Orbitopal Fixing in SAT

Markus Anders[1], Cayden Codel[2], and Marijn J. H. Heule[2]

[1] RPTU Kaiserslautern-Landau, Kaiserslautern, Germany
anders@cs.uni-kl.de
[2] Carnegie Mellon University, Pittsburgh, PA, United States
{ccodel,mheule}@cs.cmu.edu

Abstract. Despite their sophisticated heuristics, boolean satisfiability (SAT) solvers are still vulnerable to symmetry, causing them to visit search regions that are symmetric to ones already explored. While symmetry handling is routine in other solving paradigms, integrating it into state-of-the-art proof-producing SAT solvers is difficult: added reasoning must be fast, non-interfering with solver heuristics, and compatible with formal proof logging. To address these issues, we present a practical static symmetry breaking approach based on *orbitopal fixing*, a technique adapted from mixed-integer programming. Our approach adds only *unit clauses*, which minimizes downstream slowdowns, and it emits succinct proof certificates in the substitution redundancy proof system. Implemented in the SATSUMA tool, our methods deliver consistent speedups on symmetry-rich benchmarks with negligible regressions elsewhere.

1 Introduction

Boolean satisfiability (SAT) solvers power a wide range of industrial and academic applications [9]. Yet despite decades of innovation, state-of-the-art SAT solvers still lack robust, broadly deployed mechanisms for symmetry reasoning, even though such mechanisms are commonplace in other paradigms [19, 33]. Without explicit symmetry reasoning, solvers can waste significant amounts of time exploring search regions that are isomorphic to ones already ruled out, leading to substantial slowdowns on highly symmetric instances.

To address this problem, prior work in SAT has explored both preprocessing (static) and on-the-fly (dynamic) symmetry-breaking techniques [1, 3, 14–17, 24, 36]. The most used approach in SAT is *static symmetry breaking*, which adds constraints to the formula before solving to avoid isomorphic solutions. For example, in graph coloring, one can fix the color of a designated vertex, since any valid coloring can be permuted accordingly. Static methods are attractive in practice because their overhead is often modest [3].

Although these techniques can yield substantial speedups on highly symmetric formulas, they can also incur severe regressions elsewhere. A key culprit of this slowdown is overly aggressive symmetry breaking: Adding too many clauses to the formula often causes the solver's performance to degrade, especially when the formula is *satisfiable*. (For instance, see work by Aloul et al. [1].) Overall, symmetry handling techniques must strike a delicate balance between reasoning strength, computational cost, and minimal interference with solver heuristics.

Complicating this trade-off even further, SAT symmetry-breaking techniques must also be compatible with proof production. This is because modern SAT solvers (since 2016) are *certifying algorithms* [29], meaning that they emit formally checkable proofs that their answers are correct. Any additional symmetry reasoning must therefore integrate cleanly with proof generation and verification.

Today, practical SAT symmetry-breaking tools suffer from several drawbacks. All current tools produce structured lex-leader constraints [14], which can blow up the size of the formula and degrade learned clause quality when encoded into SAT. Proof logging is also problematic. Proof logging for practical symmetry-breaking tools was only introduced very recently by means of the dominance rule [10]. This approach has received notable success, with an implementation in BreakID [17] earning a special prize at SAT Competition 2023. But while dominance-based rules are very general, the new proof systems needed to support them are complicated to implement, and their proofs are slow to check.

Interestingly, some symmetry-breaking techniques for mixed-integer programming (MIP) avoid the problems of using large lex-leader constraints by applying symmetry reasoning in a more surgical manner. For formulas that exhibit so-called row symmetry, *orbitopal fixing* [27] breaks symmetries by adding only *unit clauses*. This is accomplished by combining insights on symmetry with insights on cardinality. Adapting such a technique to SAT should have far fewer downsides than introducing long, structured constraints, such as lex-leader constraints.

Contribution. To tackle the challenges discussed above, we adapt orbitopal fixing from MIP to SAT to introduce three new methods of practical symmetry handling. All of our methods follow three guiding principles:

1. They exclusively add *unit clauses* to the formula.
2. They simultaneously exploit *symmetry* and *cardinality*.
3. They generate succinct proof certificates in the substitution redundancy (sr) proof system [12, 20], without the need for dominance-based rules.

We implement our new techniques in the state-of-the-art symmetry breaking tool satsuma [3].[1] Despite the apparent restrictions—foregoing lex-leader constraints and feature-rich proof systems—it turns out that, indeed, our approach produces strong practical results:

1. The performance of the state-of-the-art SAT solver CaDiCaL [8] is substantially improved on the SAT Competition 2025, the SAT anniversary track of 2022, and a set of highly symmetric crafted benchmarks.
2. The preprocessing overhead is negligible (less than 1% of average solve time).
3. The performance regression on *satisfiable instances* is significantly smaller than for lex-leader constraints (even though lex-leader constraints achieve overall better pruning than our techniques on *unsatisfiable* instances).
4. The sr proofs are succinct, easy to generate, and efficient to check.

[1] https://github.com/markusa4/satsuma.

Overall, our techniques offer a more lightweight, surgical, and stable approach to SAT symmetry breaking than lex-leader constraints, and our techniques can be easily combined with other symmetry handling methods.

2 Preliminaries

We assume that the reader is generally familiar with concepts from SAT solving. For a broad introduction to the topic, see the Handbook of Satisfiability [9].

The propositional formulas we consider in this paper are all in *conjunctive normal form* (CNF), meaning that they are conjunctions of disjunctive *clauses* containing *literals*. A literal ℓ is either a variable v or its negation $\overline{v}$. In this paper, we interpret clauses and formulas as sets. For example, we sometimes write the clause $(x \vee \overline{y} \vee z)$ as $\{x, \overline{y}, z\}$. Let $\mathrm{Var}(F)$ and $\mathrm{Lit}(F)$ be the set of variables and literals occurring in F, respectively.

Two formulas F and F' are *equisatisfiable* if F is satisfiable iff F' is satisfiable. This definition is bidirectional, but since we only consider formulas F' that are formed by adding clauses to F (i.e., $F \subseteq F'$), the reverse direction is trivial, and thus we omit it in our proofs.

2.1 Unique Literal Clauses

A clause $C \in F$ is a *unique literal clause* (ULC) with respect to F if none of its literals $\ell \in C$ appear elsewhere in $F \setminus C$. ULCs enjoy the following property, which is key to adapting orbitopal fixing to SAT (see Section 3.1).

Lemma 1 ([38], Lemma 4). *Let F be a formula, and let $C \in F$ be a ULC. If F is satisfiable, then it can be satisfied by a truth assignment that sets exactly one literal in C to true.*

Another nice property of ULCs is that the set of ULCs in a formula F can be computed in linear time: First store how many times each literal appears in F, and then check each clause to see if all of its literals appear exactly once in F.

2.2 Syntactic Symmetry of Formulas

A symmetry σ of a formula F is a permutation of $\mathrm{Lit}(F)$ that maps F to itself. Formally, let σ be a permutation of $\mathrm{Lit}(F)$, and define $\sigma(F)$ as the formula created by relabeling the literals of F under σ. Then σ is a *syntactic symmetry* of F if:

1. $\sigma(F) = F$, and
2. $\neg\sigma(l) = \sigma(\overline{\ell})$ for all $\ell \in \mathrm{Lit}(F)$ (i.e., σ commutes with negation).

