)^{enc})$. Combining this with Lemma 1 results in:

$$\begin{aligned}C(\alpha) &= T(\alpha^{enc}) \\ C(\alpha \circ P) = T((\alpha \circ P)^{enc}) &= T(\alpha^{enc} \circ P^{enc})\end{aligned}$$

Which can only hold if $C(\alpha) = C(\alpha \circ P) \Leftrightarrow T(\alpha^{enc}) = T(\alpha^{enc} \circ P^{enc})$, meaning P induces a symmetry if and only if P^{enc} induces a symmetry. □

An analog result holds for value symmetries (proof omitted):

Proposition 5. *Given a CSP $\Pi = (V, D, C)$, Π's standard encoding $\Pi^{enc} = (W, T)$ and a value permutation Q, then Q induces a symmetry for Π if and only if Q^{enc} induces a symmetry for Π^{enc}.*

Proposition 4 and 5 show that the standard encoding of both a CSP with variable symmetry and a CSP with value symmetry, exhibit variable symmetry. As a result, since variable permutations encode to *Enc*-row permutations and value permutations encode to *Enc*-column permutations, the following corollary holds:

Corollary 1. *Let $\Pi = (V, D, C)$ be a CSP and $\Pi^{enc} = (W, T)$ its standard encoding. Π is variable interchangeable iff Π^{enc} is row interchangeable for Enc. Π is value interchangeable iff Π^{enc} is column interchangeable for Enc.*

Example 8. In Example 3 it was argued that PH_{SAT} is a standard encoding of PH_{CSP}. In Example 5 it was pointed out that PH_{CSP} is both variable and value interchangeable. Because of Corollary 1, it is no coincidence that we argued in Example 7 that PH_{SAT} is both row and column interchangeable for *Enc*.

Corollary 1 establishes a formal connection between variable and value interchangeability in a CSP Π, and respectively row and column interchangeability in a matrix encoding of Π. This connection is known in literature [16, 15], but so far no formal proof was presented.

2.5 Piecewise symmetry

It rarely happens that a CSP is interchangeable for all of its variables or all of its values at once. The notions of *piecewise variable interchangeability* and *piecewise value interchangeability* [2] remedy this.

Definition 12.

We define a piecewise row interchangeable or column interchangeable CSP in the same way:

Definition 13. *A CSP $\Pi = (V, D, C)$ is piecewise row interchangeable for some partition $\{V_1, \ldots, V_n\}$ and a variable matrix M_i for each V_i, if Π is row interchangeable for each M_i. Similarly, Π is piecewise column interchangeable for some partition $\{V_1, \ldots, V_m\}$ and a variable matrix M_j for each V_j, if Π is column interchangeable for each M_j.*

Note that we allow the number of rows and columns of different variable matrices $M_{\ldots}$ to be different. Corollary 1 then generalizes to the piecewise case:

Corollary 2. *Let $\Pi = (V, D, C)$ be a CSP and $\Pi^{enc} = (W, T)$ its standard encoding. If Π is piecewise variable interchangeable for some partition $\mathcal{V} = \{V_1, \ldots, V_n\}$ of V, then Π^{enc} is piecewise row interchangeable for $\mathcal{V}$ and the variable matrices $Enc_{Vi} : V_i \times D \to W_{Vi} : (v, d) \mapsto Enc(v, d)$ for V_i. If Π is piecewise value interchangeable for some partition $\mathcal{D} = \{D_1, \ldots, D_m\}$ of D, then Π^{enc} is piecewise column interchangeable for $\mathcal{D}$ and the variable matrices $Enc_{Dj} : V \times D_j \to W_{Dj} : (v, d) \mapsto Enc(v, d)$ for D_j. W_{Vi} and W_{Dj} denote the appropriate subsets of W that make Enc_{Vi} and Enc_{Dj} bijections.*

3 Breaking and detecting row interchangeability in SAT

In the previous section, we showed that SAT encodings of (piecewise) variable or value interchangeable CSPs are respectively (piecewise) row or column interchangeable. Since a CNF theory has no inherent rows or columns, we will ignore column interchangeability for the rest of this paper, as it is merely another way of looking at row interchangeability. Note that we also are not investigating symmetry groups arising from both row *and* column interchangeable variable matrices, which is a related but different topic.

