Symmetry in SAT: an overview

August 27th, 2018

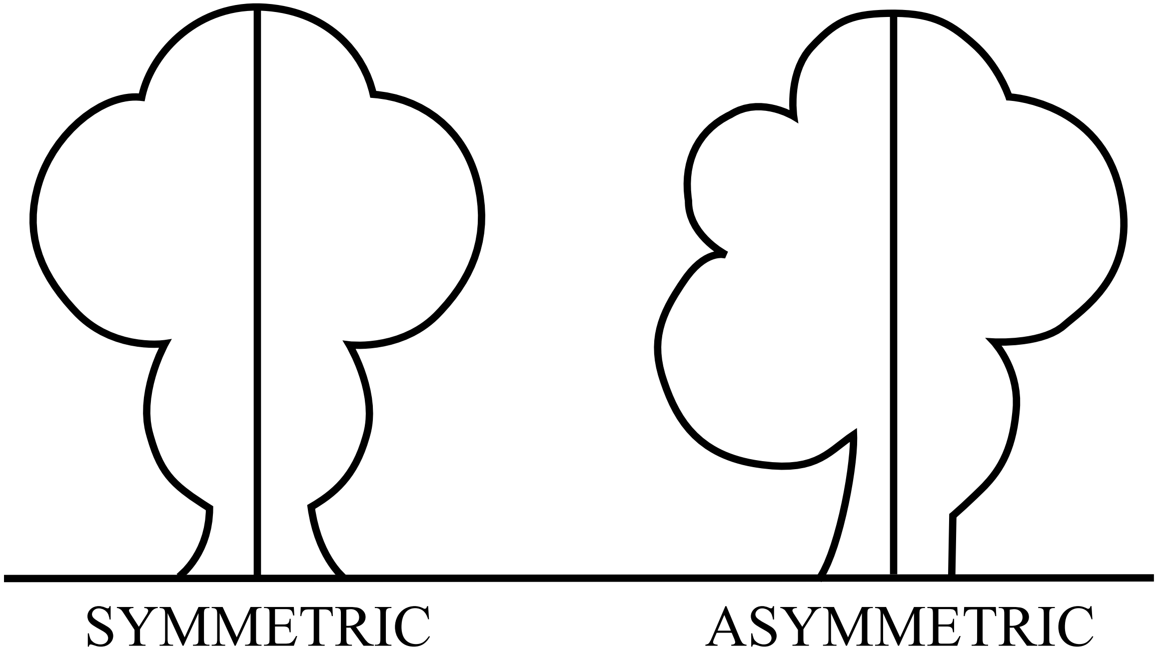
Theory and Practice of SAT solving

Oaxaca, Mexico

Jo Devriendt, KU Leuven

Intro

Everyone knows symmetry:

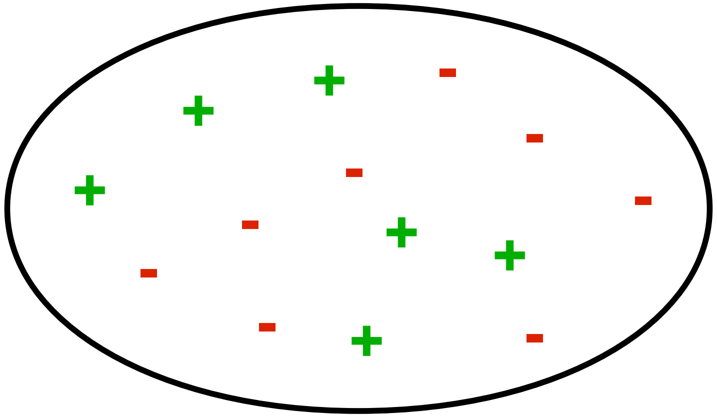


"something does not change under a set of transformations"
- Wikipedia

In combinatorial solving

Symmetry :=

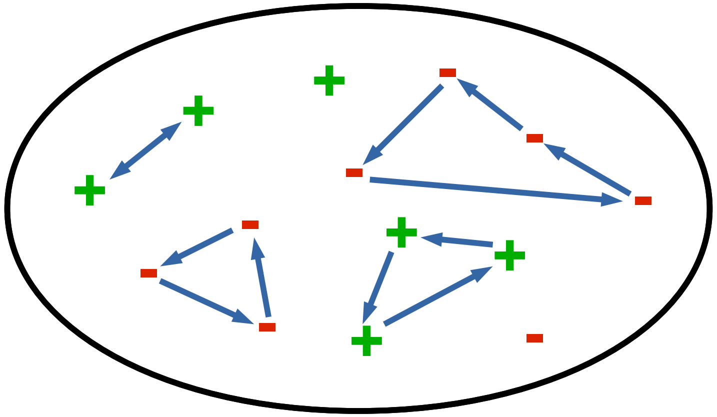
Permutation of variable assignments that preserves satisfaction



In combinatorial solving

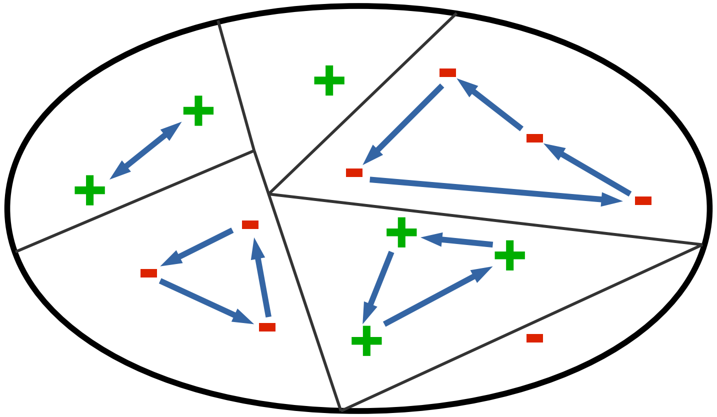
Symmetry :=

Permutation of variable assignments that preserves satisfaction



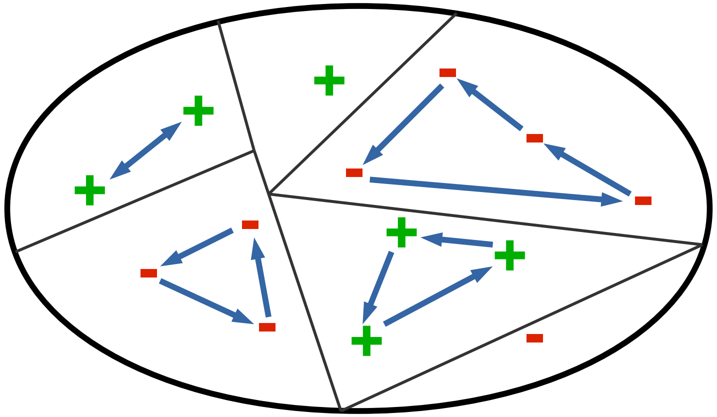
In combinatorial solving

Symmetry induces symmetry classes:

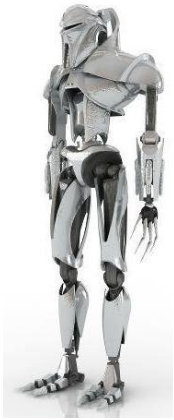


In combinatorial solving

Symmetry induces symmetry classes:



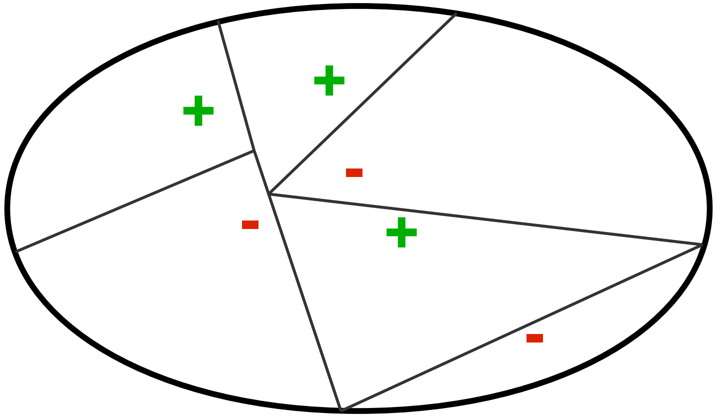
"......calculating......"



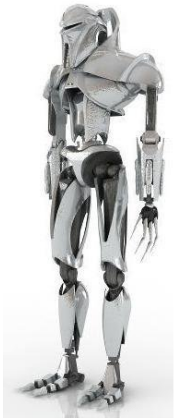
Symmetry classes tend to get huge -> search problem

In combinatorial solving

Goal: investigate only asymmetrical cases



!!