When defining a symmetry σ, condition (2) says it is sufficient to specify $\sigma(\ell)$ for only positive literals ℓ. We will write symmetries as $\sigma \coloneqq (\ell \mapsto \ell', \dots)$, meaning that $\sigma(\ell) = \ell'$. All literals not explicitly listed are assumed to map back to themselves, i.e., $\sigma(x) = x$.

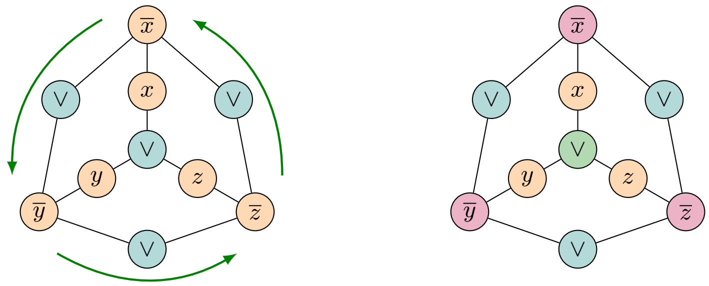


Fig. 1. A graph modeling the symmetries of the formula $(x \vee y \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{y} \vee \overline{z})$. Every clause is connected to its component literals, and every literal is adjacent to its negation. On the left, the green arrows indicate the symmetry mapping x to y, y to z, and z to x. On the right, the colors indicate the orbits of the vertices.

Example 1. Let $F = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{y} \vee \overline{z})$. Then the permutation $\sigma := (x \mapsto y, y \mapsto z, z \mapsto x)$ is a symmetry of F, since

$$\sigma(F) = (y \vee z \vee x) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{y} \vee \overline{x}) \wedge (\overline{z} \vee \overline{x}) = F.$$

In practice, the symmetries of a CNF formula are computed by modeling the formula as a graph and then giving that graph to an off-the-shelf graph isomorphism solver, such as NAUTY [30, 31], BLISS [25, 26], TRACES [31, 34], or DEJAVU [4, 5]. Figure 1 (left) illustrates a graph modeling the symmetries of Example 1. More-compact graph representations are typically used in practice [2].

As it turns out, the symmetries of a formula $\mathrm{Aut}(F)$[2] form a *permutation group*, which means we can use concepts from group theory to reason about them. In this paper, we use two such concepts: stabilizers and orbits.

Stabilizers are sets of symmetries that map literals back to themselves in certain ways. The *pointwise stabilizer* $\mathrm{Aut}(F)_{(L)}$ contains all symmetries of F that stabilize each individual literal in a set of literals L, while the *setwise stabilizer* $\mathrm{Aut}(F)_{\{L\}}$ contains all symmetries of F that map L back to itself. Formally,

$$\mathrm{Aut}(F)_{(L)} \coloneqq \{\sigma \in \mathrm{Aut}(F) \mid \sigma(\ell) = \ell \text{ for all } \ell \in L\}, \qquad \text{(Pointwise)}$$
$$\mathrm{Aut}(F)_{\{L\}} \coloneqq \{\sigma \in \mathrm{Aut}(F) \mid \sigma(L) = L\}. \qquad \text{(Setwise)}$$

Since the condition for pointwise stabilizers is stronger than the one for setwise stabilizers, we have that $\mathrm{Aut}(F)_{(L)} \subseteq \mathrm{Aut}(F)_{\{L\}}$. Each set is always nonempty, since the identity permutation is a member of both stabilizers.

The *orbit* of a literal ℓ is the set of literals that can be reached from ℓ by a permutation group G of symmetries of F. Often, $G = \mathrm{Aut}(F)$, but G is allowed

[2] The notation Aut() is due to how the symmetries of F are also its **aut**omorphisms.

to be any subgroup of $\mathrm{Aut}(F)$. Two literals ℓ_1 and ℓ_2 are *in the same orbit* with respect to G if there exists a symmetry $\sigma \in G$ such that $\sigma(\ell_1) = \ell_2$. The orbits under G form a partition of the literals, where literals in the same orbit are in the same equivalence class. Figure 1 (right) illustrates the orbits of Example 1.

For a more general introduction to permutation groups, we refer the reader to work by Seress [37].

2.3 Substitution Redundancy Proofs

When performing static symmetry breaking for SAT, we add *symmetry-breaking clauses* to a CNF formula F to forbid symmetric solutions. For the additions to be valid, we must prove that each addition preserves the equisatisfiability of F. We can write such a proof of equisatisfiability in a *clausal proof system*, and we can check the proof with a formally-verified proof checker. In this paper, we use the *substitution redundancy* (SR) proof system [12, 20, 35], which is a generalization of the popular RAT [23] and PR [21] proof systems.

In a clausal proof system, each proof step either adds a clause or deletes a clause. Added clauses C must be *redundant*, meaning that F and $(F \wedge C)$ are equisatisfiable. Addition steps may also include a *witness* ω that helps prove that C is redundant. Crucially, these witnesses allow for efficient proof checking.

In SR, the witness is a substitution $\omega : \mathrm{Var}(F) \to \mathrm{Lit}(F) \cup \{\top, \bot\}$ that maps each variable to a literal or to a fixed truth value.[3] This substitution extends to literals in the natural way. As it turns out, substitutions can express the symmetry reasoning involved in orbitopal fixing, which makes it easy to generate SR proofs for the symmetry-breaking clauses we discuss in this paper.

To show that C is redundant, it is sufficient to show that C is *substitution redundant* (SR) for F. In the definition below, we use the following notation. We write $\neg C$ for the negation of the disjunctive clause C, i.e., $\neg C \coloneqq \wedge_{\ell \in C} \overline{\ell}$. We write $F_{|\omega}$ for the reduction of the formula F under the substitution ω, where every literal ℓ in F is replaced with $\omega(\ell)$. We write $\vdash_1$ for entailment via unit propagation, where $F \vdash_1 \bot$ means that F causes a contradiction under unit propagation, $F \vdash_1 C$ means that $F \wedge \neg C \vdash_1 \bot$, and $F \vdash_1 G$ means that $F \wedge \neg D \vdash_1 \bot$ for all $D \in G$.

Definition 1 (Substitution redundant). *A clause C is* substitution redundant *for a formula F if there exists a substitution ω such that $F \wedge \neg C \vdash_1 (F \wedge C)_{|\omega}$.*

Intuitively, the witness ω provides a way to repair any assignment τ that satisfies F but not C into an assignment that satisfies both. If ω expresses a symmetry of F, then it suffices to show that the repaired assignment $\tau \circ \omega$ satisfies C, where "$\circ$" acts as a kind of function composition, with $(\tau \circ \omega)(\ell) = \omega(\ell)$ if $\omega(\ell) \in \{\top, \bot\}$ and $(\tau \circ \omega)(\ell) = \tau(\omega(\ell))$ otherwise.

The SR rule uses unit propagation $\vdash_1$ rather than general entailment $\vDash$ because the use of unit propagation enables the SR rule to be checked efficiently by proof

[3] PR witnesses are partial assignments, and RAT witnesses are partial assignments on a single literal. Thus, SR is a natural generalization of these two systems.

checkers. Today, only the DSR/LSR [13] and VeriPB [20] proof formats support SR reasoning. Our tool can generate proofs in either format.

