Symmetry-breaking for SAT: The Mysteries of Logic Minimization

Fadi A. Aloul, Igor L. Markov and Karem A. Sakallah
University of Michigan, EECS
Outline

- Motivation and Goals
- Mathematical Background
- Previous work
- New Constructions for Single Perms
- Symmetry-breaking for Multiple Generators
- Conclusions and On-going work
Motivation

- Exponential gap in proof lengths
  - The pigeon-hole principle (as a SAT instance)
  - Exponential lower bounds for resolution proofs
    - Beame, Karp and Pitassi, 2002: $\Omega(2^{n/20})$
    - Resolution + reasoning by symmetry $\rightarrow$ poly-sized proofs
    - Krishnamurthy, 1988
- Lower bounds on resolution proofs apply to the behavior of DP/DLL SAT solvers
- One must also capture the complexity of symmetry extraction and the size of symmetry representation
Goals of This Work

- **Push the envelope of constraint satisfaction**
  - Problem instances from apps have symmss

- **Develop generic methods for CSPs**
  - For now, our focus is on Boolean Satisfiability
    - *This problem is fundamental to Math & CS*
  - Yet, we feel our work applies beyond SAT
    - 0-1 ILP and generic ILP are natural extensions

- **High-performance, competitive methods**
  - Must minimize overhead of dealing w symmetry
How We Do It

CNF → Graph → Call generic SAT solver → Pre-processed CNF instance → Graph symmetries → CNF symmetries
Benchmarking

- Take a strong SAT solver, e.g., Chaff
- Run it on a CNF benchmark
- Run the proposed flow
- Compare runtimes
  - Plain SAT solver \textit{versus}
  - Pre-processing + SAT solver
- No assumption about “symmetries being known” – this enables many applications
Why Pre-processing?

- Alternative: hack some SAT solver
  - More sophisticated strategies possible
    - *Ditto for publications* 😊
  - Can potentially capture “partial symmetries”
  - Additional overhead: is this really worth doing?
    - Li and Purdom, SAT 2002: *in some cases, yes*

- Do you really want to hack Chaff?
  - Perhaps, but that does not prevent applying pre-processing first!
Our Experience

- Our DAC 2002 paper shows that Pre-processing + Chaff beats plain Chaff (or is very close) on realistic benchmarks.
- Symmetry extraction time is significant.
  - In some cases that limits competitiveness.
  - Sometimes there are very few symmetries.
    → Symmetry extraction isn’t very useful and therefore must be fast.
- There may be more room for improvement.
Mathematical Background(1)

- A symmetry (in a broad sense) of an object
  - Is a transformation that preserves its properties

- From Abstract Algebra
  - A group is a set with a binary operation on it
    - Must be associative
    - Must have a neutral element (unit)
    - Every element must have a (unique) inverse

- A subgroup is a subset closed under the op
  - Is a group by itself
Mathematical Background (2)

- The Lagrange Theorem
  - The size of a finite group is divisible by the size of its subgroups
  - Corollary: proper subgroups are \( \frac{1}{2} \) size or less

- A set of generators of a group
  - Every group element is a product of generators

- For a group of size \( N \), an irredundant set of generators has no more than \( \log_2 N \) elements
Felix Klein studied symmetries of geometric shapes in the XIX century. Group Theory was developed in XIX century as a formalism for capturing symmetries. It was used much earlier, e.g., by Galois. Today, it is one of the major branches of Mathematics. Symmetries are fundamental to modern physics, particularly Quantum Mechanics and Relativity, which deal with groups of symmetries. It seems natural to try using Group Theory in Computer Science!
More Definitions

- **Symmetries of a graph**
  - Permutations of vertices that preserve edges

- **Symmetries of a SAT formula**
  - Permutations of variables that preserve clauses
  - Simultaneous negations of sets of variables that preserve clauses
    - “phase-shift” symmetries (auto-symmetries)
  - Compositions of the two types

- Can talk about the symmetry group of
  (i) a graph, (ii) a CNF formula
Computational Group Theory (CGT)

- Finite (and some infinite!) groups routinely represented by generators
- The CGT was in the works since 1900s, and flourished since 1960s
  - Reasonably efficient algorithms for perm groups (Sims, Knuth, Babai, others)
  - Excellent implementations available today (GAP)
- Graph Automorphism programs (NAUTY)
Using Symmetries in SAT

1. Pre-processed CNF instance
2. Call generic SAT solver
3. CNF symmetries
4. Graph symmetries
5. Symmetry-breaking
6. CNF
7. Graph
8. CNF instance
A symmetry-breaking predicate (SBP) is what we add to a CNF formula to speed up DLL SAT solvers by pruning the search space. If a formula is satisfiable, a valid SBP must evaluate to TRUE on some SAT assignments. E.g., if $N$ truth assignments are symmetric, an SBP may pick only one of them. We allow “partial SBPs” that pick $>1$ solutions.
Symmetry-Breaking Predicates

Classes of symmetric truth assignments

SAT assignments
Previous Work (1)

  - CNF symmetries via Graph Automorphism
  - Full lex-leader SBPs from symmetries
    - Rather impractical *per se*, but of fundamental value
  - The concept of a symmetry tree
    - Not used in our work
  - A discussion of examples, several ideas we use
  - No convincing empirical results
How to Select Lex-leaders

- Idea: Select lexicographically smallest assignments from each equivalence class.
- Crawford et al. construct an SBP for that:
  - Map a given assignment by all symmetries.
  - Require that every image be lex-greater.