Contents

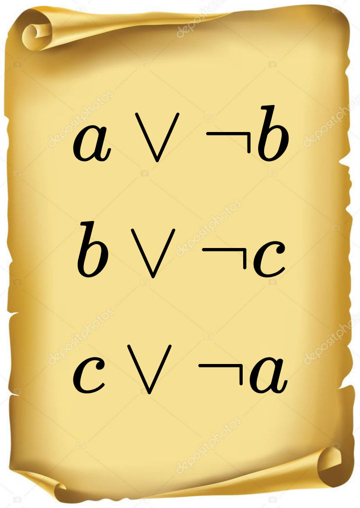
1. Intro
2. SAT Prelims
3. "Classic" symmetry breaking
4. The pigeonhole problem
5. "Recent" symmetry breaking
6. Non-breaking approaches
7. Bonus: symmetry, local search & maxSAT

Contents

"Interesting research question"

In SAT:

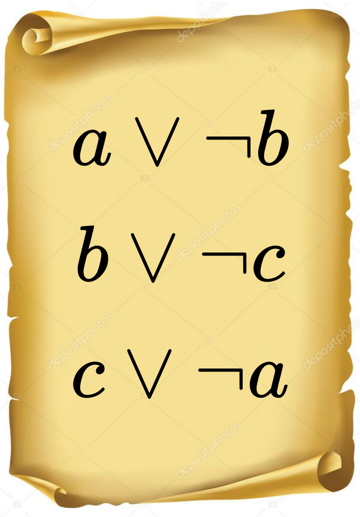
Syntactic symmetry :=
literal permutation that preserves the CNF



$$a \mapsto b$$
$$\neg a \mapsto \neg b$$
$$b \mapsto c$$
$$\neg b \mapsto \neg c$$
$$c \mapsto a$$
$$\neg c \mapsto \neg a$$

In SAT:

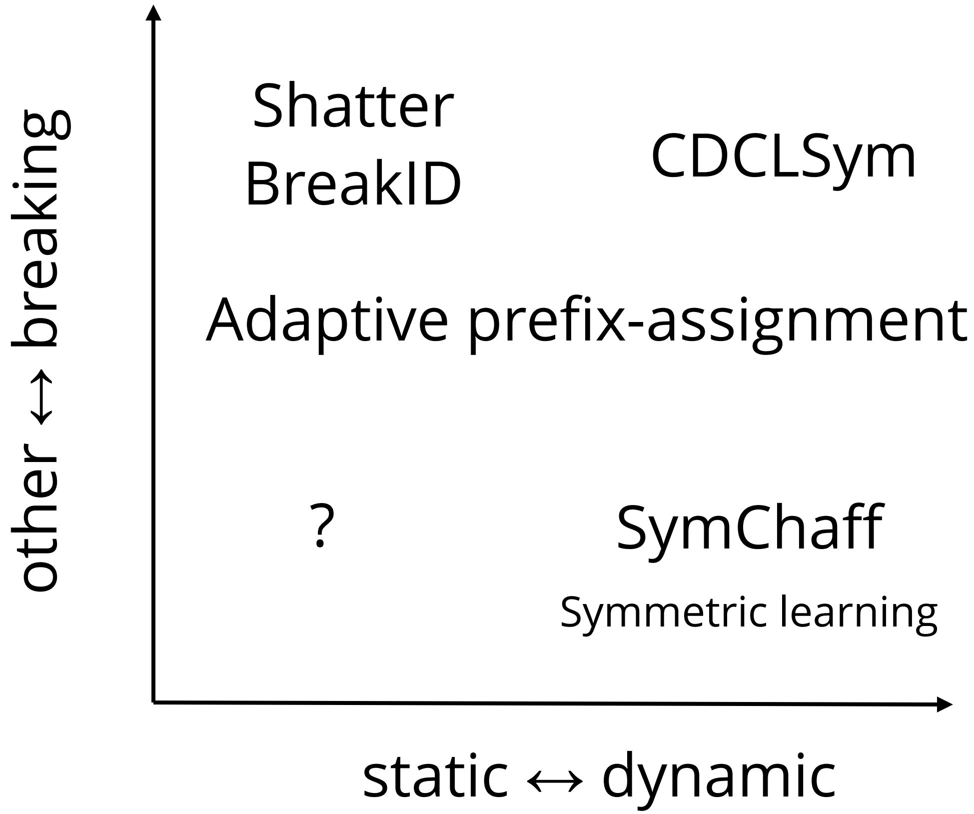
Syntactic symmetry :=
literal permutation that preserves the CNF



$$
\begin{aligned}
a &\mapsto b \\
\neg a &\mapsto \neg b \\
b &\mapsto c \\
\neg b &\mapsto \neg c \\
c &\mapsto a \\
\neg c &\mapsto \neg a
\end{aligned}
$$

$$(a\ b\ c)$$

In SAT literature:



Terminology

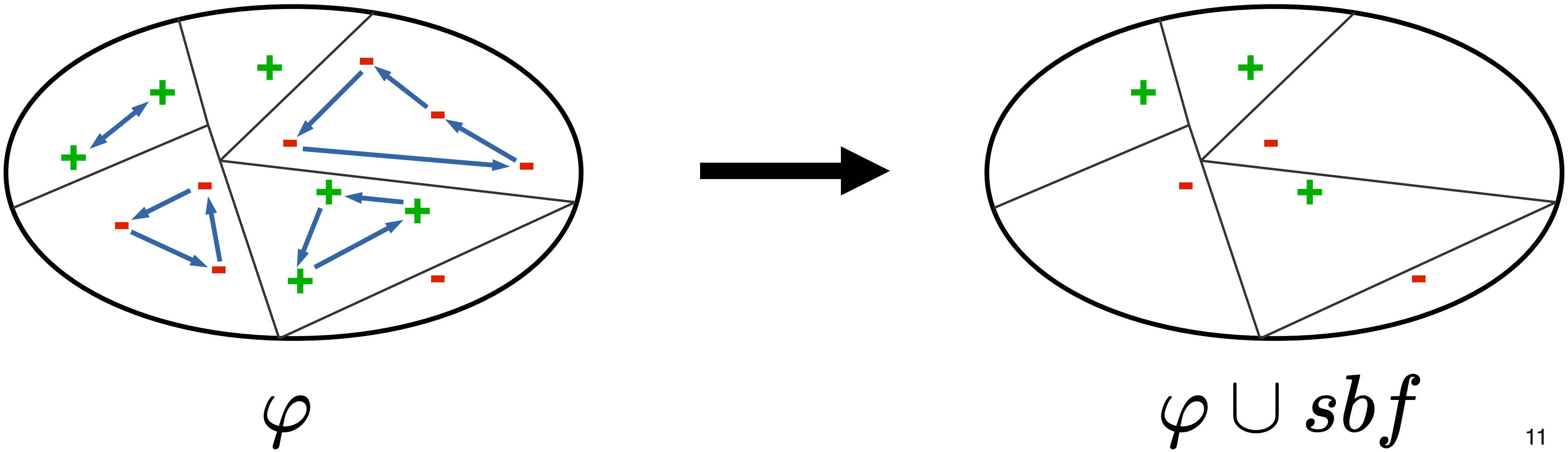
- variable ***x***
  - set of all variables ***X***
- literal ***l***
- clause ***c***
- (propositional) formula $\boldsymbol{\varphi}$
- (variable) assignment $\boldsymbol{\alpha}$
  - $\alpha(l)$ is the truth value of l in α
- symmetry $\boldsymbol{\sigma}$
  - $\sigma(\ldots)$ is the symmetrical image of ...
- symmetry group $\boldsymbol{\Sigma}$
  - $\Sigma(\ldots)$ is the orbit of ... under Σ
  - generator set $\boldsymbol{gen(\Sigma)}$

3. "Classic" symmetry breaking

Symmetry breaking formulas: Crawford [1]

Given: φ, Σ

Find: symmetry breaking formula ***sbf***
that invalidates symmetrical assignments



Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes
"α 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 ϵ Σ

$$x_1 \leq \sigma(x_1)$$

$$x_1 = \sigma(x_1) \Rightarrow x_2 \leq \sigma(x_2)$$

$$(x_1 = \sigma(x_1) \wedge x_2 = \sigma(x_2)) \Rightarrow x_3 \leq \sigma(x_3)$$

$$\ldots$$

Symmetry breaking formulas: Crawford [1]