Now, given that a row interchangeability symmetry group can be broken very efficiently, one should strive to exploit row interchangeability in SAT theories. Two obstacles currently block our path. The first is that it is unknown how one can efficiently infer from a CNF theory that a significant subset of variables can be structured as a row interchangeable variable matrix. This is contrasts with higher level CP languages such as MiniZinc [17], or relational languages such as FO$(\cdot)$ [18] and Alloy [19], where symmetry groups that reduce to row interchangeability in a lower level CNF theory are easily detectable [1]. Secondly, even if a set of symmetry inducing permutations swapping consecutive rows of a variable matrix is given, current SAT symmetry breaking mechanisms would use this knowledge in an incomplete way: they might post incomplete symmetry breaking constraints for the row interchangeability symmetry group.

We continue this section by summarizing how symmetry detection and static symmetry breaking are currently implemented in SAT. Next, we explain the shortcomings of this approach with respect to interchangeable rows. Finally, we propose a complete row interchangeability breaking approach for SAT, and give a first attempt at row interchangeability detection for SAT.

3.1 Breaking Row Symmetry in SAT

In SAT, symmetry is often detected by converting a SAT problem Π to a graph G with the property that automorphisms of G correspond to permutations of literals of Π that induce a symmetry of Π. Graph automorphism detection tools [20, 21] are used to detect generators of the graph automorphism group, which then correspond to generators of a symmetry inducing permutation group of Π. The detected symmetry can be broken statically, e.g., by Shatter [22], or dynamically, e.g., by Symmetry Propagation [23], two methods that are designed to break any symmetry induced by a permutation of literals. Many other approaches exist, of which we briefly mention SymChaff [24]. SymChaff expects so-called *k-complete multiclass symmetry* specifications to be given as part of its input, and can break these symmetries completely dynamically. We suspect that k-complete multiclass symmetry groups actually are row interchangeability symmetry groups, but we are not able to confirm this link yet.

The most convenient of these symmetry breaking approaches is the static symmetry breaking approach taken by Shatter, a preprocessor for SAT solvers that works on any CNF theory. It uses Saucy [20] for symmetry detection and extends the CNF theory with symmetry breaking constraints. Below, we explain

[1] The original context that led to this research was symmetry exploitation for the IDP system [18]. As pointed out in [14], when searching for a relation $\mathcal{R} : D_1 \times \ldots \times D_n$, it is possible to break the symmetry induced by the permutation group of a single disjoint domain D_i with a polynomially sized symmetry breaking constraint. This is consistent with our findings, where the reduction of domain independent relational constraints to SAT induces row interchangeability in the resulting CNF theory.

this approach in detail and argue that it fails to exploit row interchangeability.

Given a SAT problem $\Pi = (W, T)$, Saucy outputs a set of permutations of literals of Π such that each of these permutations induce a symmetry of Π. From now on, we denote this induced set of symmetries by Σ, which is a *minimal* set of generators of a symmetry group G_Σ of Π. For this section, we assume that these symmetries are induced by permutations of *variables* only, since these induce variable symmetries, which we are interested in . This is neither a very restrictive nor an essential assumption.

Following [25], Shatter uses a total order $<_\mathrm{W}$ on the Boolean variables in W to induce a total lexicographic order $<_\mathrm{lex}$ on the assignment space: $\alpha_1 <_\mathrm{lex} \alpha_2$ if for some w, $\alpha_1(w) = \mathbf{f}$ and $\alpha_2(w) = \mathbf{t}$ while for every $w' <_\mathrm{W} w$, $\alpha_1(w') = \alpha_2(w')$. Shatter then extends the CNF theory with a symmetry breaking constraint $\phi_\sigma \equiv \alpha \leq_\mathrm{lex} \sigma(\alpha)$ for each $\sigma \in \Sigma$. Together, these constraints cut away all assignments that are lexicographically larger than their image under the symmetries in Σ, but not necessarily under all symmetries in the group G_Σ.

