Algorithms Transcending the SAT-Symmetry Interface

Markus Anders Mate Soos Pascal Schweitzer

SAT 2023

Boolean Satisfiability

Problem (SAT)

Input: *Boolean formula F*
Output: *Is there an assignment for the variables of F such that it evaluates to* ***true****?*

Boolean Satisfiability

Problem (SAT)

Input: *Boolean formula F*
Output: *Is there an assignment for the variables of F such that it evaluates to* ***true****?*

- F is always in *conjuctive normal form* (CNF), e.g., $F = (a \vee b \vee c) \wedge (\overline{a} \vee c) \wedge (b)$

Boolean Satisfiability

Problem (SAT)

Input: *Boolean formula F*
Output: *Is there an assignment for the variables of F such that it evaluates to* ***true****?*

- F is always in *conjuctive normal form* (CNF), e.g., $F = (a \vee b \vee c) \wedge (\overline{a} \vee c) \wedge (b)$
- a CNF F can also be written as a set of sets, i.e., $F = \{\{a, b, c\}, \{\overline{a}, c\}, \{b\}\}$

Boolean Satisfiability

Problem (SAT)

Input: *Boolean formula F*
Output: *Is there an assignment for the variables of F such that it evaluates to* ***true****?*

- F is always in *conjuctive normal form* (CNF), e.g., $F = (a \vee b \vee c) \wedge (\overline{a} \vee c) \wedge (b)$
- a CNF F can also be written as a set of sets, i.e., $F = \{\{a, b, c\}, \{\overline{a}, c\}, \{b\}\}$
- a disjunction $C \in F$ is called a *clause*

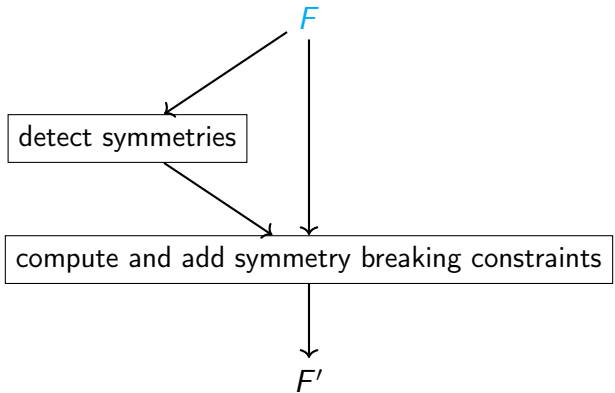
Boolean Satisfiability

Problem (SAT)

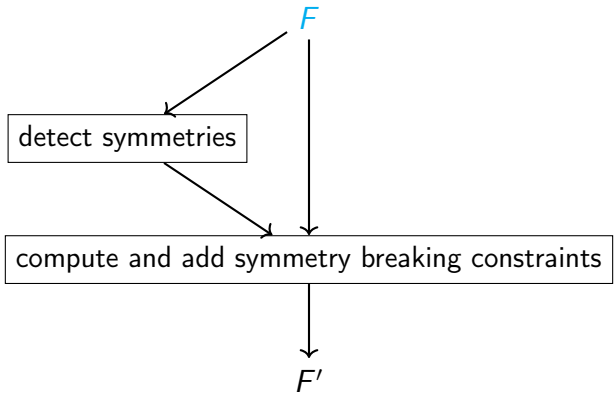
Input: *Boolean formula F*
Output: *Is there an assignment for the variables of F such that it evaluates to* ***true****?*

- F is always in *conjuctive normal form* (CNF), e.g., $F = (a \vee b \vee c) \wedge (\overline{a} \vee c) \wedge (b)$
- a CNF F can also be written as a set of sets, i.e., $F = \{\{a, b, c\}, \{\overline{a}, c\}, \{b\}\}$
- a disjunction $C \in F$ is called a *clause*
- an element $l \in C$ is called a *literal*

