Symmetry in SAT: an overview

August 27th, 2018
Theory and Practice of SAT solving
Oaxaca, Mexico

Jo Devriendt, KU Leuven
Intro

Everyone knows symmetry:

"something does not change under a set of transformations"
- Wikipedia
In combinatorial solving

Symmetry :=
Permutation of variable assignments that preserves satisfaction
In combinatorial solving

Symmetry :=
Permutation of variable assignments that preserves satisfaction
In combinatorial solving

Symmetry induces symmetry classes:
In combinatorial solving

Symmetry induces symmetry classes:

"......calculating......"

Symmetry classes tend to get huge -> search problem
In combinatorial solving

Goal: investigate only asymmetrical cases
Contents

1. Intro
2. SAT Prelims
3. "Classic" symmetry breaking
4. The pigeonhole problem
5. "Recent" symmetry breaking
6. Non-breaking approaches
7. Bonus: symmetry, local search & maxSAT
1. Intro
2. SAT Prelims
3. "Classic" symmetry breaking
4. The pigeonhole problem
5. "Recent" symmetry breaking
6. Non-breaking approaches
7. Bonus: symmetry, local search & maxSAT

"Interesting research question"
In SAT:

**Syntactic** symmetry :=

literal permutation that preserves the CNF

\[
\begin{align*}
a \lor \neg b & \quad \rightarrow \quad b \\
\neg a & \quad \rightarrow \quad \neg b \\
b \lor \neg c & \quad \rightarrow \quad c \\
\neg b & \quad \rightarrow \quad \neg c \\
c \lor \neg a & \quad \rightarrow \quad a \\
\neg c & \quad \rightarrow \quad \neg a
\end{align*}
\]
In SAT:

**Syntactic symmetry** := literal permutation that preserves the CNF

\[
\begin{align*}
    a & \mapsto b \\
    \neg a & \mapsto \neg b \\
    b & \mapsto c \\
    \neg b & \mapsto \neg c \\
    c & \mapsto a \\
    \neg c & \mapsto \neg a
\end{align*}
\]

\((a \lor \neg b) \lor (b \lor \neg c) \lor (c \lor \neg a)\)
In SAT literature:

- Shatter
- BreakID
- CDCLSym
- Adaptive prefix-assignment
- SymChaff
- Symmetric learning
Terminology

- variable $x$
  - set of all variables $X$
- literal $l$
- clause $c$
- (propositional) formula $\varphi$
- (variable) assignment $\alpha$
  - $\alpha(l)$ is the truth value of $l$ in $\alpha$
- symmetry $\sigma$
  - $\sigma(...)$ is the symmetrical image of $...$
- symmetry group $\Sigma$
  - $\Sigma(...)$ is the orbit of $...$ under $\Sigma$
  - generator set $\text{gen}(\Sigma)$
3. "Classic" symmetry breaking
Symmetry breaking formulas: Crawford [1]

Given: \( \varphi, \Sigma \)
Find: symmetry breaking formula \( \text{sbf} \)
that invalidates symmetrical assignments
Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes "\(\alpha\) is lexicographically smaller than \(\sigma(\alpha)\)" for \(\sigma \in \Sigma\)
Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes "α is lexicographically smaller than σ(α)"
for σ ∈ Σ

\[ x_1 \leq \sigma(x_1) \]
\[ x_1 = \sigma(x_1) \Rightarrow x_2 \leq \sigma(x_2) \]
\[ (x_1 = \sigma(x_1) \land x_2 = \sigma(x_2)) \Rightarrow x_3 \leq \sigma(x_3) \]

...
Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes "α is lexicographically smaller than σ(α)" for σ ∈ Σ

\[ x_1 \leq \sigma(x_1) \]

\[ x_1 = \sigma(x_1) \Rightarrow x_2 \leq \sigma(x_2) \]

\[ (x_1 = \sigma(x_1) \land x_2 = \sigma(x_2)) \Rightarrow x_3 \leq \sigma(x_3) \]

\[ \ldots \]

parameter: total order on X
Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes
"α is lexicographically smaller than σ(α)"
for all σ ∈ Σ