Example 2. Consider the pigeonhole problem (PHP) of placing m pigeons into n holes such that each pigeon gets its own hole. Whenever $m > n$, this task is impossible. A common SAT encoding of PHP is:

$$\text{PHP}(m,n) = \bigwedge_{j=1}^{m} \left(\bigvee_{i=1}^{n} p_{i,j} \right) \wedge \bigwedge_{i=1}^{n} \bigwedge_{1 \le j < k \le m} \left(\overline{p}_{i,j} \vee \overline{p}_{i,k} \right),$$

where the variables $p_{i,j}$ mean that pigeon j is placed in hole i. When visualized as a matrix, the m columns contain the at-least-one constraints, and the n rows contain the at-most-one constraints.

This encoding exhibits a lot of symmetry. In particular, we are free to relabel the holes or the pigeons however we wish. (In other words, the encoding exhibits *row symmetry*; see Section 3.1.) The presence of this symmetry allows us to use SR reasoning to add redundant clauses to the formula.

Suppose we want to use the SR rule to show that pigeon 1 does not go in hole 1, i.e., that the unit clause $C = \{\overline{p}_{1,1}\}$ is SR. Since the left-hand side of the $\vdash_1$ turnstile in the SR rule assumes $\neg C$, we are essentially assuming that pigeon 1 gets placed in hole 1. To "repair" this situation, we will use the witness ω that swaps holes 1 and 2, meaning that pigeon 1 now gets placed in hole 2. Formally, $\omega \coloneqq (p_{1,1} \mapsto \bot,\, p_{2,1} \mapsto \top,\, p_{1,j} \mapsto p_{2,j},\, p_{2,j} \mapsto p_{1,j})$. Note that ω explicitly sets the truth values for $p_{1,1}$ and $p_{2,1}$, which forces pigeon 1 to be placed in hole 2. All other variables for holes 1 and 2 get swapped. Viewing the variables as a matrix, ω swaps rows 1 and 2.

We now show that C and ω satisfy the SR condition. The good news is that most clauses in $(F \wedge C)_{|\omega}$ have a trivial unit propagation refutation. In general, any clause $D \in F \wedge C$ where $D_{|\omega} = \top$ or $D_{|\omega} = D$ has a trivial refutation. Here, the at-least-one column constraint containing $p_{2,1}$ and any at-most-one row constraints containing $\overline{p}_{1,1}$ are satisfied by ω, and the remaining column constraints and the constraints for rows 3 through n are mapped back to themselves under ω. That leaves the at-most-one constraints for rows 1 and 2.

The row 1 constraints are easy. Either they contain $\overline{p}_{1,1}$ and are satisfied by ω, or they are mapped to a row 2 constraint, which causes a trivial refutation.

Finally, for the row 2 constraints, we do some unit propagation. By assuming $\neg C = \{p_{1,1}\}$ on the left-hand side of $\vdash_1$, we can derive $\{\overline{p}_{1,j}\}$ for all $j \neq 1$ via unit propagation on $\{\overline{p}_{1,1}, \overline{p}_{1,j}\}$. This lets us derive a refutation with the row 2 constraints on the right-hand side of $\vdash_1$, since any constraint $\{\overline{p}_{2,k}, \overline{p}_{2,k'}\}$, mapped under ω will contain $\overline{p}_{1,j}$ for some j, conflicting with the $\{\overline{p}_{1,j}\}$ unit we derived.

3 Fixing Rules

In this section, we introduce our new symmetry-breaking techniques. Notably, our techniques exclusively *assign* or *fix variables*, i.e., they exclusively add unit clauses to the formula. We also prove that each technique preserves equisatisfiability and is compatible with SR proof production.

3.1 Orbitopal Fixing

Our first symmetry-breaking technique is *orbitopal fixing*, which combines row symmetry in a matrix of literals M with the presence of unique literal clauses (ULCs) in the formula to fix literals of M. Our technique is inspired by a procedure of the same name used in MIP [27, 32].

Intuitively, a subset of a formula's literals exhibit row symmetry if they can be arranged into a rectangular matrix M such that there exist symmetries that swap any two rows of M. Formally, let F be a formula, and let $M := (\ell_{i,j})$ be an $n \times m$ matrix comprising a subset of $\mathrm{Lit}(F)$. Then M exhibits *row symmetry* [18] if there are symmetries $\{\sigma_{i_1,i_2}\}_{i_1,i_2 \in [1,n]} \subseteq \mathrm{Aut}(F)$ that swap rows i_1 and i_2 via:

$$\sigma_{i_1,i_2}(\ell) = \begin{cases} \ell_{i_2,j} & \text{if } \ell = \ell_{i_1,j} \text{ for some } j \in [1,m] \\ \ell_{i_1,j} & \text{if } \ell = \ell_{i_2,j} \text{ for some } j \in [1,m] \\ \ell_{i,j} & \text{if } \ell = \ell_{i,j} \in M \text{ and } i \notin \{i_1, i_2\} \end{cases} .$$

Note that for our purposes, row swaps are free to affect literals $\mathrm{Lit}(F) \setminus M$ that lie outside of the matrix. By composing row swaps, every possible reordering of the rows can be achieved. In group-theoretic terms, these swaps generate the symmetric group over the rows. Row symmetry and related structures are crucial for practical symmetry-handling algorithms, and they can be detected by generator-based [16, 22, 33] or more-recently developed graph-based approaches [3].

We now turn to orbitopal fixing. Suppose our formula F has a matrix of literals M that exhibits row symmetry. If each column of M is a unique literal clause of F, then we may fix the bottom literal in the first column $\ell_{n,1}$ to true and all literals in the upper-triangular portion above $\ell_{n,1}$ to false. Figure 2 shows an example of orbitopal fixing. More formally:

Definition 2 (Orbitopal fixing). *Let F be a formula, and let $M := (\ell_{i,j})$ be an $n \times m$ matrix that exhibits row symmetry in F. If every column of M is ULC with respect to F, i.e., if $C_j := \{\ell_{i,j}\}_{i \in [1,n]} \in F$ is ULC for every j, then* orbitopal fixing *derives unit clauses $(\ell_{n,1})$ and $(\overline{\ell}_{i,j})$ for every $j \in [1, \min(n,m)]$ and $i \leq n - j$.*

At first, it might be surprising that we may fix so many literals at once. But by using the property of ULCs from Lemma 1, we may assume that every column of M is satisfied by exactly one literal. This assumption allows us to strategically swap the rows with the satisfied literals into the lower-triangular portion of M, thus allowing us to fix the upper-triangular portion to false. This argument is formalized in the following lemma.