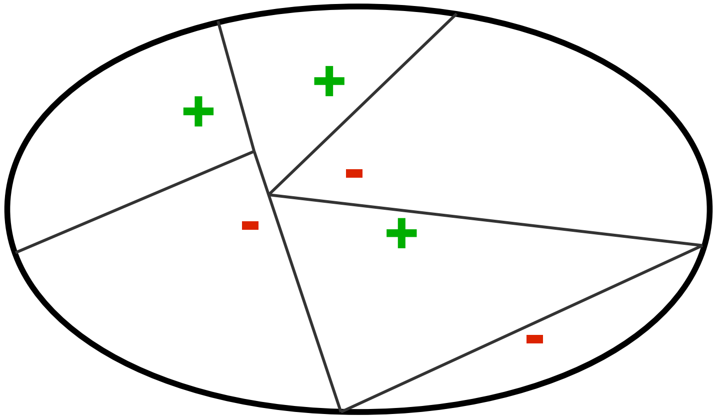
Core idea: sbf encodes
"α is lexicographically smaller than σ(α)"
for σ ϵ Σ

$$x_1 \leq \sigma(x_1)$$
$$x_1 = \sigma(x_1) \Rightarrow x_2 \leq \sigma(x_2)$$
$$(x_1 = \sigma(x_1) \wedge x_2 = \sigma(x_2)) \Rightarrow x_3 \leq \sigma(x_3)$$
$$\dots$$

parameter: total order on X

Symmetry breaking formulas: Crawford [1]

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

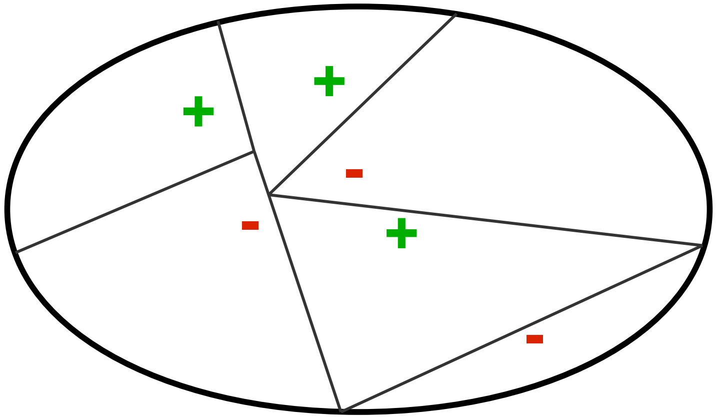


$$\varphi \cup sbf$$

Symmetry breaking formulas: Crawford [1]

Core idea: sbf encodes
"α is lexicographically smaller than σ(α)"
for **all** $\sigma \in \Sigma$

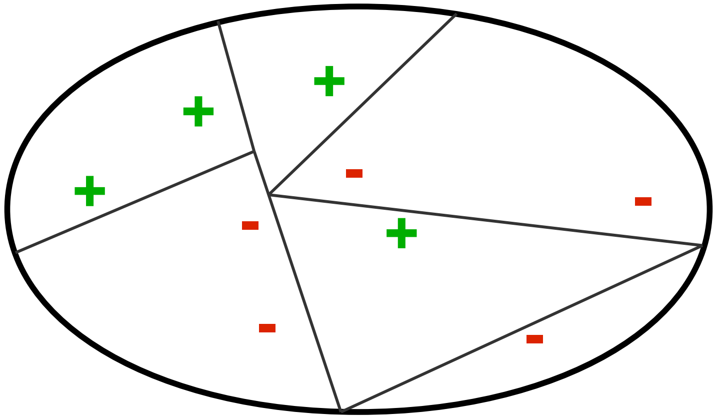
- Sound
- Complete
- Huge: $O(|X|^2|\Sigma|)$



$$\varphi \cup sbf$$

Symmetry breaking: Shatter [2]

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

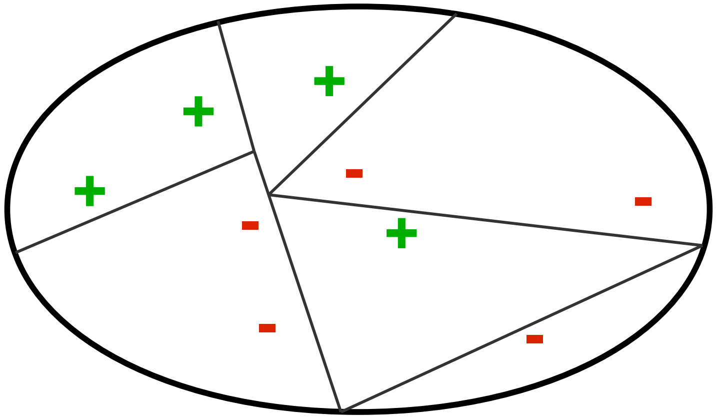


$$\varphi \cup \boldsymbol{sbf}$$

Symmetry breaking: Shatter [2]

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

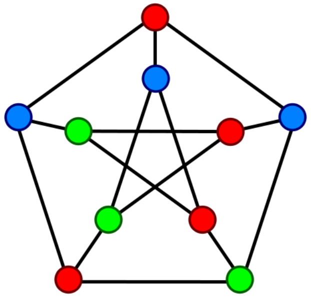
- ✓ Sound
- ✗ Incomplete
- ✓ Feasible: O(|X||gen(Σ)|)



$\varphi \cup sbf$

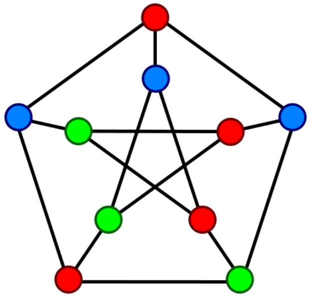
Detecting symmetry: Saucy [3]

Sparse graph automorphism detection



Detecting symmetry: Saucy [3]

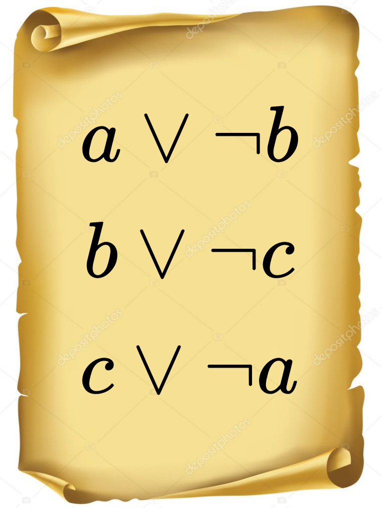
Sparse graph automorphism detection



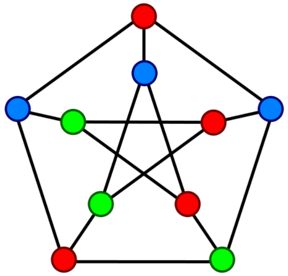
- Graph construction from CNF:
  - node for each literal and clause
  - edge between literals occurring in clause
  - edge between literal and negation
- No polynomial algorithm known
- Output: generators to automorphism group

Static symmetry breaking: Shatter+Saucy

Propositional description



Graph automorphism detection



$(a\ b\ c)$

Add symmetry breaking formulas

$\neg a \vee b$

off-the-shelf SAT solver

SAT/UNSAT

4. The pigeonhole problem

Pigeonhole!

Do n pigeons fit in n-1 holes?

$$\forall p: \bigvee_h x_{ph}$$

$$\forall h: \forall p \neq p': \neg x_{ph} \vee \neg x_{p'h}$$

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

Pigeonhole!

Do n pigeons fit in n-1 holes?

$$\forall p\colon\ \bigvee_h x_{ph}$$
$$\forall h\colon \forall p \neq p'\colon \neg x_{ph} \vee \neg x_{p'h}$$

- Proof-theoretic problem
- Simple but large symmetry group
  - composition of "pigeon interchangeability" and "hole interchangeability"
  - 1 symmetry class

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

Pigeonhole!

Original Shatter experiment:

Bench-mark Family	Instance	# Generators	# Generators & their compositions	Time to find symmetries (sec)	Time to solve orig. instance (sec)	Time to solve instances and SBPs (sec)			
						Generators only			Generators & their com-positions
						All Bits	Irredundant Bits		
						Quadratic construction		Linear construction	
Hole-n	hole07	13	102	0.00	0.03	0.03	0.01	0.01	0.01
	hole08	15	133	0.00	0.15	0.17	0.01	0.01	0.01
	hole09	17	168	0.01	0.97	0.30	0.01	0.01	0.01
	hole10	19	207	0.02	14.4	2.87	0.01	0.01	0.01
	hole11	21	250	0.02	133	9.04	0.01	0.01	0.02
	hole12	23	297	0.02	>1000	6.90	0.01	0.01	0.03

Pigeonhole!