\( \varphi \cup \text{sbf} \)
Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes "α is lexicographically smaller than σ(α)"
for all σ ∈ Σ

- Sound
- Complete
- Huge: Ω(|X|^2 |Σ|)

φ ∪ sbf
Symmetry breaking: Shatter [2]

- construct sbf for -much smaller- gen(\(\Sigma\))
- "chain encoding"
- improved clausal encoding
Symmetry breaking: Shatter [2]

- construct sbf for -much smaller- gen(Σ)
- "chain encoding"
- improved clausal encoding

Sound ✓
Incomplete ✗
Feasible: $O(|X| |\text{gen}(\Sigma)|)$
Detecting symmetry: Saucy [3]

Sparse graph automorphism detection
Detecting symmetry: Saucy [3]

Sparse graph automorphism detection

- Graph construction from CNF:
  - node for each literal and clause
  - edge between literals occurring in clause
  - edge between literal and negation
- No polynomial algorithm known
- Output: generators to automorphism group
Static symmetry breaking: Shatter+Saucy

Propositional description

\[ a \lor \neg b \]
\[ b \lor \neg c \]
\[ c \lor \neg a \]

Graph automorphism detection

Add symmetry breaking formulas

\[ \neg a \lor b \]

SAT/UNSAT

off-the-shelf SAT solver
4. The pigeonhole problem
Pigeonhole!

Do n pigeons fit in n-1 holes?

\[ \forall p: \bigvee_h x_{ph} \]
\[ \forall h: \forall p \neq p': \neg x_{ph} \lor \neg x_{p'h} \]
Do n pigeons fit in n-1 holes?

\[ \forall p: \bigvee_h x_{ph} \]

\[ \forall h: \forall p \neq p': \neg x_{ph} \lor \neg x_{p'h} \]

- Proof-theoretic problem
- Simple but large symmetry group
  - composition of "pigeon interchangeability" and "hole interchangeability"
  - 1 symmetry class
### Original Shatter experiment:

<table>
<thead>
<tr>
<th>Benchmark Family</th>
<th>Instance</th>
<th># Generators</th>
<th># Generators &amp; their compositions</th>
<th>Time to find symmetries (sec)</th>
<th>Time to solve orig. instance (sec)</th>
<th>Time to solve instances and SBPs (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Generators only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>All Bits</td>
</tr>
<tr>
<td>Hole-n</td>
<td>hole07</td>
<td>13</td>
<td>102</td>
<td>0.00</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>hole08</td>
<td>15</td>
<td>133</td>
<td>0.00</td>
<td>0.15</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>hole09</td>
<td>17</td>
<td>168</td>
<td>0.01</td>
<td>0.97</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>hole10</td>
<td>19</td>
<td>207</td>
<td>0.02</td>
<td>14.4</td>
<td>2.87</td>
</tr>
<tr>
<td></td>
<td>hole11</td>
<td>21</td>
<td>250</td>
<td>0.02</td>
<td>133</td>
<td>9.04</td>
</tr>
<tr>
<td></td>
<td>hole12</td>
<td>23</td>
<td>297</td>
<td>0.02</td>
<td>&gt;1000</td>
<td>6.90</td>
</tr>
</tbody>
</table>

- **Generators only**
- **Irredundant Bits**
- **Quadratic construction**
- **Linear construction**
Pigeonhole!