Theorem 1. *Let F be a formula with the conditions from Definition 2, and let $F \wedge L$ be the formula obtained by applying the orbitopal fixing rule to F. Then the following hold:*

1. *F and $F \wedge L$ are equisatisfiable.*

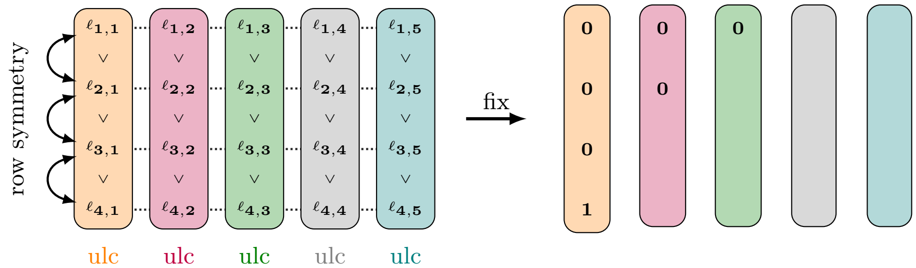


Fig. 2. An example of orbitopal fixing applied to a 4×5 matrix of literals from some formula F. The matrix exhibits row symmetry, and the columns are ULCs of F. Putting these two conditions together, we may fix the bottom-left literal $\ell_{4,1}$ to true and the upper-triangular portion of literals above $\ell_{4,1}$ to false.

2. *There exists an* SR *proof that adds the unit clauses of L to F in a particular order, namely, column-wise, top to bottom, left to right.*

Proof. **(1).** By Lemma 1, let τ be a satisfying assignment for F that satisfies each column C_j of the matrix M with exactly one true literal. We will now transform τ into a new assignment that also satisfies the additional constraints in $F \wedge L$.

First, consider the leftmost column. Let i be the row containing the satisfied literal of C_1 under τ. If $i \neq n$, then we may use the row symmetry $\sigma_{i,n}$ to change τ into a new assignment $\tau \circ \sigma_{i,n}$ by swapping the truth values for rows i and n. Since $\sigma_{i,n}$ is a symmetry of F, the assignment $\tau \circ \sigma_{i,n}$ still satisfies F, and by our assumption that τ satisfies exactly one literal of C_1, it also satisfies the unit clauses in L setting $\ell_{n,1}$ to true and $\ell_{i,1}$ to false for all $i < n$.

Now consider the j-th column, and let i be the row containing the satisfied literal of C_j. If $i \leq n - j$, then we may do the same thing as before and swap rows i and $n - j + 1$ to form $\tau \circ \sigma_{i,n-j+1}$. Note that swapping these two rows exchanges only falsified literals in the columns to the left, since their true literals lie beneath the $(n - j + 1)$-th row. As a result, we still satisfy the constraints in these columns. And since the true literal of C_j is now beneath row $n - j$, the new assignment sets every literal at and above this row to false, which satisfies the unit clauses in L corresponding to column j.

By applying this procedure to the columns in order from left to right, we obtain a modified τ that satisfies both F and $F \wedge L$.

(2). The series of SR clause additions generated by orbitopal fixing follows the proof of **(1)** exactly, except we must add the clauses in order (top to bottom, left to right), and we must provide a row-permutation witness for each unit clause. More specifically, when we add the unit clause $\{\overline{\ell}_{i,j}\}$, we use the witness ω with

$$\omega(\ell) \coloneqq \begin{cases} \bot & \text{if } \ell = \ell_{i,j} \\ \top & \text{if } \ell = \ell_{i+1,j} \\ \sigma_{i,i+1}(\ell) & \text{otherwise.} \end{cases}$$

Let $F \wedge L'$ be the formula constructed so far by adding unit clauses.

According to Definition 1, for us to show that $\{\overline{\ell}_{i,j}\}$ is SR, we must show that $F \wedge L' \wedge \{\ell_{i,j}\} \vdash_1 (F \wedge L' \wedge \{\overline{\ell}_{i,j}\})_{|\omega}$. For most clauses, this is trivial: any clauses $C \in F$ whose literals are not modified by ω, i.e., where $C \cap \{\ell \in \mathrm{Lit}(F) \mid \ell \neq \sigma(\ell)\} = \varnothing$, are immediately entailed. The new unit clause $\{\overline{\ell}_{i,j}\}$ is also trivially satisfied, since $\omega(\ell_{i,j}) = \bot$.

The remaining types of clauses modified by the witness are: (i) previously added unit literals L', (ii) clauses of F *not* containing the variables of $\ell_{i,j}$ and $\ell_{i+1,j}$, and (iii) clauses of F containing the variables of $\ell_{i,j}$ and $\ell_{i+1,j}$.

(Case i.) The order of the unit additions ensures that all previously added unit literals $\ell_{i,j'}$ for $1 \leq j' < j$ are mapped to other propagated literals $\ell_{i+1,j'}$. Hence, they also entail each other.

(Case ii.) For every clause $C \in F$ not containing the variables of $\ell_{i,j}$ or $\ell_{i+1,j}$, we observe that the symmetry $\sigma_{i,i+1}$ applies, and $\omega(C) \in F$ holds.

(Case iii.) The ULC containing $\ell_{i,j}$ and $\ell_{i+1,j}$ is entailed, since $\omega(\ell_{i+1,j}) = \top$. And because this clause is a ULC, all other clauses may only contain $\overline{\ell}_{i,j}$ and $\overline{\ell}_{i+1,j}$. All clauses containing $\overline{\ell}_{i,j}$ are immediately entailed.

It remains to show that clauses C containing $\overline{\ell}_{i+1,j}$ but not $\overline{\ell}_{i,j}$ are entailed. Consider the clause C' we obtain by mapping C under the row swap exchanging rows i and $i+1$, that is, $C' = \sigma_{i,i+1}(C)$. Since the row swap is a symmetry of F, $C' \in F$ holds. In other words, C' is a premise.

By assumption, $C \cap \{\overline{\ell}_{i,j}, \ell_{i,j}, \ell_{i+1,j}\} = \varnothing$ and $\overline{\ell}_{i+1,j} \in C$ hold, and thus we can conclude $C' = C_{|\omega} \cup \{\overline{\ell}_{i,j}\}$. Assuming $\neg C_{|\omega}$ together with the additional premise $\ell_{i,j}$ thus contradicts the premise C'. Hence, $C_{|\omega}$ is entailed. □

Example 3. The pigeonhole problem with m pigeons and n holes exhibits row symmetry when encoded as in Example 2, with the rows corresponding to the holes and the columns corresponding to the pigeons. Thus, orbitopal fixing may be applied. Figure 2 illustrates the case of 5 pigeons and 4 holes. When applying orbitopal fixing to this formula, we end up with the following formula:

$$\textsc{php}(5,4) \wedge \overline{p}_{1,1} \wedge \overline{p}_{2,1} \wedge \overline{p}_{3,1} \wedge p_{4,1} \wedge \overline{p}_{1,2} \wedge \overline{p}_{2,2} \wedge \overline{p}_{3,1}.$$

3.2 Clausal Fixing

Our second symmetry-breaking technique, called *clausal fixing*, is based on the observation that every clause must contain at least one satisfied literal. When all literals of a clause belong to the same orbit under the formula's symmetries, a representative literal can be fixed without loss of generality.

Definition 3 (Clausal fixing). *Let F be a formula with clause $\{\ell_1, \dots, \ell_k\} \in F$ where all ℓ_i are in the same orbit of* $\mathrm{Aut}(F)$*. That is, there are symmetries σ with $\sigma(\ell_1) = \ell_i$ for each i. Then* clausal fixing *derives the unit clause $\{\ell_1\}$.*

Intuitively, if we have any satisfying assignment, we know that there must be at least one satisfied literal in that clause. Using the orbit, we can always swap the satisfied literal with ℓ_1. Thus, we can just assign ℓ_1 directly.