Symmetry Breaking

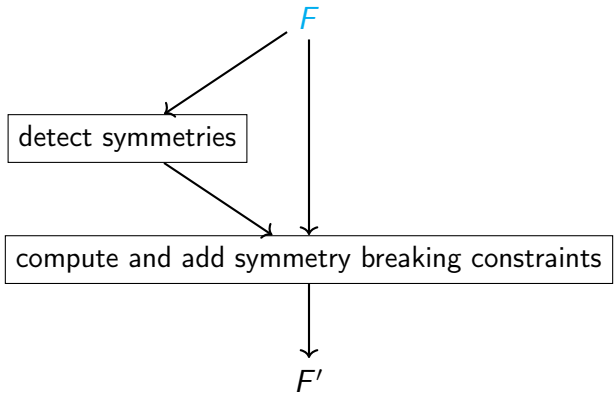


Symmetry Breaking



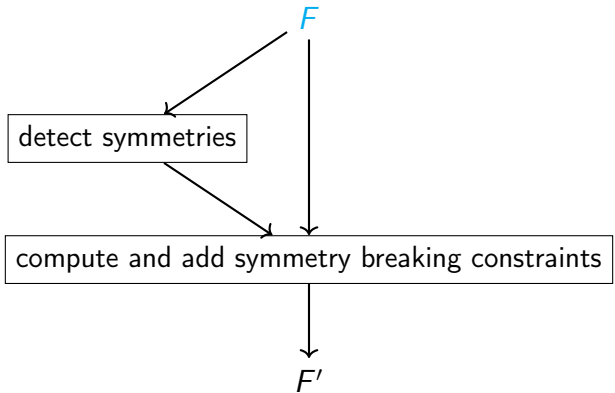
- F and F' equi-satisfiable

Symmetry Breaking



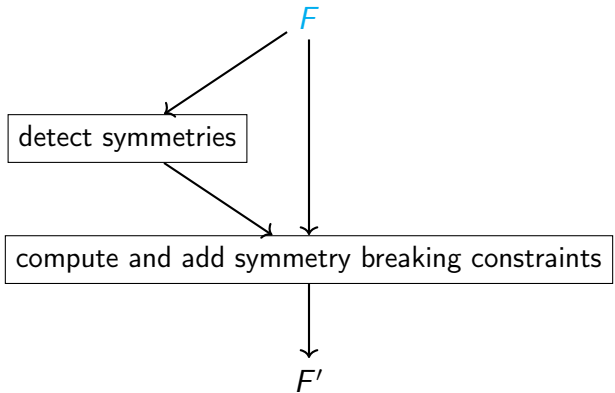
- F and F' equi-satisfiable
- F' often considerably easier to solve

Symmetry Breaking



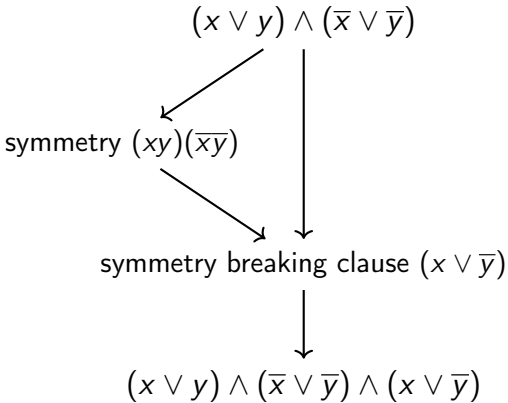
- F and F' equi-satisfiable
- F' often considerably easier to solve
- very effective on some instance types (combinatorics, logistics, ...)

Symmetry Breaking

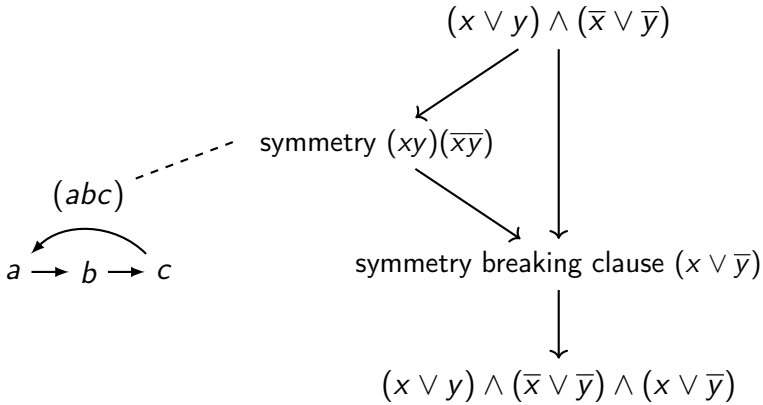


- F and F' equi-satisfiable
- F' often considerably easier to solve
- very effective on some instance types (combinatorics, logistics, ...)
- overhead is an issue