Original Shatter experiment:

<table>
<thead>
<tr>
<th>Benchmark Family</th>
<th>Instance</th>
<th># Generators</th>
<th># Generators &amp; their compositions</th>
<th>Time to find symmetries (sec)</th>
<th>Time to solve orig. instance (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hole-n</td>
<td>hole07</td>
<td>13</td>
<td>102</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>hole08</td>
<td>15</td>
<td>133</td>
<td>0.00</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>hole09</td>
<td>17</td>
<td>168</td>
<td>0.01</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>hole10</td>
<td>19</td>
<td>207</td>
<td>0.02</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>hole11</td>
<td>21</td>
<td>250</td>
<td>0.02</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>hole12</td>
<td>23</td>
<td>297</td>
<td>0.02</td>
<td>&gt;1000</td>
</tr>
</tbody>
</table>

|                  |          |              | Time to solve instances and SBPs (sec) |
|                  |          |              | Generators only | Generators & their compositions |
|                  |          |              | All Bits | Irredundant Bits | Quadratic construction | Linear construction | Quadratic construction | Linear construction |
|                  |          |              |          |                |                           |                           |          |                |                           |                           |
|                  |          |              | 0.03     | 0.01            |                           |                           |          |                |                           |                           |
|                  |          |              | 0.17     | 0.01            |                           |                           |          |                |                           |                           |
|                  |          |              | 0.30     | 0.01            |                           |                           |          |                |                           |                           |
|                  |          |              | 2.87     | 0.01            |                           |                           |          |                |                           |                           |
|                  |          |              | 9.04     | 0.01            |                           |                           |          |                |                           |                           |
|                  |          |              | 6.90     | 0.01            |                           |                           |          |                |                           |                           |

Seems ok?
Pigeonhole!

Own Shatter experiment:

- **glucose**
- **glucose+shatter**
Own Shatter experiment:

Modest gains...
Better results in original paper?
Pigeonhole!

- Propositional encoding reduces "pigeon interchangeability" to "row interchangeability"
- Shatter's sbf's are complete [4] with correct choice of
  - \( \text{gen}(\Sigma) \)
  - variable order

\[
\begin{array}{ccc}
  x_{11} & x_{12} & x_{13} \\
  x_{21} & x_{22} & x_{23} \\
  x_{31} & x_{32} & x_{33} \\
  x_{41} & x_{42} & x_{43} \\
\end{array}
\]

- \( |\text{full sbf}| = O(n^2) \)
Propositional encoding reduces "pigeon interchangeability" to "row interchangeability"

Shatter's sbf's are complete [4] with correct choice of
- $\text{gen}(\Sigma)$
- variable order

$|\text{full sbf}| = O(n^2)$
5. "Recent" symmetry breaking
Symmetry breaking: BreakID [5]

Core idea: detect "row swap" symmetries

*Approximative algorithm
Symmetry breaking: BreakID [5]

Core idea: detect "row swap" symmetries

*Approximative algorithm

1. Search $\sigma_1, \sigma_2 \in \text{gen}(\Sigma)$ that form 2 subsequent row swaps
   - forms initial 3-rowed variable matrix $M$
Symmetry breaking: BreakID [5]

Core idea: detect "row swap" symmetries

*Approximative algorithm

1. Search $\sigma_1, \sigma_2 \in \text{gen}(\Sigma)$ that form 2 subsequent row swaps
   - forms initial **3-rowed variable matrix** $M$
2. Apply every $\sigma \in \text{gen}(\Sigma)$ to all detected rows $r \in M$ so far
   - images $\sigma(r)$ disjoint of $M$ are candidates to extend $M$
   - test if swap $r \leftrightarrow \sigma(r)$ is a symmetry by syntactical check on $\varphi$
   - if success, **extend $M$ with $\sigma(r)$**
Symmetry breaking: BreakID [5]

Core idea: detect "row swap" symmetries

*Approximative algorithm

1. Search $\sigma_1, \sigma_2 \in \text{gen}(\Sigma)$ that form 2 subsequent row swaps
   
   - forms initial 3-rowed variable matrix $M$

2. Apply every $\sigma \in \text{gen}(\Sigma)$ to all detected rows $r \in M$ so far
   
   - images $\sigma(r)$ disjoint of $M$ are candidates to extend $M$
   - test if swap $r \leftrightarrow \sigma(r)$ is a symmetry by syntactical check on $\varphi$
   - if success, extend $M$ with $\sigma(r)$

3. Use Saucy to extend $\text{gen}(\Sigma)$ with new symmetry generators by fixing all variable nodes with variable in $M$, first row excepted
Symmetry breaking: BreakID [5]

Core idea: detect "row swap" symmetries

Caveat!
Symmetry breaking: BreakID [5]

Core idea: detect "row swap" symmetries

Caveat!