This approach still has one free parameter, namely the ordering $<_\mathrm{W}$. Since each variable w typically has an integer identifier $id(w)$, Shatter pragmatically determines $w <_\mathrm{W} w'$ iff $id(w) < id(w')$. However, as we shall see in Example 9, the ordering is actually very important and choosing the correct ordering can make the difference between breaking the entire symmetry group and breaking almost nothing.

Example 9. Let $PH_{32} = (W, T)$ be a standard encoding of a pigeonhole instance with 3 pigeons and 2 holes, where W consists of variables $w_{ij}, i \in \{1, 2, 3\}, j \in \{1, 2\}$ and variable w_{ij} denotes whether pigeon i is assigned hole j. The mapping $M : (i, j) \mapsto w_{ij}$ is a variable matrix for PH_{32} where every row of M represents a pigeon, and every column a hole.

Now, we focus on the interchangeability of pigeons, or equivalently, on the M-row interchangeability. Suppose that Saucy detects that all pigeons are interchangeable, and returns a minimal set of generator permutations that induce the interchangeable pigeon symmetry group. For example, Saucy returns P_{12} and P_{13}, where P_{ij} is the permutation that swaps pigeon i with pigeon j, or equivalently, the permutation that swaps row i with row j in M.

Suppose the order on the variables $<_\mathrm{W}$ is defined as follows:

$$w_{11} <_\mathrm{W} w_{12} <_\mathrm{W} w_{21} <_\mathrm{W} w_{22} <_\mathrm{W} w_{31} <_\mathrm{W} w_{32}. \tag{1}$$

Shatter then posts constraints that intuitively read as "pigeon 1 resides in a hole smaller than or equal to the hole in which pigeon 2 resides" and "pigeon 1 resides in a hole smaller than or equal to the hole in which pigeon 3 resides". It is clear that these constraints do not break all symmetries, because for example, they do not relate pigeon 2 and pigeon 3. This is a suboptimal outcome, since we already mentioned in Proposition 3 that there exists a short symmetry breaking constraint that completely breaks a row interchangeability symmetry group. This constraint is quite simple: just state that all rows must be lexicographically ordered. In other words, the constraints "pigeon 1 resides in a hole smaller than or equal to the hole in which pigeon 2 resides" and "pigeon 2 resides in a hole smaller than or equal to the hole in which pigeon 3 resides" would break the M-row interchangeability symmetry group completely. This is simply due to the transitivity of the "smaller than or equal to" relation, which implies that also pigeon 1's hole is smaller than or equal to pigeon 3's hole.

Now we see the importance of the variable ordering: had Shatter used the variable ordering

$$w_{21} <_{\mathrm{W}} w_{22} <_{\mathrm{W}} w_{11} <_{\mathrm{W}} w_{12} <_{\mathrm{W}} w_{31} <_{\mathrm{W}} w_{32}, \tag{2}$$

its constraints would have completely broken the M-row interchangeability symmetry group.

These observations for the pigeon hole problem can be generalized:

Proposition 6. *Given a SAT problem $\Pi = (W, T)$ with variable matrix $M : Ro \times Co \to W$ such that Π is M-row interchangeable. Assume without loss of generality that $Ro = \{1, \ldots, n\}$ and $Co = \{1, \ldots m\}$. Let $<_{\mathrm{W}}$ be the total order defined by $w_{ij} <_{\mathrm{W}} w_{kl}$ if $i < k$ or if $i = k$ and $j < l$ and let $\sigma_{i,j}$ denote the symmetry induced by swapping row i and j. Then, the set of constraints*

$$\alpha \leq_{\mathrm{lex}} \sigma_{i,i+1}(\alpha) \text{ for } 0 < i < n$$

breaks the M-row interchangeability symmetry group completely.