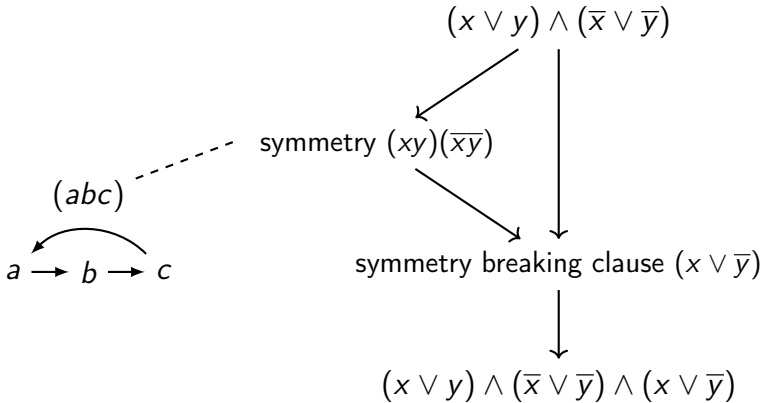
Symmetry Breaking: Example



Symmetry Breaking: Example

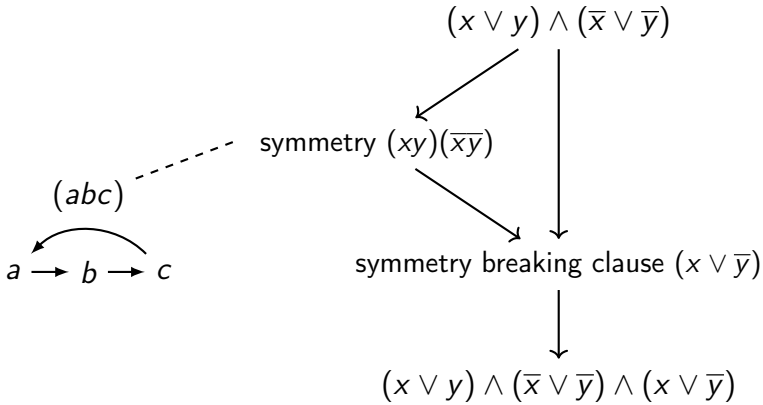


Symmetry Breaking: Example



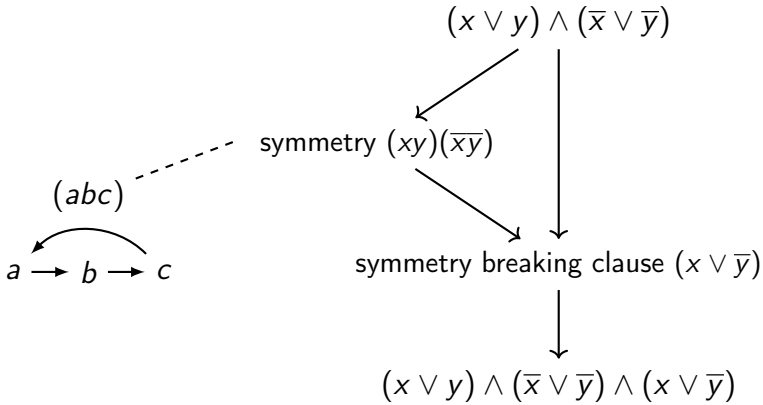
- symmetry implies $x = \text{true}, y = \text{false}$ gives same value as $x = \text{false}, y = \text{true}$