Detect row interchangeability subgroups?
Symmetry breaking: BreakID [5]

Core idea: maximize number of binary sbf clauses
Symmetry breaking: BreakID [5]

Core idea: maximize number of **binary sbf clauses**

- First clause in sbf for $\sigma$ is binary:
  \[ \neg x_1 \lor \sigma(x_1) \]

- $x$ is **stabilized** by $\Sigma$ iff $\Sigma(x) = \{x\}$

- Given $\Sigma$ with **smallest non-stabilized** $x$, for each $x' \in \Sigma(x)$:
  \[ \neg x \lor x' \]

  is clause of sbf under some $\sigma \in \Sigma$
Symmetry breaking: BreakID [5]

Core idea: exploit binary sbf clauses
Symmetry breaking: BreakID [5]

Core idea: exploit **binary sbf clauses**

- Create **stabilizer chain** of $\Sigma$:
  \[
  \Sigma \supset \Sigma_1 \supset \Sigma_2 \supset \ldots \supset 1
  \]

- $\Sigma_i$ is the **stabilizer subgroup** of $\Sigma_{i-1}$ stabilizing the next non-stabilized variable in ordering
  - $\Sigma_i$ have different smallest non-stabilized variables $x_i$

- For each $x' \in \Sigma_i(x_i)$:
  \[
  \neg x_i \lor x'
  \]

is a clause of some sbf
Symmetry breaking: BreakID [5]

Core idea: exploit binary sbf clauses
Symmetry breaking: BreakID [5]

Core idea: exploit *binary sbf clauses*

- Approximative implementation
  - which adapts the variable order!
Symmetry breaking: BreakID [5]

Core idea: exploit binary sbf clauses

- Approximative implementation
  - which adapts the variable order!

- Works well for polarity symmetry \( \sigma \) where for all \( x \):
  \[
  \sigma(x) = \neg x
  \]
  as sbf is equivalent to unit clause
  \[
  \neg x_1
  \]
  and their number is maximized through adopted variable order.
Symmetry breaking: BreakID [5]

Core idea: exploit binary sbf clauses

• Approximative implementation
  ▪ which adapts the variable order!

• Works well for polarity symmetry $\sigma$ where for all $x$:
  \[ \sigma(x) = \neg x \]
  as sbf is equivalent to unit clause
  \[ \neg x_1 \]
  and their number is maximized through adopted variable order.

Complete algorithm?
Symmetry breaking: CDCLSym [6]

Core idea: generate sbf dynamically
Symmetry breaking: CDCLSym [6]

Core idea: generate sbf dynamically

- Keep track of **reducer** symmetries where $\sigma(\alpha) < \alpha$
  - by watching smallest variable s.t. $\sigma(v) \neq v$
- **Generate clause** from sbf forcing $\alpha \leq \sigma(\alpha)$

Additionally: try Bliss instead of Saucy
Symmetry breaking: CDCLSym [6]

Core idea: generate sbf dynamically

- Keep track of **reducer** symmetries where $\sigma(\alpha) < \alpha$
  - by watching smallest variable s.t. $\sigma(v) \neq v$
- **Generate clause** from sbf forcing $\alpha \leq \sigma(\alpha)$

Additionally: try Bliss instead of Saucy

Use clauses for propagation?
Not only generator symmetries?
Symmetry breaking: On completeness

- Pigeon *interchangeability* can be completely broken with polynomial sbf
Symmetry breaking: On completeness