We now formally prove the correctness of the clausal fixing rule.

Theorem 2. *Let $F \wedge \{\ell_1\}$ be the formula obtained from F by an application of the clausal fixing rule to clause $C = \{\ell_1, \ldots, \ell_k\}$. Then the following hold:*

1. *F and $F \wedge \{\ell_1\}$ are equisatisfiable.*
2. *There is an* SR *proof deriving $F \wedge \{\ell_1\}$ from F in k steps.*

Proof. **(1).** Let τ be a satisfying assignment of F. We show how to transform τ into an assignment that satisfies $F \wedge \{\ell_1\}$.

Let ℓ_i be a satisfied literal of C. If $\ell_i = \ell_1$, then τ would already satisfy $F \wedge \{\ell_1\}$, so assume otherwise. By definition of the clausal fixing rule, for every literal $\ell_i \in C$, there exists a symmetry σ with $\sigma(\ell_1) = \ell_i$. Using this symmetry, we obtain $\tau' = \tau \circ \sigma$, which now sets ℓ_1 to true. Since σ is a symmetry, the resulting assignment τ' still satisfies F.

(2). We obtain an SR proof as follows. First, the proof derives binary symmetry-breaking clauses $\{\ell_1, \overline{\ell}_i\}$ for all $i \in [2, k]$. Each of these clauses is SR using the symmetry ω_i mapping $\omega_i(\ell_1) = \ell_i$. After that, we may derive the unit clause $\{\ell_1\}$ by resolution on the added binary clauses.

It suffices to show that each binary clause $\{\ell_1, \overline{\ell}_i\}$ is SR. Let L' be the set of binary clauses we have already added. We must show that

$$F \wedge L' \wedge \{\overline{\ell}_1\} \wedge \{\ell_i\} \vdash_1 (F \wedge L' \wedge \{\ell_1, \overline{\ell}_1\})_{|\omega_i}.$$

The result is immediate: Every clause in F is entailed, since ω_i is a symmetry of F, and every clause in $C \in L' \cup \{\ell_1, \overline{\ell}_1\}$ is entailed by the unit clause $\{\ell_i\}$, since $\ell_1 \in C$ and $\omega_i(\ell_1) = \ell_i$. □

3.3 Negation Fixing

Lastly, we describe the *negation fixing* rule. When a literal can be mapped to its negation, we can fix it without loss of generality.

Definition 4 (Negation fixing). *Let F be a formula, let $\ell \in \mathrm{Lit}(F)$ be a literal, and let $\sigma \in \mathrm{Aut}(F)$ be a symmetry that maps $\sigma(\ell) = \overline{\ell}$. Then* negation fixing *derives the unit clause $\{\ell\}$.*

In a sense, the negation fixing rule also exploits cardinality: trivially, at least one of ℓ and $\overline{\ell}$ must be true. We prove the correctness of the rule.

Theorem 3. *Let $F \wedge \{\ell\}$ be a formula obtained from F by an application of the negation fixing rule. Then the following hold:*

1. *F and $F \wedge \{\ell\}$ are equisatisfiable.*
2. *The unit clause $\{\ell\}$ is* SR.

Proof. **(1.)** Let τ be a satisfying assignment of F. If τ sets ℓ to true, then we're done, so assume otherwise. Then the assignment $\tau \circ \sigma$ satisfies $F \wedge \{\ell\}$, where σ is the symmetry from the negation fixing rule swapping ℓ with $\overline{\ell}$.

(2.) The clause $\{\ell\}$ is SR using as witness the symmetry mapping $\sigma(\ell) = \overline{\ell}$. □

3.4 Repeated Applications of Rules

All of our rules use the symmetries of a formula F to add a set of unit clauses L to the formula, yielding a new formula $F \wedge L$. In turn, subsequent rule applications would use the symmetries of $F \wedge L$. However, recomputing formula symmetries after *every* rule application is not practical.

Instead of recomputing symmetries, we can *update* the set of applicable symmetries using pointwise and setwise stabilizers (see Section 2.2). This approach may indeed yield fewer applicable symmetries than computing the full group $\mathrm{Aut}(F \wedge L)$, but it's cheaper to do so.

Let us now observe that by stabilizing the set of added unit literals L, we obtain symmetries of $F \wedge L$ from the symmetries of F.

Lemma 2. *Let F be a CNF formula and $L \subseteq \mathrm{Lit}(F)$ a subset of its literals. It holds that* $\mathrm{Aut}(F)_{\{L\}} \subseteq \mathrm{Aut}(F \wedge \{\{\ell\} \mid \ell \in L\})$.

Proof. Let $\sigma \in \mathrm{Aut}(F)_{\{L\}}$. This means that $\sigma(F) = F$ and $\sigma(L) = L$. Consider a clause C of the formula $F \wedge \{\{\ell\} \mid \ell \in L\}$. If $C \in F$, then $\sigma(C) \in F \wedge L$ follows. If $C = \{\ell\}$ is one of the unit clauses from L, then $\sigma(\ell) \in L$ follows, and $\{\sigma(\ell)\} \in \{\{\ell\} \mid \ell \in L\}$. Hence, σ is a symmetry of $F \wedge \{\{\ell\} \mid \ell \in L\}$. □

Unfortunately, setwise stabilizers are also expensive to compute. (In fact, the problem of computing them is at least as hard as computing symmetries [28].) However, since $\mathrm{Aut}(F)_{(L)} \subseteq \mathrm{Aut}(F)_{\{L\}}$ holds, we can use pointwise stabilizers instead. Polynomial-time algorithms exist to find pointwise stabilizers [37], and they are often efficient in practice.

4 Implementation Details

We implemented our new fixing algorithms in the existing SATSUMA symmetry breaking tool [3]. The tool is implemented in `C++`.

To enable efficient symmetry handling, SATSUMA simplifies the formula in various ways: Duplicate literals are removed from clauses, duplicate clauses and tautological clauses are removed from the formula, and unit propagation is applied until fixpoint. But other than these simplifications, the tool only adds symmetry-breaking clauses. In particular, the ordering of literals and clauses in the formula remains unchanged.

Orbitopal Fixing. For the orbitopal fixing approach, we leverage the existing structure detection of SATSUMA for row symmetry, row-column symmetry, and the symmetries of so-called Johnson graphs [3].[4] Although orbitopal fixing is only defined over row symmetry, the more complex structures identified by SATSUMA

[4] A Johnson graph $J(n, k)$ represents the k-element subsets of an n-element set, where two vertices share an edge if their set intersection has cardinality $(k-1)$. Intuitively, Johnson graphs model the symmetries of the edges of complete graphs, or, more generally, relational structures.

often *contain* a row symmetry. In particular, row-column symmetry is inherently composed of two row symmetries. For Johnson symmetry, the relationship is more intricate, but certain instances of this structure can also contain an underlying row symmetry.

Once a structure is identified as potentially having row symmetry, we check its columns to see if they coincide with a unique literal clause in the formula. Orbitopal fixing is applied to columns that fulfill the condition.