Symmetry Breaking: Example



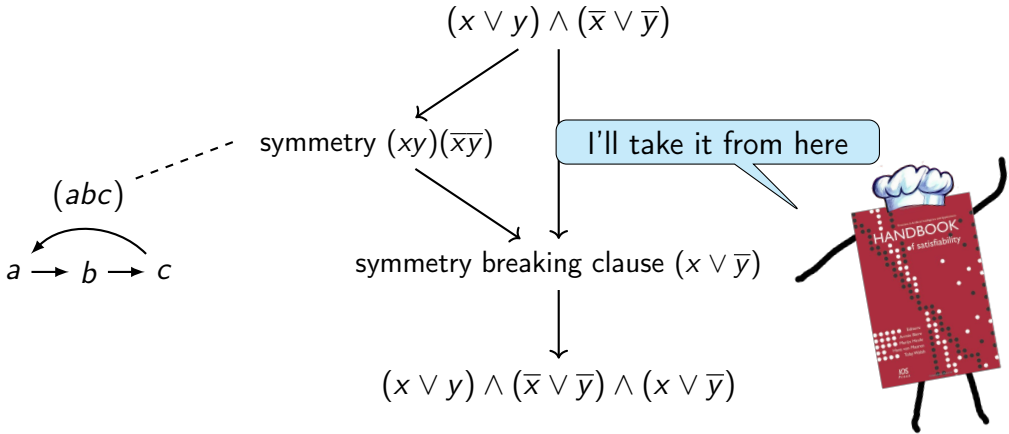
- symmetry implies $x = \text{true}, y = \text{false}$ gives same value as $x = \text{false}, y = \text{true}$
- falsify one of the symmetrical options with $(x \vee \overline{y})$

Symmetry Breaking: Example



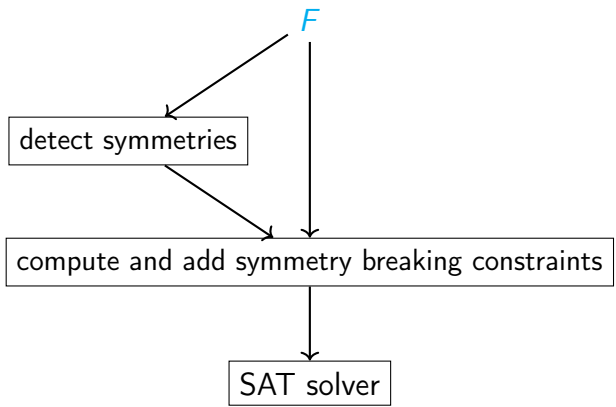
- symmetry implies $x = \text{true}, y = \text{false}$ gives same value as $x = \text{false}, y = \text{true}$
- falsify one of the symmetrical options with $(x \vee \overline{y})$
- many competing techniques (e.g., dynamic techniques)

Symmetry Breaking: Example



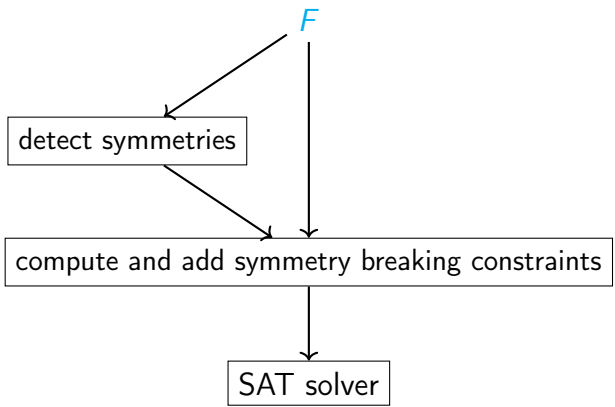
- symmetry implies $x = \text{true}, y = \text{false}$ gives same value as $x = \text{false}, y = \text{true}$
- falsify one of the symmetrical options with $(x \vee \overline{y})$
- many competing techniques (e.g., dynamic techniques)