Proof. Applying Shatter's equivalency preserving compression techniques for symmetry breaking constraints turns $\alpha \leq_{\mathrm{lex}} \sigma_{i,i+1}(\alpha)$ into a small constraint stating that the assignment to the variables of row i is lexicographically smaller than or equal to the assignment to the variables of row $i + 1$. Hence, Proposition 6 is in fact a reformulation of the constraints used to break row interchangeability completely in CSPs [14, 15].

Proposition 6 shows that if the order on the variables correctly follows the matrix structure *and* the set of symmetry inducing generators consists of the permutations that swap two consecutive rows, then the symmetry breaking constraints posted by Shatter are complete. Hence, a good way to improve Shatter is to ensure that the chosen variable ordering and the set of symmetry generators are compatible. As we explain later, this is the approach we took with BreakID.

3.2 Detecting Row Symmetry in SAT

We remark again that detecting a set of symmetry inducing permutations that swap two consecutive rows is not a trivial task, simply because a CNF theory has no builtin notion of rows or matrices. A good symmetry detection tool however should be able to extract the row interchangeable matrix structure of (a subset of) variables, if such structure is present. Currently, Saucy is not "row interchangeability aware", and hence getting the right symmetry inducing generators is not guaranteed; the only guarantee Saucy gives is that the set of generator permutations is minimal. For instance, in Example 9, Saucy could also have returned the set $\{P_{123}, P_{13}\}$, which still induces the row interchangeability symmetry group, but which reveals less of the matrix structure of PH_{32}. We give a first attempt to remedy this symmetry detection problem in this section.

Taking into account that we want to detect multiple row interchangeability symmetry groups, we formalise the symmetry detection problem as follows:

Problem 1 (Piecewise Row Interchangeability Detection for CNF (PRID)). Given a SAT problem $\Pi = (W, T)$, *PRID* consists of finding a partition $\{W_1, \ldots, W_n\}$ of W and a set $\{M_1, \ldots, M_n\}$ such that

- every M_i is a variable matrix of W_i,
- Π is M_i-row interchangeable for every i,
- the sum of the number of rows of each M_i is maximal.

The third condition ensures that the detected symmetry groups are significant, and avoids a trivial solution to the PRID problem by partitioning W into singleton sets.

To the best of our knowledge, the PRID problem has not been mentioned in literature, let alone solved. In the rest of this subsection, we design a first attempt at an algorithm for PRID. We give a rough sketch.

First, remark that in Example 9 we assumed Saucy returned a set Σ of generator permutations that already contained some structure: the generator permutations were row swaps of the known variable matrix. Permutations that swap rows carry a partial structure of their corresponding variable matrix in them, so it makes sense to investigate those closer. A first characterization of a row swapping permutation P is that P is an *involution*: $P = P^{-1}$. So if we have a set of symmetry inducing involutions, they might encode row swapping permutations on some unknown variable matrix. This leads to the following terminology:

Given an involution P, a *candidate row* of P is a sequence of variables $c = w_1, \ldots, w_n$, such that

- variable w does not occur in c if $P(w) = w$,
- if $P(w) \neq w$, then either w occurs in c or $P(w)$ occurs in c (but not both).

Given a candidate row $c = w_1, \ldots, w_n$ of P, its *symmetrical counterpart* is the candidate row $P(c) = P(w_1), \ldots, P(w_n)$. Since P is an involution, $P(P(c)) = c$, which partitions all candidate rows of P into pairs of symmetrical counterparts $\{c, P(c)\}$. Note that if P is a symmetry inducing involution for a problem Π, then each such pair of symmetrical counterparts actually represents a row-interchangeable variable matrix with two rows for Π.

Now, let Σ be a set of generator permutations detected by Saucy, and let $P_1, P_2 \in \Sigma$ be two symmetry inducing involutions. If P_1 and P_2 share a common candidate row c, but do not share any other variables, then $M = \{P_1(c), c, P_2(c)\}$ forms a set of three interchangeable rows. And if M contained a candidate row c_2 of some other involution $P_3 \in \Sigma$, but M does not contain any variables from the symmetrical counterpart $P_3(c_2)$, then M can be extended with $P_3(c_2)$ as fourth row.