Clausal Fixing. Our implementation of clausal fixing follows a four-step procedure:

1. *(Orbits.)* We first compute the orbits of the currently considered group.
2. *(Check clauses.)* Each clause is considered once. If all of its literals belong to the same orbit, then the clausal fixing rule is applied and one literal is propagated. Since orbit partitions refine strictly under pointwise stabilizers, a clause that does not qualify at this stage will not qualify at any later stage.
3. *(Witness.)* To generate SR proofs, we must identify symmetries that can act as witnesses. When we apply clausal fixing to a literal ℓ in a clause C, we explicitly compute symmetries σ such that $\sigma(\ell) = \ell'$ for each $\ell' \in C$.
4. *(Stabilize.)* Whenever propagation occurs, we update the group by taking the pointwise stabilizer of the propagated literals. Then we go back to Step 1 and recompute the orbits for the refined group.

The process stops once each clause has been checked once.

We employ two different algorithms to compute pointwise stabilizers and witness symmetries: a more involved Schreier-Sims-based implementation, and a faster heuristic for binary clauses.

1. The Schreier-Sims algorithm [37, 39] computes pointwise stabilizers. Internally, it stores a so-called transversal which can immediately provide the necessary witness symmetries. While it tends to be quite fast for many groups, it often exhibits quadratic scaling in practice. As a result, we set computational limits on the use of Schreier-Sims in our implementation. We use the Schreier-Sims implementation of the DEJAVU [4] library.
2. For large groups, we implement a more rudimentary heuristic which only tests binary clauses. It greedily searches for an existing generator that can serve as the witness symmetry. The heuristic takes pointwise stabilizers by filtering generators to ones that stabilize the desired points.

Negation Fixing. Negation fixing follows a very similar strategy to clausal fixing, except instead of iterating over clauses, it iterates over variables. For each variable v, we check if v and $\overline{v}$ are in the same orbit. As with clausal fixing, we employ both a Schreier-Sims based implementation and a more efficient heuristic.

5 Experimental Evaluation

We evaluated the effectiveness of our fixing techniques as implemented in SATSUMA on three benchmark suites. The first suite is from the anniversary track of the

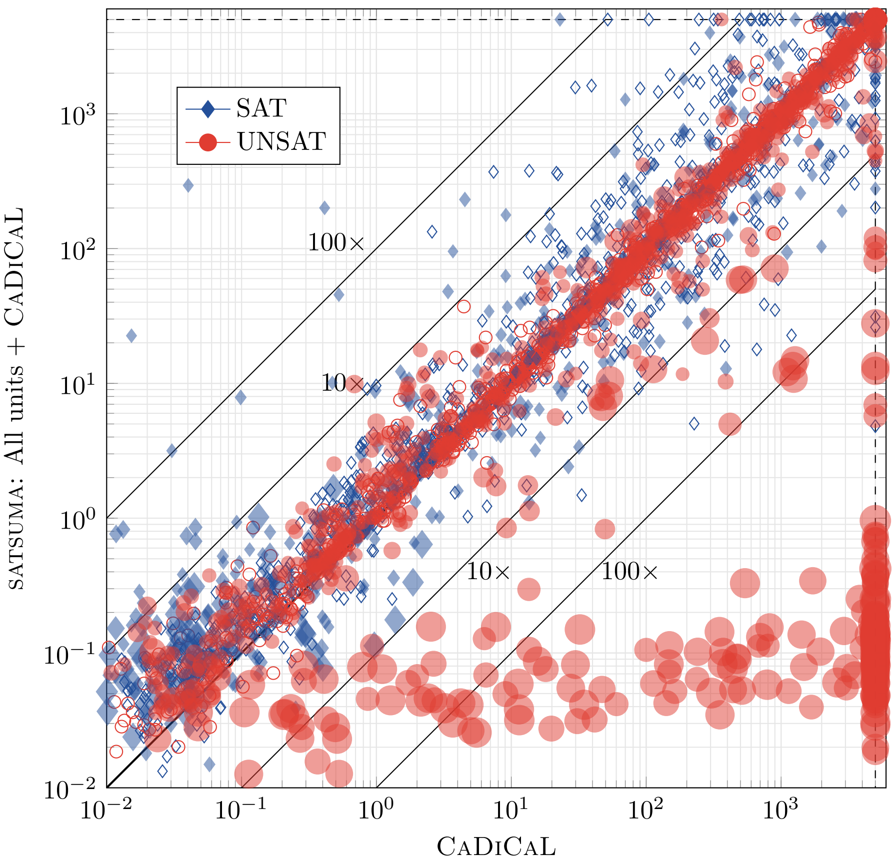


Fig. 3. A scatter plot of CaDiCaL with and without fixing on the anniversary suite. The times are in seconds and include preprocessing time. Empty marks denote that no units were added. The size of the marks correlates to the number of fixed units relative to the number of formula variables. Points below the diagonal benefited from fixing.

SAT Competition 2022 [6], comprising non-random benchmarks from the SAT Competitions 2002 to 2021, duplicates excluded. The second suite comprises the benchmarks from the main track of the SAT Competition 2025. The third suite comprises highly symmetric synthetic benchmarks that have appeared in various papers on symmetry breaking, including pigeonhole, Tseitin, clique coloring, and graph coloring formulas, as well as multiple instances from Ramsey theory.

For each suite, we filtered out all formulas larger than 1 GB in size, since for such large formulas symmetry handling often incurs prohibitive computational overhead, and are thus skipped by the symmetry breaking tools anyway. After this filter, the anniversary suite has 5344 formulas, the 2025 suite has 386 formulas, and the synthetic suite has 137 formulas.

We ran our experiments on the cluster at the Pittsburgh Supercomputing Center [11]. Each machine has 128 cores and 256 GB of RAM. We ran every tool in parallel across all cores. We used a timeout of 5000 seconds for CaDiCaL[5] (which matches the official timeout for the SAT competitions), as well as a 300 second timeout for satsuma and dsr-trim [13],[6] an (unverified) sr proof checker. Notably, all satsuma runs finished before timeout.

We ran satsuma with five different settings: each of our three fixing techniques individually, all of our techniques combined ("all-units"), and a control version of satsuma that produces lex-leader constraints. We checked all satsuma-generated sr proofs with dsr-trim. Then we ran CaDiCaL on all formulas.

Figure 3 shows the results of applying all fixing techniques to the anniversary benchmarks. Our fixing techniques allow CaDiCaL to solve many dozens of formulas very quickly, including 67 formulas that can be solved in a second of preprocessing time, while CaDiCaL without symmetry breaking times out after 5000 seconds. Many empty points below the diagonal are due to satsuma's formula simplification (especially on satisfiable instances). Note that few points are clearly above the diagonal, thereby showing that our techniques have minimal negative impact on the performance across the anniversary benchmarks.

Figure 4 shows similar results on the synthetic and 2025 suites. Notably, most of the synthetic benchmarks become easy after fixing, showing, somewhat surprisingly, that symmetry breaking is all that is needed to solve these instances.