- Pigeon **interchangeability** can be completely broken with polynomial sbf
- How about **edge interchangeability**?
  - E.g., find coloring of complete graph (Ramsey numbers)
  - Recent interest [11] [14]
Symmetry breaking: On completeness

- Pigeon **interchangeability** can be completely broken with polynomial sbf
- How about **edge interchangeability**?
  - E.g., find coloring of complete graph (Ramsey numbers)
  - Recent interest [11] [14]
- How about **general interchangeability** over arbitrary high dimensional relations?
Symmetry breaking: On completeness

- Pigeon **interchangeability** can be completely broken with polynomial sbf
- How about **edge interchangeability**?
  - E.g., find coloring of complete graph (Ramsey numbers)
  - Recent interest [11] [14]
- How about **general interchangeability** over arbitrary high dimensional relations?

Tractable sbf for edge interchangeability?
Symmetry breaking: Prefix breaking [7]

Core idea: enumerate asymmetrical assignments to variable prefix

∀x: ∃y: \( \varphi(x, y) \)
Symmetry breaking: Prefix breaking [7]

Core idea: enumerate asymmetrical assignments to variable prefix

∀x: ∃y: ϕ(x, y)
6. Non-breaking approaches
Symmetry handling: SymChaff [8]

Core idea: search decisions consider row interchangeability

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
</tr>
</tbody>
</table>
Symmetry handling: SymChaff [8]

Core idea: search decisions consider row interchangeability

- Only for row interchangeability symmetry
- Keep track of row-interchangeable variables
  - interchangeability reduces under previous choices
- Use **cardinality decision points** over row-interchangeable variables

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
</tr>
</tbody>
</table>
Symmetry handling: SymChaff [8]

Core idea: search decisions consider row interchangeability

- Only for row interchangeability symmetry
- Keep track of row-interchangeable variables
  - interchangeability reduces under previous choices
- Use **cardinality decision points** over row-interchangeable variables

Cardinality decision of 1 over first column:

<table>
<thead>
<tr>
<th></th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td></td>
</tr>
<tr>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
<td></td>
</tr>
<tr>
<td>$x_{41}$</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
<td></td>
</tr>
</tbody>
</table>
Symmetry handling: SymChaff [8]

Core idea: search decisions consider row interchangeability

- Only for row interchangeability symmetry
- Keep track of row-interchangeable variables
  - interchangeability reduces under previous choices
- Use **cardinality decision points** over row-interchangeable variables

Cardinality decision of 1 over first column:

<table>
<thead>
<tr>
<th></th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
</tr>
<tr>
<td>0</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
</tr>
<tr>
<td>1</td>
<td>$x_{42}$</td>
<td>$x_{43}$</td>
</tr>
</tbody>
</table>
Symmetry handling: SymChaff [8]

Strong performance on pigeonhole

<table>
<thead>
<tr>
<th>Problem</th>
<th>SymChaff</th>
</tr>
</thead>
<tbody>
<tr>
<td>009-008</td>
<td>0.01</td>
</tr>
<tr>
<td>013-012</td>
<td>0.01</td>
</tr>
<tr>
<td>051-050</td>
<td>0.24</td>
</tr>
<tr>
<td>091-090</td>
<td>0.84</td>
</tr>
<tr>
<td>101-100</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Symmetry handling: Symmetric learning [9]

Core idea: consider symmetrical learned clauses
Symmetry handling: Symmetric learning [9]

Core idea: consider symmetrical learned clauses

- Learnt clauses are **logical consequences** of \( \varphi \)
- Whenever \( c \) is a consequence of \( \varphi \), so is \( \sigma(c) \)
- Problem: \( \Sigma(c) \) is huge
  - Learn only \( \sigma(c) \) for \( \sigma \in \text{gen}(\Sigma) \)
Symmetry handling: Symmetric learning [9]