Symmetry Breaking

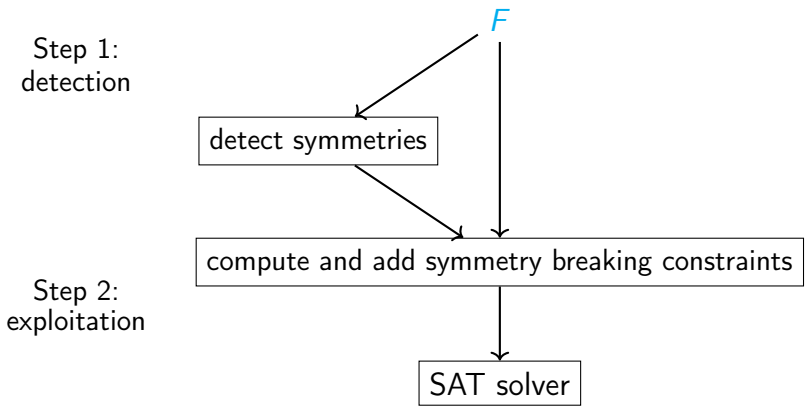


Symmetry Breaking

Step 2:
exploitation



Symmetry Breaking

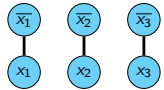


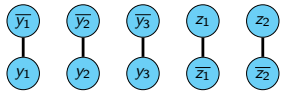
Symmetry Detection: Model Graph

$$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$

Symmetry Detection: Model Graph

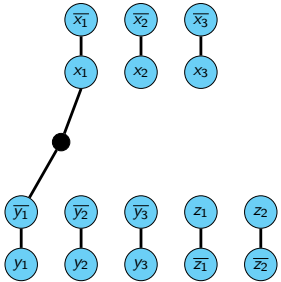
$$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$





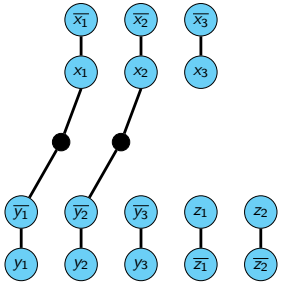
Symmetry Detection: Model Graph

$$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$



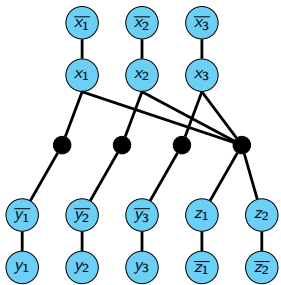
Symmetry Detection: Model Graph

$$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$



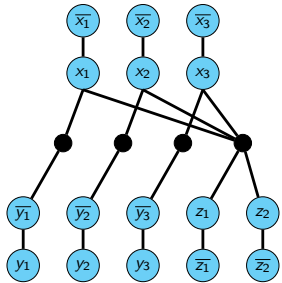
Symmetry Detection: Model Graph

$$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$



Symmetry Detection: Model Graph

$$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$



- symmetries of graph are symmetries of formula (and vice versa)

By the way... what are symmetries again? (CNF)

- bijection of literals $\varphi : \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ with $\varphi(F) = F$

By the way... what are symmetries again? (CNF)

- bijection of literals $\varphi : \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ with $\varphi(F) = F$
- (and it should also induce a well-defined bijection on the variables)

By the way... what are symmetries again? (CNF)

- bijection of literals $\varphi : \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ with $\varphi(F) = F$
- (and it should also induce a well-defined bijection on the variables)

Back to our example...

By the way... what are symmetries again? (CNF)

- bijection of literals $\varphi : \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ with $\varphi(F) = F$
- (and it should also induce a well-defined bijection on the variables)

Back to our example...

$$F = \{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$

By the way... what are symmetries again? (CNF)

- bijection of literals $\varphi : \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ with $\varphi(F) = F$
- (and it should also induce a well-defined bijection on the variables)

Back to our example...

$$F = \{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$
$$\varphi = (x_1 x_2)(\overline{x_1 x_2})(y_1 y_2)(\overline{y_1 y_2})$$

By the way... what are symmetries again? (CNF)

- bijection of literals $\varphi : \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ with $\varphi(F) = F$
- (and it should also induce a well-defined bijection on the variables)

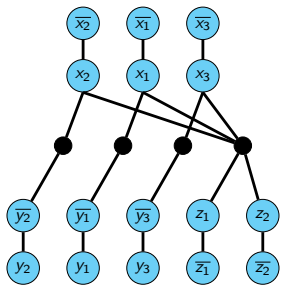
Back to our example...

$$F = \{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$$
$$\varphi = (x_1 x_2)(\overline{x_1 x_2})(y_1 y_2)(\overline{y_1 y_2})$$