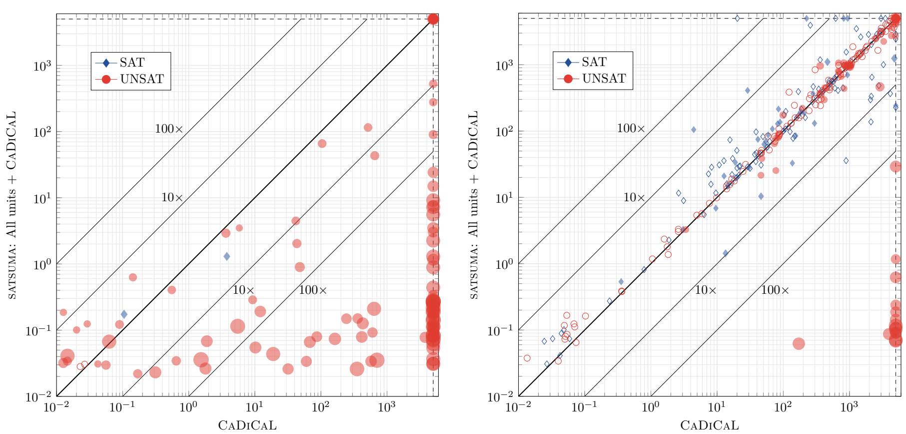


Fig. 4. Scatter plots of CaDiCaL with and without fixing on the synthetic suite (left) and the 2025 suite (right). The times are in seconds and include preprocessing time.

Figure 5 shows a cumulative solved formulas (CSF) plot comparing the runtimes of CaDiCaL with each satsuma setting against base CaDiCaL on the anniversary suite. The left CSF plot shows the regression due to symmetry

5 https://github.com/arminbiere/cadical. We used version 2.1.3.

6 https://github.com/ccodel/dsr-trim.

breaking on satisfiable instances, since symmetry breaking on satisfiable formulas typically has no benefit and can be harmful instead. The plot shows that the lex-leader setting performs the worst, while the new techniques limit the amount of regression. The regression is similar across SATSUMA settings, which suggests that SATSUMA's formula preprocessing and simplification dominate the costs.

The right CSF plot of Figure 5 shows the runtime performance on unsatisfiable instances. Lex-leader performs the best, but the all-units setting is close behind. Below them, each individual setting performs similarly, with clausal fixing performing the worst. All techniques outperform base CaDiCaL.

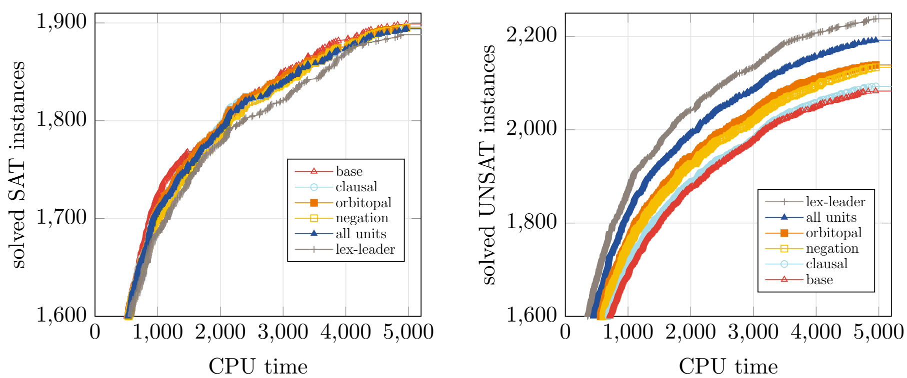


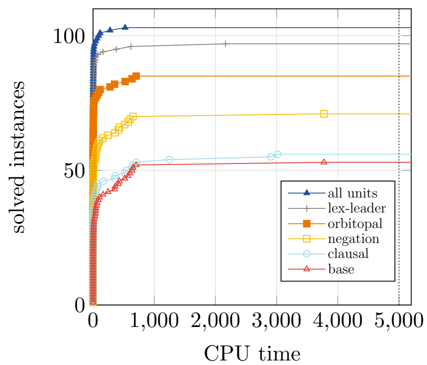
Fig. 5. CSF plots on SAT (left) and UNSAT (right) formulas of the anniversary suite

Figure 6 shows the CSF plots for the synthetic and 2025 suites. On the synthetic instances, the all-units configuration performs the best. The main reason for its success is that negation fixing can easily solve all Tseitin formulas, in contrast to lex-leader. (A similar approach could be implemented for lex-leader, but this is currently not present in SATSUMA.) On the 2025 suite, lex-leader shows the strongest performance, followed by the all-units configuration.

Table 1 summarizes the data shown in the figures.

As part of our experiments, we generated and checked SR and DRAT proofs. SATSUMA generated SR proofs when applying our fixing techniques, and CaDiCaL generated DRAT proofs for unsatisfiable formulas. All proofs were either accepted by the DSR-TRIM and LSR-CHECK proof checkers, or caused a memout/timeout. Proof checking time was generally low, with most proofs finishing in under 15 seconds. Table 2 summarizes the times taken for proof checking.

One particular advantage of SATSUMA's SR proof generation is that it composes with CaDiCaL's DRAT proof generation. For any unsatisfiable symmetry-broken formula, we appended the DRAT proof to the end of its SR proof, and then checked the proof with respect to the original formula. In this way, the two proofs form a complete proof of unsatisfiability of the original formula.



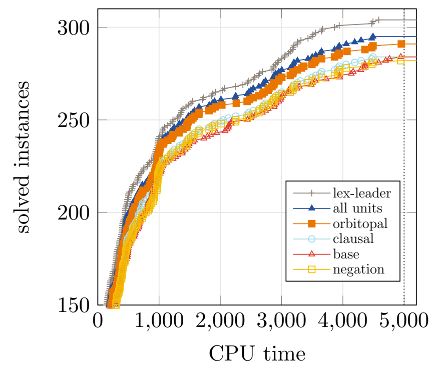


Fig. 6. CSF plots on the synthetic suite (left) and the 2025 suite (right)

Table 1. Results across the benchmark suites for the average preprocessing time in seconds (PPT), the average CaDiCaL runtime with preprocessing in seconds (CRT), and the average number of units added to the formula by the fixing technique (#U).

Suite	Anni22			Satcomp25			Synth		
Setting	PPT	CRT	#U	PPT	CRT	#U	PPT	CRT	#U
Orbitopal	2.51	1555.5	11.23	8.06	1746.7	8.37	0.44	1919.2	743.0
Negation	1.93	1572.0	32.59	5.77	1870.6	52.15	0.32	2469.4	16.5
Clausal	1.83	1605.7	23.64	5.45	1845.3	14.33	0.73	3041.0	15.2
All	1.81	1504.5	67.30	5.27	1694.3	71.48	0.76	1250.0	768.6
Lex-leader	1.73	1472.4	–	4.85	1593.7	–	0.32	1486.6	–
CaDiCaL	–	1607.8	–	–	1882.9	–	–	3138.6	–

Table 2. Results for SR proof checking times across the various benchmark suites and settings, showing the average DSR-TRIM checking time in seconds (DCT) and the average LSR-CHECK checking time in seconds (LCT).