These observations lead to the following algorithm sketch:

Initially, the algorithm searches for two involutions $P_1, P_2 \in \Sigma$ that share a candidate row and that do not share any other variables. These form a variable matrix M_i with three rows. Then the algorithm searches iteratively for more involutions which share a row with M_i, but no other variables. If one is found, M_i is extended with another row. If no more can be found, M_i is added to the set of detected variable matrices, and all permutations Q that share a variable with M are removed from Σ. Then the search restarts by looking for two new $P_1', P_2' \in \Sigma$ to form a new variable matrix.

By removing all permutations Q that share a variable with a variable matrix, we guarantee that the detected variable matrices share no variables, and hence the above algorithm outline finds a variable matrix partition as required by the PRID problem.

Recall that Saucy does not guarantee that any involutions are present in its generator set Σ. However, the symmetry inducing permutation group G_Σ generated by Σ is guaranteed to contain any symmetry inducing row swapping permutation. Hence, before we perform the above described row-interchangeable matrix extraction, we try to find more involutions contained in G_Σ. This is done by a heuristic enumeration scheme: given the generator set Σ outputted by Saucy, we take two elements $\sigma, \sigma' \in \Sigma$. If $\sigma \circ \sigma'$ or $\sigma' \circ \sigma$ permutes relatively few literals, we extend Σ with it. This process is repeated until timeout. Since row candidates typically are relatively small involutions, the focus on small compositions hopefully increases the number of involutions in Σ which represent row swaps in variable matrices.

To summarize: our answer to the PRID problem in CNF consists of two steps (**i**) a heuristic method to generate a part of the permutation group generated by the output of Saucy that hopefully contains many involutions, and (**ii**) a variable matrix extraction phase based on involutions swapping two rows of a matrix.

4 BreakIDGlucose

4.1 BreakID: a Row Interchangeability Breaking Preprocessor

We implemented the row interchangeability detection and breaking techniques presented in the previous section into BreakID [26], a symmetry breaking preprocessor for SAT. Given a SAT problem $\Pi = (W, T)$, BreakID implements the following steps:

1. Detect a generating set Σ of symmetry inducing variable and literal permutations on the CNF using Saucy (as Shatter does).
2. Search heuristically for a large set of variable involutions Σ' generated by Σ, as explained in Section 3.2.
3. Extract disjoint variable matrices M_i and accompanying M_i-row permutations from Σ', as explained in Section 3.2.
4. Break each symmetry group induced by an M_i-row interchangeability constraint completely by choosing the right order $<_W$ for the accompanying

M_i-row permutations $\sigma_{i,i+1}$, and post the appropriate $\alpha \leq_{\text{lex}} \sigma_{i,i+1}(\alpha)$ constraints as explained in Section 3.1.

5. Use the order $<_W$ obtained in the previous step to break each symmetry σ_P induced by a generator permutation $P \in \Sigma$ by the same constraint Shatter uses: $\alpha \leq_{\text{lex}} \sigma_P(\alpha)$.

So BreakID actually is an extension of Shatter, generating extra symmetry breaking constraints for any detected row interchangeable variable matrix. Also, each symmetry breaking constraint $\phi \equiv \alpha \leq_{\text{lex}} \sigma(\alpha)$ is converted to clausal form using the same techniques as those used by Shatter [22]. This results in ϕ having a size linear in the number of variables of Π. The number of constraints ϕ is bounded by the number of variables of Π, hence the overall size of the symmetry breaking constraint is quadratic in the number of variables of Π.

4.2 Experimental verification

To verify the relevance of the BreakID preprocessor, we sent two symmetry breaking solvers to the 2013 international SAT competition. The first one was BreakID coupled with state-of-the-art SAT solver Glucose 2.1 [7], the second one consists of the Shatter preprocessor coupled to the same Glucose 2.1. We denote these approaches with the terms *BreakIDGlucose* and *ShatterGlucose* respectively. The results can be found at the SAT competition website [8], where BreakIDGlucose obtained the gold medal on the SAT+UNSAT hard combinatorial track, while ShatterGlucose reached the 8th spot. BreakIDGlucose outperforming ShatterGlucose shows that interchangeable row symmetry is relevant in today's SAT competition problems, while BreakIDGlucose outperforming Glucose 3.0 and other solvers shows that state-of-the-art solvers do not handle these symmetries very well.