$\varphi(F) =$
$\{\varphi(x_1) \vee \varphi(\overline{y_1}), \varphi(x_2) \vee \varphi(\overline{y_2}), \varphi(x_3) \vee \varphi(\overline{y_3}), \varphi(x_1) \vee \varphi(x_2) \vee \varphi(x_3) \vee \varphi(z_1) \vee \varphi(z_2)\} =$
$\{(x_2 \vee \overline{y_2}), (x_1 \vee \overline{y_1}), (x_3 \vee \overline{y_3}), (x_2 \vee x_1 \vee x_3 \vee z_1 \vee z_2)\} =$
$\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\} =$
F

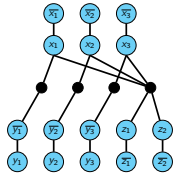
By the way... what are symmetries again? (Graphs)

- bijection of vertices $\varphi : V \to V$ with $\varphi(G) = (\varphi(V), \varphi(E)) = G$



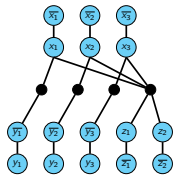
Symmetry Detection on Graphs

Input: Graph G



Symmetry Detection on Graphs

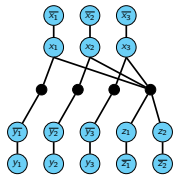
Input: Graph G



Output: All symmetries $\mathrm{Aut}(G)$

Symmetry Detection on Graphs

Input: Graph G

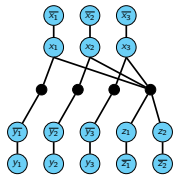


Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$

Symmetry Detection on Graphs

Input: Graph G



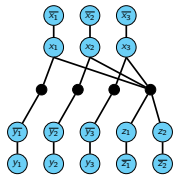
Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$

$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$

Symmetry Detection on Graphs

Input: Graph G

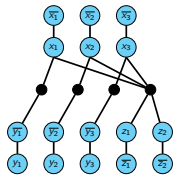


Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3})$,
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})$,

Symmetry Detection on Graphs

Input: Graph G

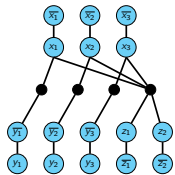


Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2}),$
$(z_1z_2)(\overline{z_1z_2}),$

Symmetry Detection on Graphs

Input: Graph G

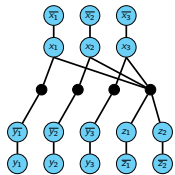


Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2}),$
$(z_1z_2)(\overline{z_1z_2}),$
$(x_1x_3)(\overline{x_1x_3})(y_1y_3)(\overline{y_1y_3}),$

Symmetry Detection on Graphs

Input: Graph G

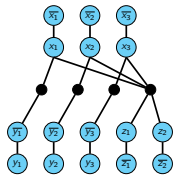


Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3})$,
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})$,
$(z_1z_2)(\overline{z_1z_2})$,
$(x_1x_3)(\overline{x_1x_3})(y_1y_3)(\overline{y_1y_3})$,
$(x_2x_3)(\overline{x_2x_3})(y_2y_3)(\overline{y_2y_3})$,

Symmetry Detection on Graphs

Input: Graph G



Output: All symmetries $\mathrm{Aut}(G)$

$\mathrm{Aut}(G) = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3})$,
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})$,
$(z_1z_2)(\overline{z_1z_2})$,
$(x_1x_3)(\overline{x_1x_3})(y_1y_3)(\overline{y_1y_3})$,
$(x_2x_3)(\overline{x_2x_3})(y_2y_3)(\overline{y_2y_3})$,
$\ldots$

Tread carefully...

- a graph G can have an exponential number of symmetries

Tread carefully...

- a graph G can have an exponential number of symmetries
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition

Tread carefully...

- a graph G can have an exponential number of symmetries
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition
- if φ and φ' are symmetries, so is $\varphi \circ \varphi'$

Tread carefully...

- a graph G can have an exponential number of symmetries
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition
- if φ and φ' are symmetries, so is $\varphi \circ \varphi'$
- only write down small subset S which *generates* $\mathrm{Aut}(G)$

Tread carefully...

- a graph G can have an exponential number of symmetries
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition
- if φ and φ' are symmetries, so is $\varphi \circ \varphi'$
- only write down small subset S which *generates* $\mathrm{Aut}(G)$

Problem (Symmetry Detection)

Input: *Graph G*

Tread carefully...