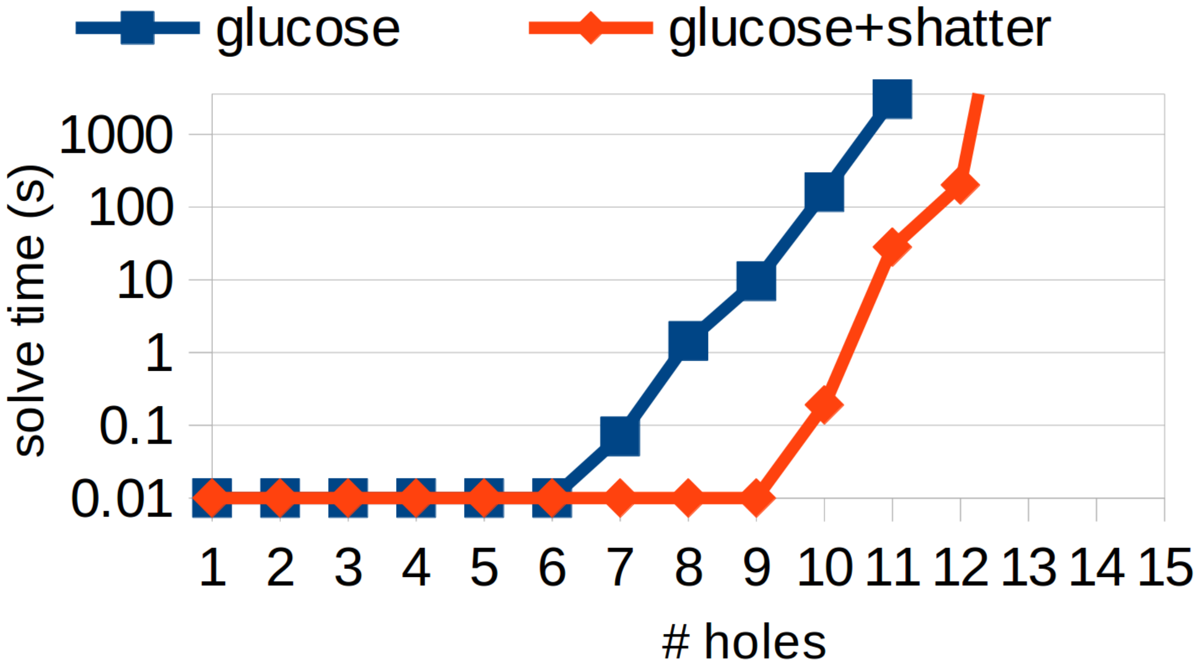
Original Shatter experiment:

Benchmark Family	Instance	# Generators	# Generators & their compositions	Time to find symmetries (sec)	Time to solve orig. instance (sec)	Time to solve instances and SBPs (sec): Generators only, All Bits, Quadratic construction	Time to solve instances and SBPs (sec): Generators only, Irredundant Bits, Quadratic construction	Time to solve instances and SBPs (sec): Generators only, Irredundant Bits, Linear construction	Time to solve instances and SBPs (sec): Generators & their compositions, Irredundant Bits, Linear construction
Hole-n	hole07	13	102	0.00	0.03	0.03	0.01	0.01	0.01
	hole08	15	133	0.00	0.15	0.17	0.01	0.01	0.01
	hole09	17	168	0.01	0.97	0.30	0.01	0.01	0.01
	hole10	19	207	0.02	14.4	2.87	0.01	0.01	0.01
	hole11	21	250	0.02	133	9.04	0.01	0.01	0.02
	hole12	23	297	0.02	>1000	6.90	0.01	0.01	0.03

Seems ok?

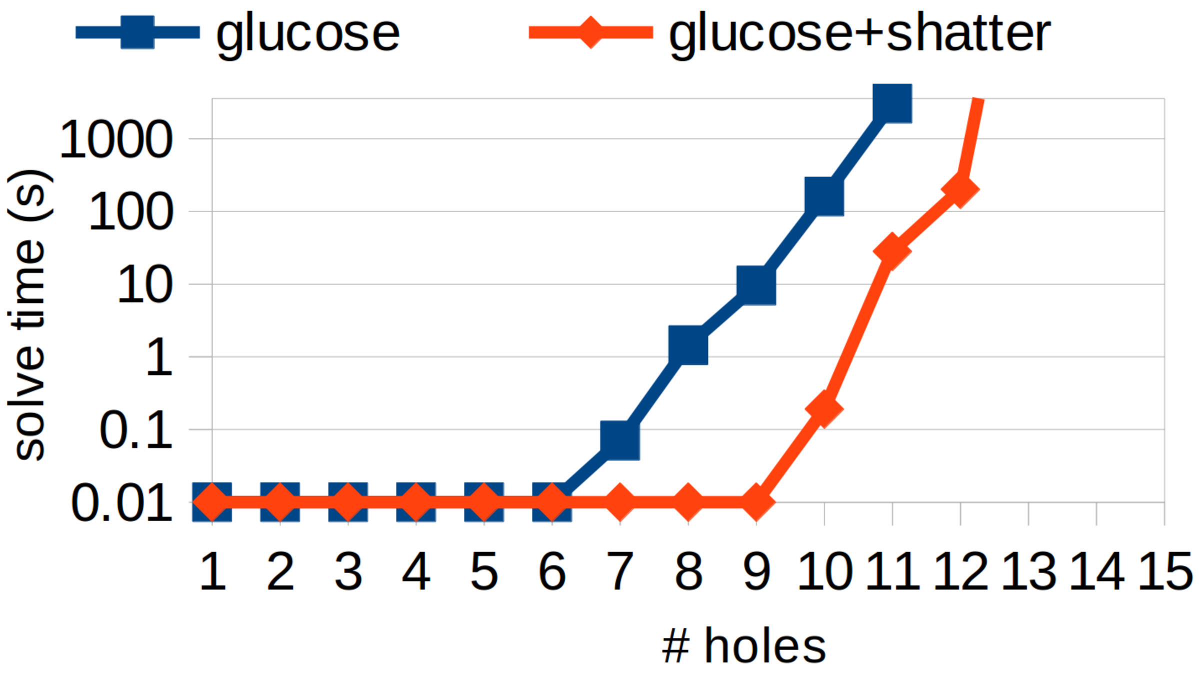
Pigeonhole!

Own Shatter experiment:



Pigeonhole!

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
  - **gen(Σ)**
  - **variable order**

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

- |full sbf| = $O(n^2)$

Pigeonhole!

- Propositional encoding reduces "pigeon interchangeability" to "**row interchangeability**"
- Shatter's sbf's are complete [4] with correct choice of
  - **gen(Σ)**
  - **variable order**

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

- |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 σ_1 , σ_2 $\in gen(\Sigma)$ 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 σ_1 , σ_2 ϵgen(Σ) that form 2 subsequent row swaps
    - forms initial **3-rowed variable matrix M**
2. Apply every σ ϵ gen(Σ) to all detected rows **r** ϵ M so far
    - images σ(r) disjoint of M are candidates to extend M
    - test if swap r ↔ σ(r) is a symmetry by syntactical check on φ
    - if success, **extend M with σ(r)**

Symmetry breaking: BreakID [5]