Core idea: consider symmetrical learned clauses

- Learnt clauses are **logical consequences** of $\varphi$
- Whenever $c$ is a consequence of $\varphi$, so is $\sigma(c)$
- Problem: $\Sigma(c)$ is huge
  - Learn only $\sigma(c)$ for $\sigma \in \text{gen}(\Sigma)$

![Symmetric learning graph](#)
Symmetry handling:
Symmetric explanation learning [10]

Core idea: consider useful symmetrical explanation clauses
Symmetry handling: Symmetric explanation learning [10]

Core idea: consider useful symmetrical explanation clauses

- Learn $\sigma(c)$ that **propagate at least once**
  - symmetries typically permute only a **fraction** of the literals
  - if $c$ is unit, $\sigma(c)$ has a good chance of being unit as well
  - explanation clauses are unit ;-)
Symmetry handling:
Symmetric explanation learning [10]

Core idea: consider useful symmetrical explanation clauses
Symmetry handling: Symmetric explanation learning [10]

Core idea: consider useful symmetrical explanation clauses

- For each $\sigma \in \text{gen}(\Sigma)$, whenever $c$ propagates, store $\sigma(c)$ in a separate clause store $\theta$
  - Propagation on $\theta$ happens only if standard unit propagation is at a fixpoint
  - Whenever a $\sigma(c) \in \theta$ propagates, upgrade it to a "normal" learned clause
  - After backjump over $c$'s propagation level, clear $\sigma(c)$ from $\theta$
Symmetry handling: Symmetric explanation learning [10]

Core idea: consider useful symmetrical explanation clauses

- For each $\sigma \in \text{gen}(\Sigma)$, whenever $c$ propagates, store $\sigma(c)$ in a separate clause store $\theta$
  - Propagation on $\theta$ happens only if standard unit propagation is at a fixpoint
  - Whenever a $\sigma(c) \in \theta$ propagates, upgrade it to a "normal" learned clause
  - After backjump over $c$'s propagation level, clear $\sigma(c)$ from $\theta$
- Every learned $\sigma(c)$ is useful & original
- Transitive effect: track $\sigma'(\sigma(c))$ when $\sigma(c)$ propagates
Symmetry handling: Symmetric explanation learning [10]
Symmetry handling: Symmetric explanation learning [10]

Caveat: performance on larger instances
Symmetry handling: Symmetric explanation learning [10]

Caveat: performance on larger instances

What is "complete" symmetrical learning? Can it be done efficiently?
Research trends:

- Symmetry detection on propositional level is hard
  - not a solved problem, cfr. pigeonhole
  - papers often assume **high-level symmetry input** [7] [8]
- Sbf construction based on **canonical labeling** [7] [11]
- Dynamical approaches often perform **lazy clause generation** [6] [10] [12]
- Use computational group algebra to detect symmetry **group structure** [5] [13]

Proof checking and symmetrical learning?
The influence of the variable order on an sbf?
Thanks for listening!
Questions?

7. Bonus: symmetry, local search & maxSAT
Bonus: symmetry, local search & maxSAT

- (Satisfying) assignments now have an associated **score**
Bonus: symmetry, local search & maxSAT

- (Satisfying) assignments now have an associated **score**
- Local search "**moves**" from one to the other based on **structure-preserving transformations**
Bonus: symmetry, local search & maxSAT

- (Satisfying) assignments now have an associated **score**
- Local search "**moves**" from one to the other based on **structure-preserving transformations**
- Designing local moves is typically done **by hand**...
Bonus: symmetry, local search & maxSAT

- (Satisfying) assignments now have an associated score
- Local search "moves" from one to the other based on structure-preserving transformations
- Designing local moves is typically done by hand...

Symmetries form moves!
Can be automatically detected!
Bonus: symmetry, local search & maxSAT

Scatter plot of objective value of knapsack instances (higher is better)
Bonus: symmetry, local search & maxSAT

Scatter plot of objective value of knapsack instances (higher is better)

Symmetry-based local search in weighted maxSAT?