One particular problem submitted to the competition was the "tph" problem, where two pigeons could be assigned to one hole, but $2n + 1$ pigeons had to be assigned to n holes, rendering the problem unsatisfiable. The only solver equipped for this problem was Lingeling [27], which could detect the cardinality constraints encoded in the clauses, and solve them with Gaussian elimination. The results of ShatterGlucose and BreakIDGlucose on tph are compared to Lingeling and all other solvers in Table 1.

Unsurprisingly, Lingeling has no problem with these instances, solving them in less than a second. All (!) other solvers can only solve the smallest instance, explained by the fact that their resolution proof is exponential [9]. ShatterGlucose does a bit better: its detected symmetries allow it to prune away a (small) part of the search tree. However, since ShatterGlucose cannot reason on row interchangeability, none of the bigger instances were solved. BreakIDGlucose also does better, even solving some of the larger instances, which can only be explained by the fact that BreakIDGlucose was able to detect and break most of tph's row interchangeability. However, for many tph instances, BreakIDGlucose still did not find a solution, because its row interchangeability detection scheme is still quite weak – if one cannot detect that *all* pigeons are interchangeable, then the search space remains exponential.

Instance	All other solvers	ShatterGlucose	BreakIDGlucose	Lingeling
tph6.cnf	5̃00	3.3005	547.611	0.004
tph7.cnf		102.876	201.638	0.007
tph8.cnf			201.315	0.012
tph9.cnf				0.019
tph10.cnf				0.027
tph11.cnf				0.039
tph12.cnf				0.061
tph14.cnf				0.125
tph15.cnf				0.174
tph16.cnf				0.242
tph18.cnf			208.46	0.4799
tph19.cnf			212.754	0.6269
tph20.cnf				0.7999

Table 1. SAT13 competition results on tph in seconds. Blank means timeout (5000s)

While this experiment shows the potential of complete row interchangeability breaking in SAT, it is clear that our heuristic detection scheme leaves much to be desired. As a result, working out better detection algorithms for the PRID problem seems a promising direction to speed up modern SAT solvers.

5 Conclusion

In this paper, we showed that the frequently occurring symmetry properties of interchangeable variables and interchangeable values in CSPs map to interchangeable row symmetries in their standard SAT encoding. We then argued that since SAT problems are CSP problems, and since interchangeable row symmetry groups can be completely broken in CSP with a simple row ordering constraint, that this must be possible in SAT as well. However, until now, no system had been devised that could both detect and completely break interchangeable row symmetries starting from a CNF theory. We proposed a first row interchangeability detection scheme for CNF, and constructed a polynomially sized symmetry breaking constraint that indeed breaks interchangeable row symmetry completely. Even though the row interchangeability detection mechanism is very rudimentary and leaves much room for improvement, BreakIDGlucose still realised significant speedups.

As a result, we are convinced that the PRID problem in CNF is an important research question, worthy of attention of the SAT community. One idea to solve PRID in CNF is to use computational algebra techniques to find the needed consecutive M-row swapping permutations, or to adjust the current graph based symmetry detection tools to actively look for involutions, as to improve variable matrix extraction from the set of generating variable permutations. Another area of further research is to investigate how other complete symmetry breaking techniques relate to row interchangeability in SAT. For instance, the SymChaff solver breaks its symmetry completely, which might point to a different way of exploiting row interchangeability in SAT.