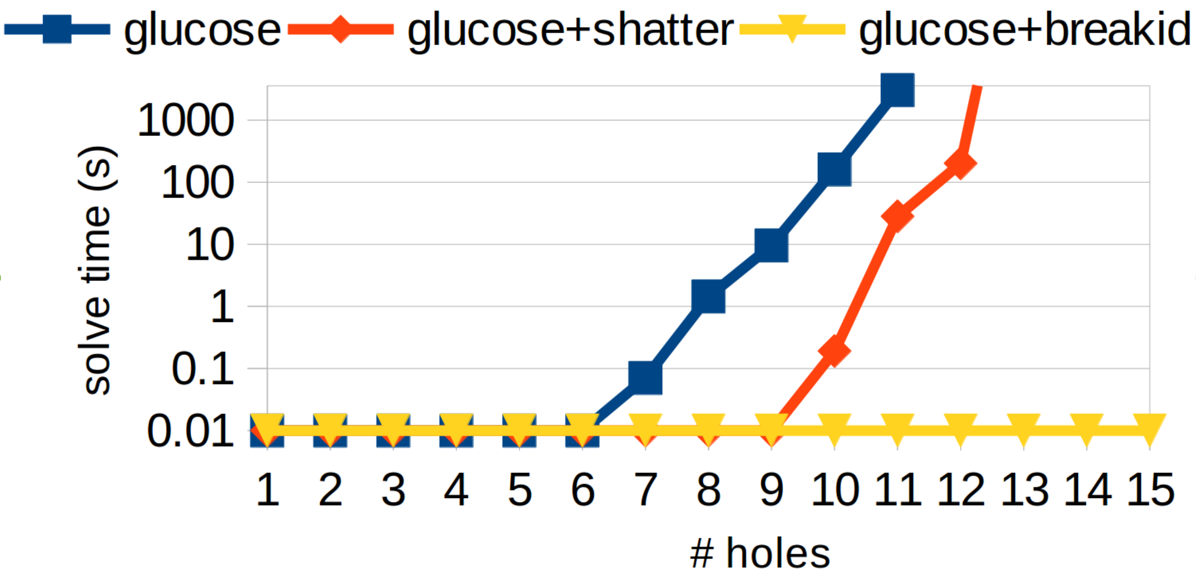
Core idea: detect "row swap" symmetries

*Approximative algorithm

1. Search σ_1 , σ_2 ϵgen(Σ) that form 2 subsequent row swaps
   - forms initial **3-rowed variable matrix M**
2. Apply every σ ϵ gen(Σ) to all detected rows **r** ϵ M so far
   - images σ(r) disjoint of M are candidates to extend M
   - test if swap r ↔ σ(r) is a symmetry by syntactical check on φ
   - if success, **extend M with σ(r)**
3. Use **Saucy** to extend gen(Σ) with new symmetry generators by fixing all variable nodes with variable in M, first row excepted

Symmetry breaking: BreakID [5]

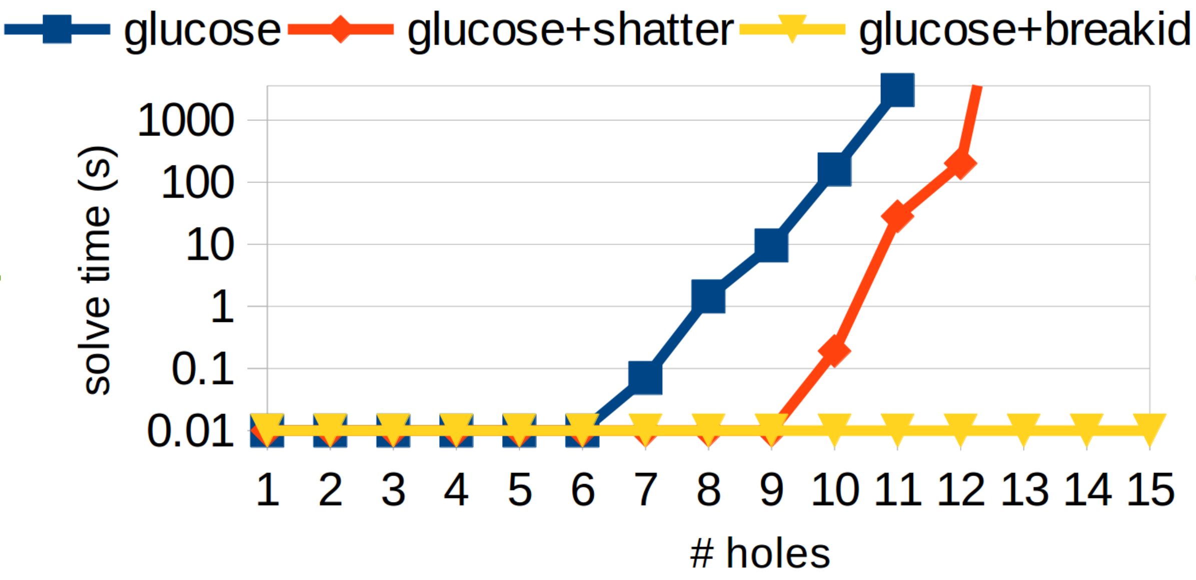
Core idea: detect "row swap" symmetries



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 σ is binary:

$$\neg x_1 \vee \sigma(x_1)$$

- x is **stabilized** by Σ iff Σ(x)={x}
- Given Σ with **smallest non-stabilized x**, for each x' ϵ Σ(x):

$$\neg x \vee x'$$

  is clause of sbf under some σ ϵ Σ

Symmetry breaking: BreakID [5]

Core idea: exploit **binary sbf clauses**

Symmetry breaking: BreakID [5]

Core idea: exploit **binary sbf clauses**

- Create **stabilizer chain** of Σ:

$$\Sigma \supset \Sigma_1 \supset \Sigma_2 \supset \ldots \supset 1$$

- Σ_i is the **stabilizer subgroup** of Σ_{i-1} stabilizing the next non-stabilized variable in ordering
  - Σ_i have different smallest non-stabilized variables x_i
- For each $x' \in \Sigma_i(x_i)$:

$$\neg x_i \vee x'$$

  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** σ 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** σ 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 intrest [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 intrest [11] [14]
- How about **general interchangeability** over arbitrary high dimensional relations?

Symmetry breaking: On completeness

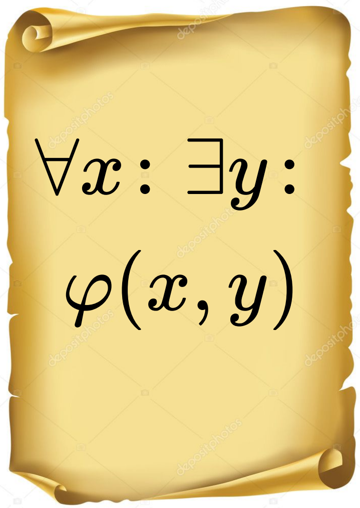
- Pigeon **interchangeability** can be completely broken with polynomial sbf
- How about **edge interchangeability**?
  - E.g., find coloring of complete graph (Ramsey numbers)
  - Recent intrest [11] [14]
- How about **general interchangeability** over arbitrary high dimensional relations?

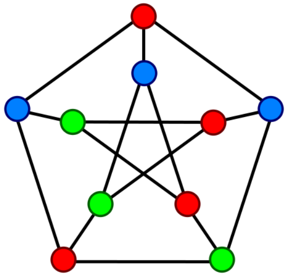
Tractable sbf for edge interchangeability?

Symmetry breaking: Prefix breaking [7]

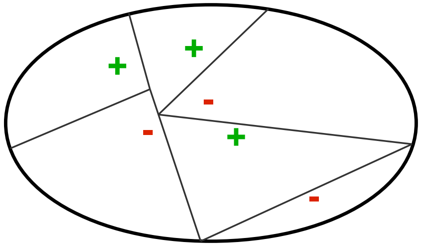
Core idea: enumerate asymmetrical assignments to variable prefix

High-level constraints → Graph representation → Symmetry class representative enumeration