Suite	Anni22		Satcomp25		Synth	
Setting	DCT	LCT	DCT	LCT	DCT	LCT
Orbitopal	5.35	0.76	15.15	1.36	8.42	4.19
Negation	5.76	1.43	16.80	1.64	0.04	0.01
Clausal	6.76	0.86	16.13	1.29	5.29	0.16
All	7.48	1.41	15.91	1.48	8.67	3.18

6 Conclusion

We presented new static symmetry-breaking techniques based solely on introducing unit clauses to a given formula, which we implemented in the state-of-the-art tool SATSUMA. The key insight behind these techniques was to combine symmetry and cardinality reasoning. This combination enabled us to, for the first time, implement meaningful, practical symmetry breaking that produces SR proofs instead of dominance-based proofs. Our experiments demonstrate significant performance improvements across a wide variety of benchmarks, with reduced regression compared to the lex-leader approach.

As for future work, we hope to expand the present techniques further to close the gap with the lex-leader approach on unsatisfiable instances. While this may be difficult by using only unit clauses, it would be interesting to achieve this without incurring any additional performance regression on satisfiable instances while still using lightweight SR proofs. Indeed, it would be intriguing to find concrete benchmark families that *can* be efficiently solved in practice using lex-leader constraints, but evade serious attempts at efficient, practical symmetry handling in SR. A concrete candidate might be small Ramsey numbers: while a very short SR proof for $R(4, 4, 18)$ is known [13], it is possible to easily solve the slightly harder $R(3, 7, 23)$ within few minutes using lex-leader constraints. We wonder if it is possible to generalize the symmetry breaking from the $R(4, 4, 18)$ proof to larger numbers. Separately, in MIP, orbitopal fixing can be used under more general conditions [7, 27], which may be interesting to adapt to SAT.

7 Data Availability and Reproducibility

All of our source code, benchmark formulas, and experimental results can be found at our artifact on Zenodo.[7] The artifact contains detailed instructions for how to download the formulas and compile the source code in a Docker container. The artifact also contains scripts to run experiments and compare the results to the ones from our paper.

Acknowledgements

We thank Christopher Hojny for pointing out important literature in MIP. This work has benefited substantially from Dagstuhl Seminar 25231 "Certifying Algorithms for Automated Reasoning." This work was supported by the National Science Foundation (NSF) under grant CCF-2415773 and funding from AFRL and DARPA under Agreement FA8750-24-9-1000.

[7] `https://doi.org/10.5281/zenodo.17491222`.

Bibliography

[1] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, pages 836–839. ACM, 2003.

[2] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(9):1117–1137, 2003.

[3] Markus Anders, Sofia Brenner, and Gaurav Rattan. Satsuma: Structure-based symmetry breaking in SAT. In *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*, volume 305 of *LIPIcs*, pages 4:1–4:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[4] Markus Anders and Pascal Schweitzer. Parallel computation of combinatorial symmetries. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[5] Markus Anders and Pascal Schweitzer. Search problems in trees with symmetries: Near optimal traversal strategies for individualization-refinement algorithms. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 16:1–16:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[6] Tomas Balyo, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2022: Solver and Benchmark Descriptions*, Department of Computer Science Series of Publications B. Helsinki Institute for Information Technology, 2022.

[7] Pascale Bendotti, Pierre Fouilhoux, and Cécile Rottner. Orbitopal fixing for the full (sub-)orbitope and application to the unit commitment problem. *Mathematical Programming*, 186(1):337–372, 2021.

[8] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024.

[9] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[10] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *J. Artif. Intell. Res.*, 77:1539–1589, 2023.

[11] Shawn T. Brown, Paola Buitrago, Edward Hanna, Sergiu Sanielevici, Robin Scibek, and Nicholas A. Nystrom. Bridges-2: A platform for rapidly-evolving and data intensive research. In *Practice and Experience in Advanced Research Computing 2021: Evolution Across All Dimensions*, PEARC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Sam Buss and Neil Thapen. DRAT and propagation redundancy proofs without new variables. *Logical Methods in Computer Science*, Volume 17, Issue 2, Apr 2021.

[13] Cayden R. Codel, Jeremy Avigad, and Marijn J. H. Heule. Verified substitution redundancy checking. In *Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024*, pages 186–196. IEEE, 2024.

[14] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 148–159. Morgan Kaufmann, 1996.

[15] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 83–100. Springer, 2017.

[16] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 104–122. Springer, 2016.

[17] Jo Devriendt, Bart Bogaerts, Broes De Cat, Marc Denecker, and Christopher Mears. 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*, pages 49–56. IEEE Computer Society, 2012.

[18] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 462–476. Springer, 2002.

[19] Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 329–376. Elsevier, 2006.

[20] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-boolean proofs. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applica-*

tions of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 3768–3777. AAAI Press, 2021.
[21] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *Journal of Automated Reasoning*, 64(3):533–554, 2020.
[22] Christopher Hojny and Marc E. Pfetsch. Polytopes associated with symmetry handling. *Math. Program.*, 175(1-2):197–240, 2019.
[23] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Automated Reasoning*, pages 355–370, 2012.
[24] Tommi A. Junttila, Matti Karppa, Petteri Kaski, and Jukka Kohonen. An adaptive prefix-assignment technique for symmetry reduction. *J. Symb. Comput.*, 99:21–49, 2020.
[25] Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007.
[26] Tommi A. Junttila and Petteri Kaski. Conflict propagation and component recursion for canonical labeling. In *Theory and Practice of Algorithms in (Computer) Systems - First International ICST Conference, TAPAS 2011, Rome, Italy, April 18-20, 2011. Proceedings*, volume 6595 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2011.
[27] Volker Kaibel, Matthias Peinhardt, and Marc E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595–610, 2011.
[28] Eugene M. Luks. Permutation groups and polynomial-time computation. In *Groups And Computation, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 7-10, 1991*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 139–175. DIMACS/AMS, 1991.
[29] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011.
[30] Brendan D. McKay. Practical graph isomorphism. In *10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)*, pages 45–87, 1981.
[31] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014.
[32] Gioni Mexi, Dominik Kamp, Yuji Shinano, Shanwen Pu, Alexander Hoen, Ksenia Bestuzheva, Christopher Hojny, Matthias Walter, Marc E. Pfetsch, Sebastian Pokutta, and Thorsten Koch. State-of-the-art methods for pseudo-boolean solving with SCIP, 2025.
[33] Marc E. Pfetsch and Thomas Rehn. A computational comparison of symmetry handling methods for mixed integer programs. *Math. Program. Comput.*, 11(1):37–93, 2019.
[34] Adolfo Piperno. Search space contraction in canonical labeling of graphs (preliminary version). *CoRR*, abs/0804.4881, 2008.
[35] Adrián Rebola-Pardo. Even shorter proofs without new variables. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference*

on Theory and Applications of Satisfiability Testing (SAT 2023), volume 271 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[36] Ashish Sabharwal. SymChaff: exploiting symmetry in a structure-aware satisfiability solver. *Constraints An Int. J.*, 14(4):478–505, 2009.

[37] Ákos Seress. *Permutation Group Algorithms.* Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

[38] Aeacus Sheng, Joseph E. Reeves, and Marijn J. H. Heule. Reencoding unique literal clauses. In *28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, August 12-15, 2025, Glasgow, Scotland*, volume 341 of *LIPIcs*, pages 29:1–29:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[39] Charles C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon, 1970.