→ Conjunction over all symmetries 😞
Previous Work (2)

  - Improved/corrected use of Graph Automorphism
  - SBPs in terms of cycles of permutations
  - Partial SBPs via generators of symmetry groups
  - Strong, detailed empirical results
  - Fast “opportunistic” symmetry extraction
SBPs in cycle notation

- Suppose the variable $z$ can be negated
  - Then we can add the SBP $(z)$

- Suppose variables $x$ and $y$ can be swapped (with or w/o other variables being swapped)
  - Then we can add the SBP $(x \leq y)$, i.e., $(x'+y)$

- Similarly if $x$, $y$ and $z$ can be permuted
  - We add the SBP $(x \leq y \leq z)$, i.e., $(x'+y) (y'+z)$

- **Compared to Crawford et al., this is a form of logic minimization**
Contributions of This Work

- Further improvements of SBPs via logic minimization (used in VLSI CAD)
  - Economical SBPs $\rightarrow$ faster SAT-solving
  - Cases: single-cycles and multiple cycles
  - Another approach: direct improvement over Crawford

- New, provable analyses of partial symmetry-breaking by generators (PSBG)

- A pitfall identified: incompatible variable orderings

- PSBG for pigeon-holes is not complete
  - Yet, works extremely well in practice
New Constructions of SBPs For Single Permutations

- Since permutations are represented in the cycle notation, we look at single cycles first.
- Then we chain multiple cycles.
- Important observation
  - The variable ordering and the chaining sequence must be compatible.
  - This makes little difference for one permutation, but can spoil things for multiple permutations.
We show a counting formula for \#classes of symmetric assignments under an $N$-cycle.

1. First few numbers are: 3 (for the 2-cycle), 4, 6, 8

The straightforward generalization from small cycles $(xy)$ and $(xyz)$ to $(xyzt)$ does not yield a valid SBP!

However, one can explicitly formulate this as a two-level logic minimization problem:

1. Starting with a truth table, or
2. Starting with a CNF given by Crawford’s SBP.
New Constructions of SBPs For Single Cycles (2)

- We solve cycles of length $<20$ with ESPRESSO --- common software for two-level logic minimization
  - This gives a full lex-leader SBP for each $N$
  - We see some patterns, but no easy description
- We also propose a construction of partial lex-leader SBPs that works for any $N$
  - Can be used for very large cycles
SBPs for Multiple Cycles

- **Lemma**: SBPs of cycles of co-prime lengths in the same permutation can be conjoined
  
  - **Proof**: Each cycle is a power of the perm

- We give a more complex procedure for cycles whose lengths are not co-prime, e.g., \((xyz)(abcdef)\) or \((xyzt)(abcd)\)

- **Prime factors of the cycle length matter!**
Symmetry-breaking for Multiple Generators (1)

- Lemma 5.1 essentially says:
  Fully-breaking a given single symmetry is equivalent to fully breaking all of its powers

- Lemma 5.2:
  If a truth assignment is a lex-leader of an equivalence class under a group G, then it is a lex-leader ... under any subgroup of G
Symmetry-breaking for Multiple Generators (2)

- We now consider cyclic subgroups generated by each generator.
- Lemma 5.3: A conjunction of lex-leader SBPs of sub-groups is a valid SBP, however it may not be a full SBP.
- Corollary 5.6: Consider two perms with disjoint support. The conjunction of their lex-leader SBPs is a full lex-leader SBP.
On Variable Orderings

- When breaking symmetries by generators
  - especially efficient SBPs can be built for each generator by changing the order of variables
- However, variable orders must be consistent for all generators
- We build a consistency graph:
  - One vertex per generator
  - Connect generators whose supports intersect
- Find a maximal (or just large) independent set
SBPs for the Pigeon-hole Principle

- To show that not all symmetries are broken by a partial lex-leader SBP
  - We give a satisfying truth assignment that is not a lex-leader (*for hole-2*)
- Recall: all holes and all pigeons are symm.
  - The symmetry group is $S_n \times S_{n+1}$
  - Consider a set of generators that is a Cartesian product of those in $S_n$ and $S_{n+1}$
  - All generators map our assignment into > ones
Conclusions and On-going work

- Logic minimization leads to better SBPs
- Symmetry-breaking by generators is a sound and viable technique
  - Yet does not provide full symmetry-breaking in some important cases
- On-going work
  - Faster symmetry extraction
    - Generic and specialized, complete and incomplete
  - Further improvements of SBPs
  - Going beyond pre-processing