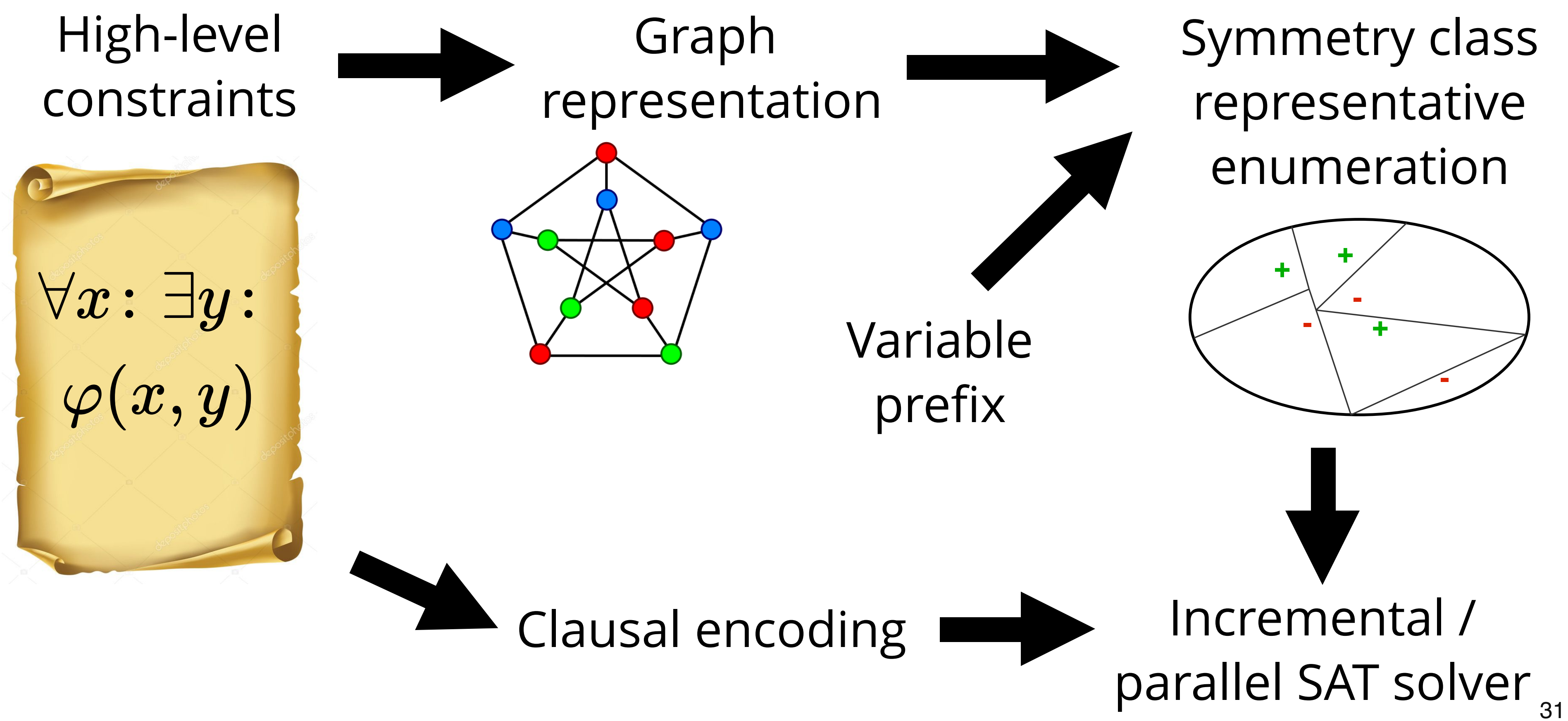


Variable prefix ↗



Symmetry breaking: Prefix breaking [7]

Core idea: enumerate asymmetrical assignments to variable prefix



6. Non-breaking approaches

Symmetry handling: SymChaff [8]

Core idea: search decisions consider row interchangeability

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

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

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

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:

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}
x_{41}	x_{42}	x_{43}

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:

0	x_{12}	x_{13}
0	x_{22}	x_{23}
0	x_{32}	x_{33}
1	x_{42}	x_{43}

Symmetry handling: SymChaff [8]

Strong performance on pigeonhole

	Problem	SymChaff
php	009-008	0.01
	013-012	0.01
	051-050	0.24
	091-090	0.84
	101-100	1.20

Symmetry handling: Symmetric learning [9]

Core idea: consider symmetrical learned clauses

Symmetry handling: Symmetric learning [9]

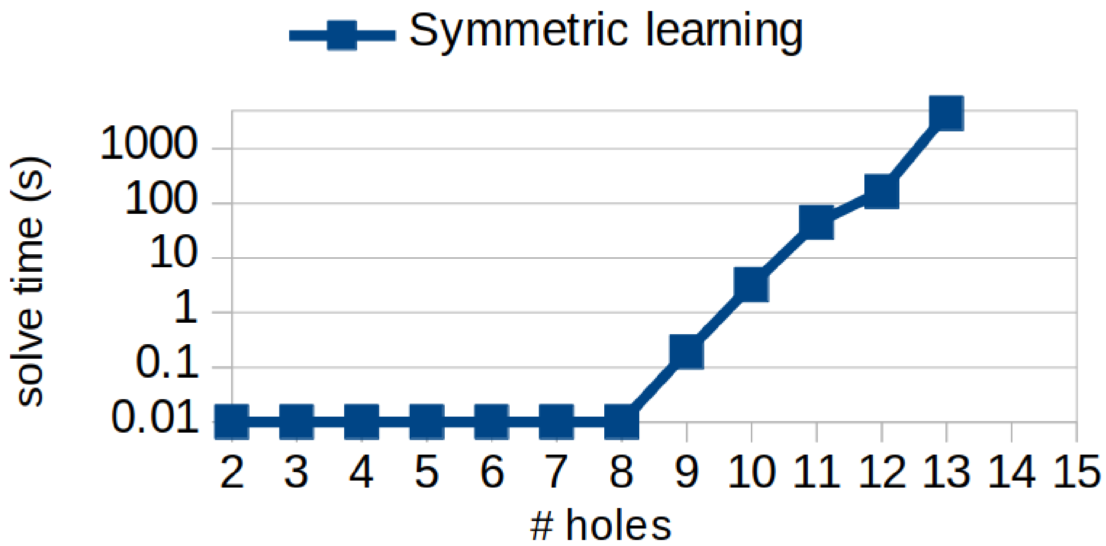
Core idea: consider symmetrical learned clauses

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

Symmetry handling: Symmetric learning [9]

Core idea: consider symmetrical learned clauses

- Learnt clauses are **logical consequences** of φ
- Whenever c is a consequence of φ, **so is σ(c)**
- Problem: Σ(c) is huge
  - Learn only σ(c) for σ ϵ gen(Σ)



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 σ(c) that **propagate at least once**
  - symmetries typically permute only a **fraction** of the literals
  - if c is unit, σ(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

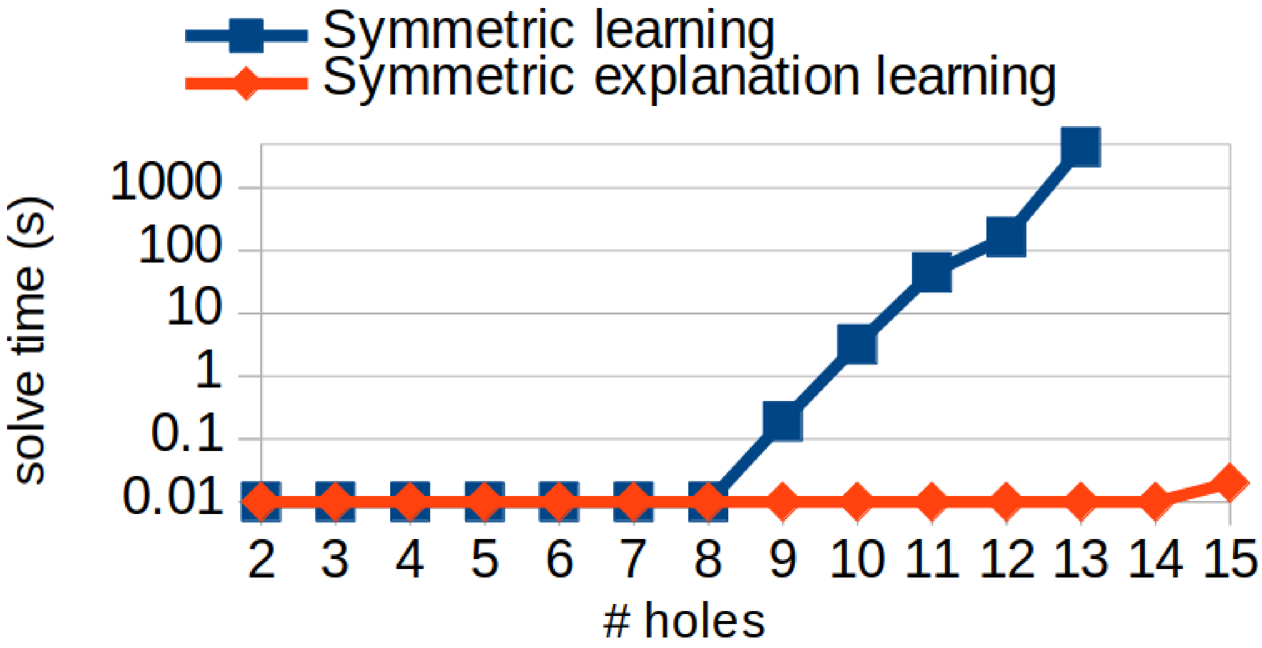
- For each σ ϵ gen(Σ), whenever c propagates, store σ(c) in a **separate clause store θ**
  - Propagation on θ happens only if standard unit propagation is at a fixpoint
  - Whenever a σ(c) ϵ θ **propagates**, **upgrade** it to a "normal" learned clause
  - After **backjump** over c's propagation level, **clear** σ(c) from θ

Symmetry handling: Symmetric explanation learning [10]