References

1. Mears, C., Garcia de la Banda, M., Wallace, M.: On implementing symmetry detection. Constraints **14**(4) (2009) 443–477
2. Van Hentenryck, P., Flener, P., Pearson, J., gren, M.: Compositional derivation of symmetries for constraint satisfaction. In Zucker, J.D., Saitta, L., eds.: Abstraction, Reformulation and Approximation. Volume 3607 of LNCS. Springer Berlin Heidelberg (2005) 234–247
3. Benhamou, B., Saıdi, M.R.: Dynamic detection and elimination of local symmetry in CSPs
4. Walsh, T.: Symmetry breaking constraints: Recent results. CoRR **abs/1204.3348** (2012)
5. Mears, C., Garcia de la Banda, M., Demoen, B., Wallace, M.: Lightweight dynamic symmetry breaking. Constraints (2013) 1–48
6. Gent, I.P., Harvey, W., Kelsey, T., Linton, S.: Generic SBDD using computational group theory. In: Principles and Practice of Constraint Programming–CP 2003, Springer (2003) 333–347
7. Audemard, G., Simon, L.: Refining restarts strategies for SAT and UNSAT. In Milano, M., ed.: Principles and Practice of Constraint Programming. LNCS. Springer Berlin Heidelberg (2012) 118–126
8. Balint, A., Belov, A., Heule, M.J., Matti, M.J.: The 2013 international SAT competition. `http://satcompetition.org/2013/results.shtml` (2013)
9. Haken, A.: The intractability of resolution. Theoretical Computer Science **39** (1985) 297 – 308 Third Conference on Foundations of Software Technology and Theoretical Computer Science.
10. Walsh, T.: SAT v CSP. In: Principles and Practice of Constraint Programming - CP 2000. Springer (2000) 441–456
11. Kasif, S.: On the parallel complexity of discrete relaxation in constraint satisfaction networks. **45**(3) (1990) 99–118
12. Cohen, D., Jeavons, P., Jefferson, C., Petrie, K.E.: Symmetry definitions for constraint satisfaction problems. In: Constraints. (2006) 115–137
13. Flener, P., Pearson, J., Sellmann, M., Van Hentenryck, P.: Static and dynamic structural symmetry breaking. In Benhamou, F., ed.: Principles and Practice of Constraint Programming - CP 2006. Volume 4204 of LNCS. Springer Berlin Heidelberg (2006) 695–699
14. Shlyakhter, I.: Generating effective symmetry-breaking predicates for search problems. Discrete Appl. Math. **155**(12) (June 2007) 1539–1548
15. 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. Volume 2470 of LNCS. Springer Berlin Heidelberg (2002) 462–477
16. Puget, J.F.: Breaking all value symmetries in surjection problems. In Beek, P., ed.: Principles and Practice of Constraint Programming - CP 2005. Volume 3709 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 490–504
17. Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. In Bessiere, C., ed.: CP'07. Volume 4741 of LNCS., Springer (2007) 529–543
18. De Cat, B., Bogaerts, B., Bruynooghe, M., Denecker, M.: Predicate logic as a modelling language: The IDP system. CoRR **abs/1401.6312** (2014)
19. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM'02) **11**(2) (2002) 256–290
20. Katebi, H., Sakallah, K.A., Markov, I.L.: Symmetry and satisfiability: An update. In Strichman, O., Szeider, S., eds.: SAT. Volume 6175 of LNCS., Springer (2010) 113–127

21. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In Applegate, D., Brodal, G.S., Panario, D., Sedgewick, R., eds.: Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics, SIAM (2007) 135–149
22. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for Boolean satisfiability. IEEE Transactions on Computers **55**(5) (2006) 549–558
23. Devriendt, J., Bogaerts, B., Mears, C., De Cat, B., Denecker, M.: Symmetry propagation: Improved dynamic symmetry breaking in SAT. In: Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence, ICTAI'12. (2012)
24. Sabharwal, A.: Symchaff: exploiting symmetry in a structure-aware satisfiability solver. Constraints **14**(4) (2009) 478–505
25. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. (1996)
26. Devriendt, J., Bogaerts, B.: BreakID, a symmetry breaking preprocessor for sat solvers. `https://bitbucket.org/krr/symbreaker` (2013)
27. Biere, A.: Lingeling, Plingeling and Treengeling entering the SAT competition 2013. Proceedings of SAT Competition 2013 (2013) 51