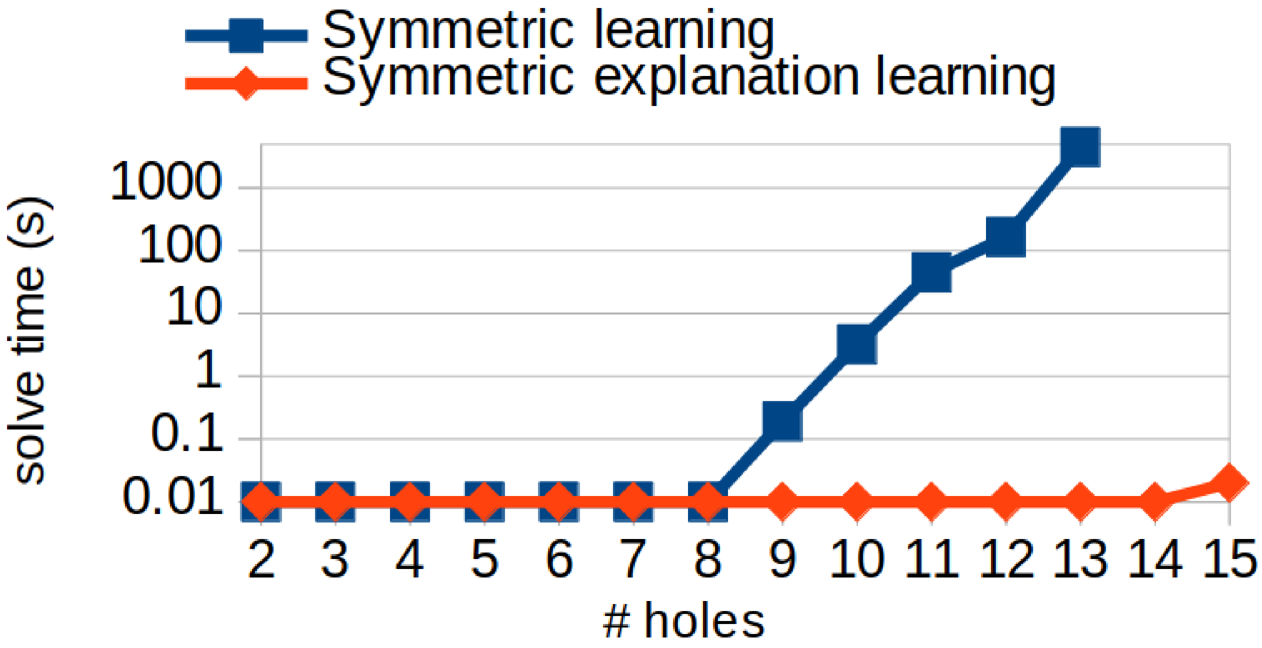
Core idea: consider useful symmetrical explanation clauses

- For each $\sigma \in gen(\Sigma)$, whenever c propagates, store $\sigma(c)$ in a **separate clause store θ**
  - Propagation on θ 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 θ
- Every learned $\sigma(c)$ is **useful** & **original**
- **Transitive** effect: track $\sigma'(\sigma(c))$ when $\sigma(c)$ propagates

Symmetry handling: Symmetric explanation learning [10]

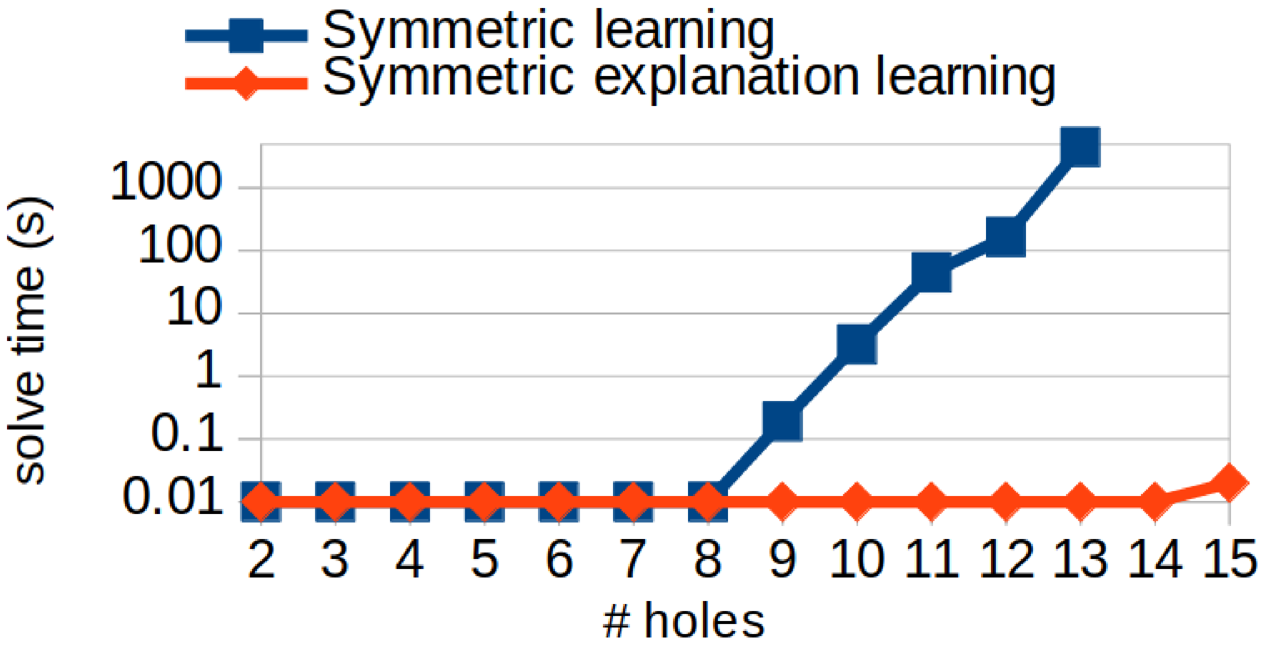


Symmetry handling:
Symmetric explanation learning [10]



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?

[1] Symmetry-Breaking Predicates for Search Problems (1996) Crawford et al.

[2] Efficient Symmetry-Breaking for Boolean Satisfiability (2003) Aloul et al.

[3] Symmetry and Satisfiability: An Update (2010) Katebi et al.

[4] Breaking row and column symmetries in matrix models (2002) Flener et al.

[5] Improved Static Symmetry Breaking for SAT (2016) Devriendt et al.

[6] CDCLSym: Introducing Effective Symmetry Breaking in SAT Solving (2018) Metin et al.

[7] An Adaptive Prefix-Assignment Technique for Symmetry Reduction (2017) Juntilla et al.

[8] Symchaff: exploiting symmetry in a structure-aware satisfiability solver (2009) Sabharwal

[9] Enhancing clause learning by symmetry in SAT solvers (2010) Benhamou

[10] Symmetric explanation learning: Effective dynamic symmetry handling for SAT (2017) Devriendt et al.

[11] Breaking Symmetries in Graphs: The Nauty Way (2016) Codish et al.

[12] Symmetries, almost symmetries, and lazy clause generation (2014) Chu et al.

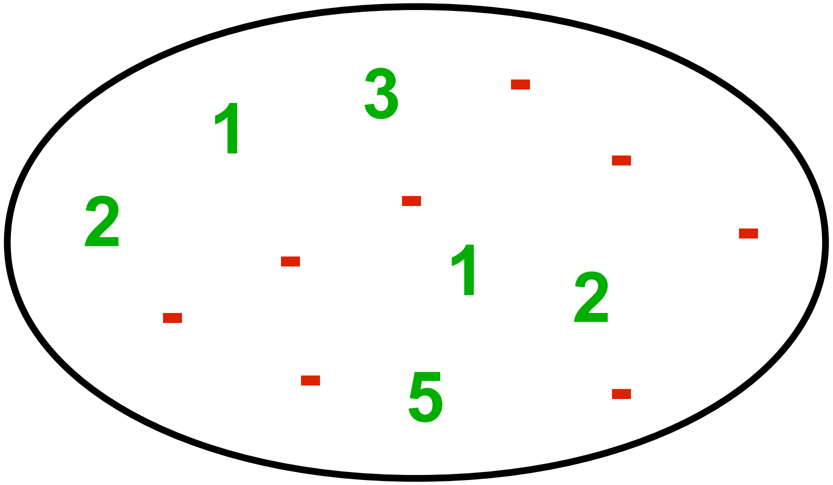
[13] Breaking symmetries in all-different problems (2005) Puget

[14] The quest for perfect and compact symmetry breaking for graph problems (2016)

7. Bonus: symmetry, local search & maxSAT

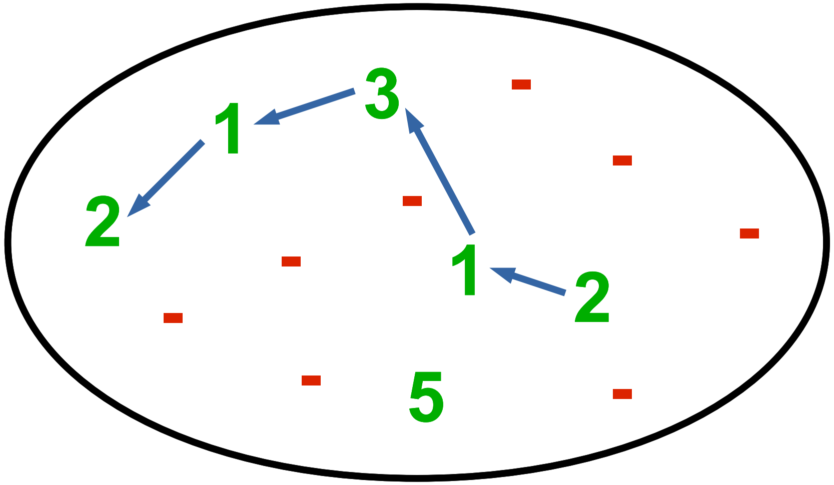
Bonus: symmetry, local search & maxSAT

- (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

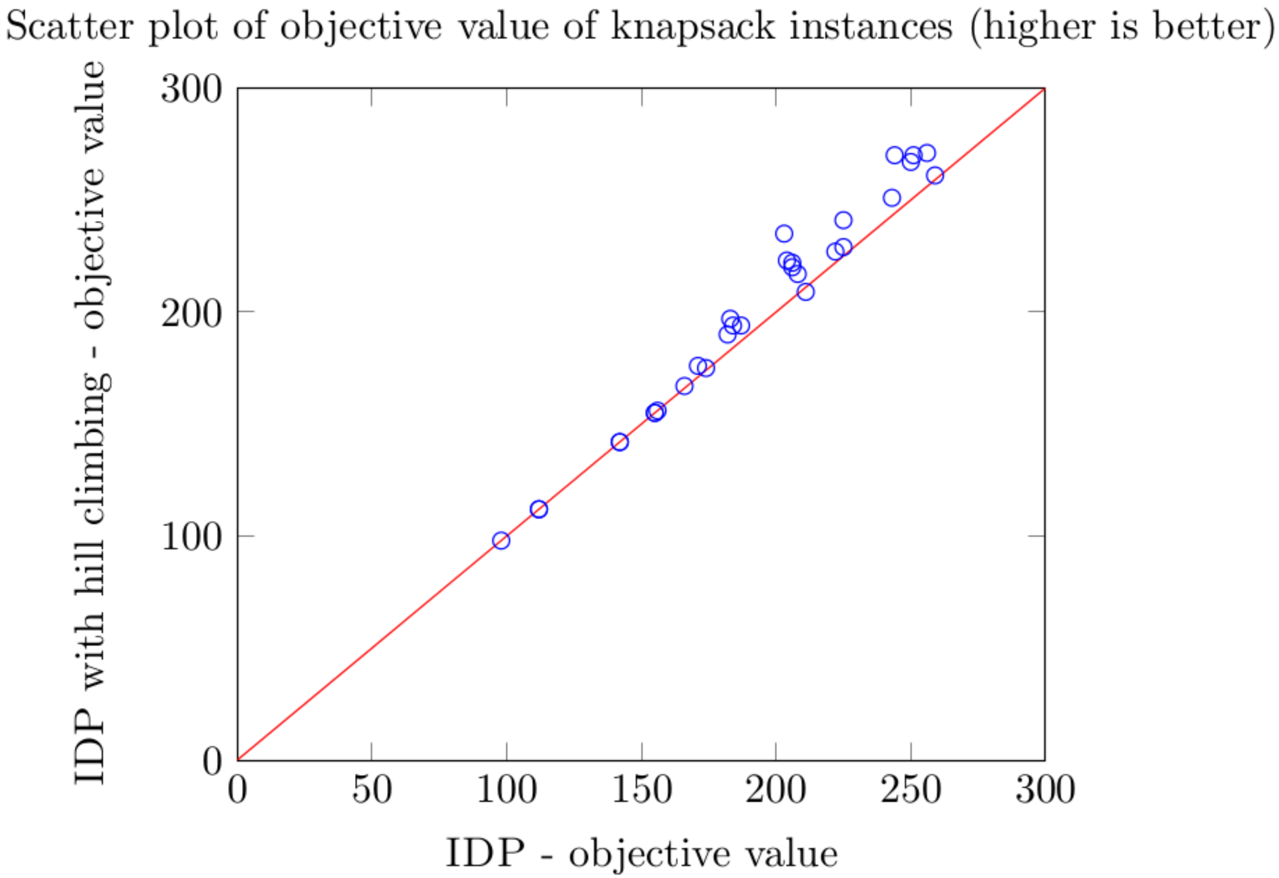
- (Satisfying) assignments now have an associated **score**
- Local search "**moves**" from one to the other based on **structure-preserving transformations**
- Designing local moves is typically done **by hand**...

Bonus: symmetry, local search & maxSAT

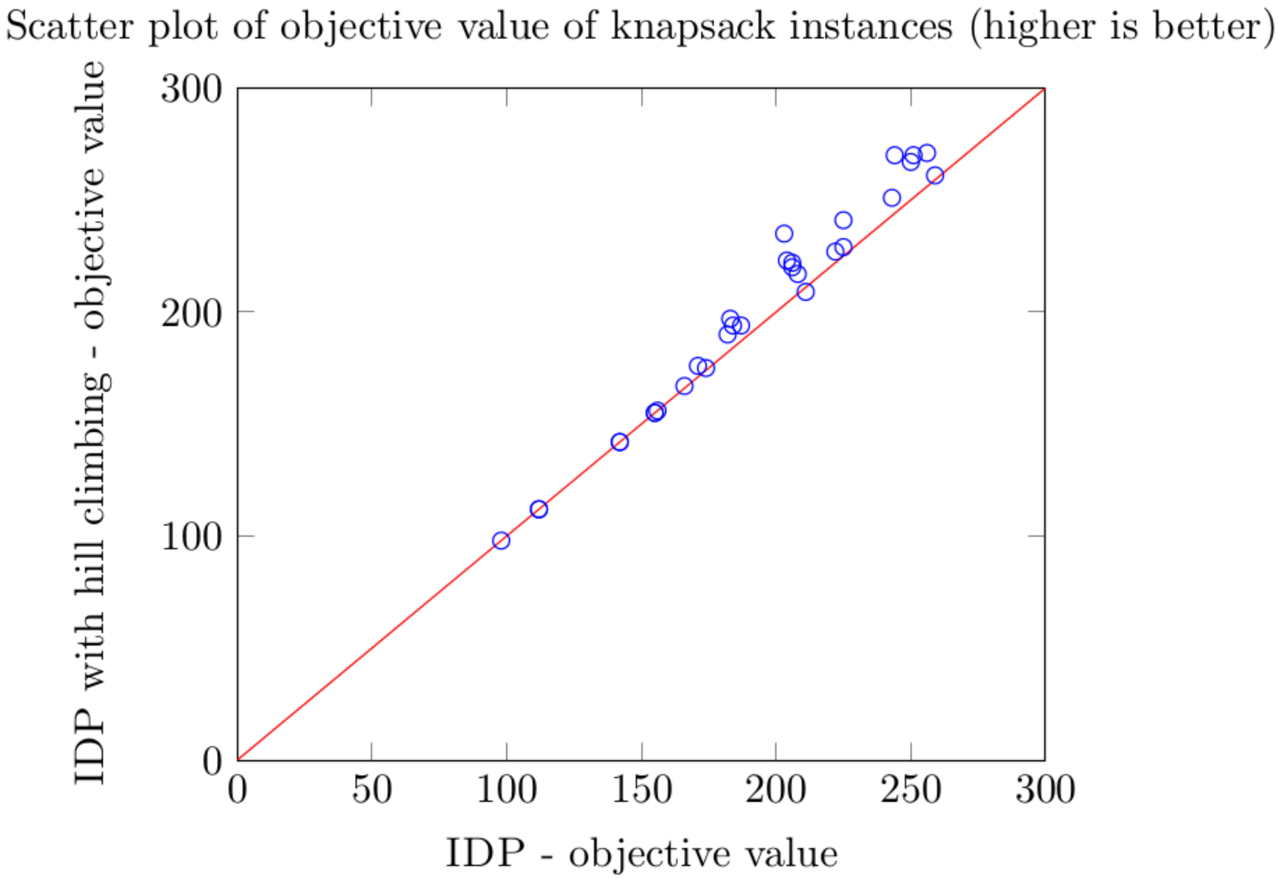
- (Satisfying) assignments now have an associated **score**
- Local search "**moves**" from one to the other based on **structure-preserving transformations**
- Designing local moves is typically done **by hand**...

Symmetries form moves!
Can be automatically detected!

Bonus: symmetry, local search & maxSAT



Bonus: symmetry, local search & maxSAT



Symmetry-based local search in weighted maxSAT?