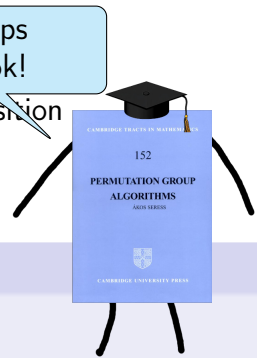
- a graph G can have an exponential number of symmetries
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition
- if φ and φ' are symmetries, so is $\varphi \circ \varphi'$
- only write down small subset S which *generates* $\mathrm{Aut}(G)$

Problem (Symmetry Detection)

Input: *Graph G*
Output: *Generating set $S \subseteq \mathrm{Aut}(G)$ with $\langle S \rangle = \mathrm{Aut}(G)$*

Tread carefully...

- a graph G can have an exponential number
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition
- if φ and φ' are symmetries, so is $\varphi \circ \varphi'$
- only write down small subset S which *generates* $\mathrm{Aut}(G)$

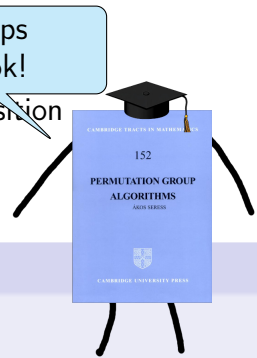


Problem (Symmetry Detection)

Input: *Graph* G
Output: *Generating set* $S \subseteq \mathrm{Aut}(G)$ *with* $\langle S \rangle = \mathrm{Aut}(G)$

Tread carefully...

- a graph G can have an exponential number
- symmetries $\mathrm{Aut}(G)$ form a permutation group under composition
- if φ and φ' are symmetries, so is $\varphi \circ \varphi'$
- only write down small subset S which *generates* $\mathrm{Aut}(G)$



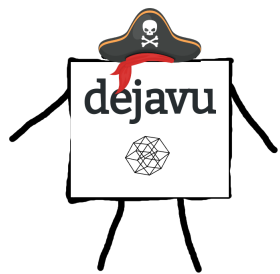
Problem (Symmetry Detection)

Input: *Graph* G
Output: *Generating set* $S \subseteq \mathrm{Aut}(G)$ *with* $\langle S \rangle = \mathrm{Aut}(G)$

- state-of-the-art tools are nauty, saucy, bliss, Traces, dejavu

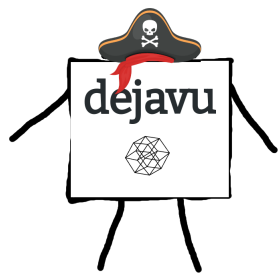
Symmetry Detection: dejavu

- I've been building the symmetry detection tool `dejavu` for the past 5 years



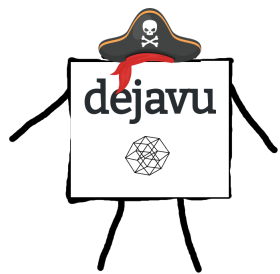
Symmetry Detection: dejavu

- I've been building the symmetry detection tool `dejavu` for the past 5 years
- it's fast



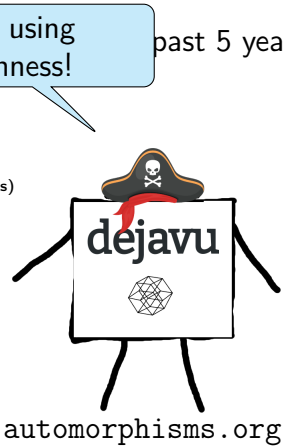
Symmetry Detection: dejavu

- I've been building the symmetry detection tool `dejavu` for the past 5 years
- it's fast
- it's fastest on SAT graphs



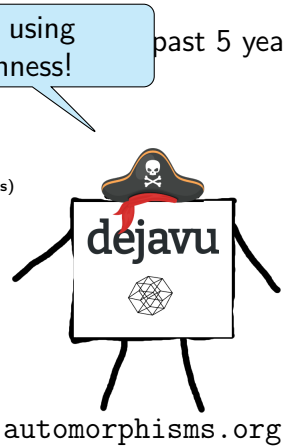
Symmetry Detection: dejavu

- I've been building the symmetry detect past 5 years
- it's fast
- it's fastest on SAT graphs
- has one-sided bounded error (does not matter for most applications)

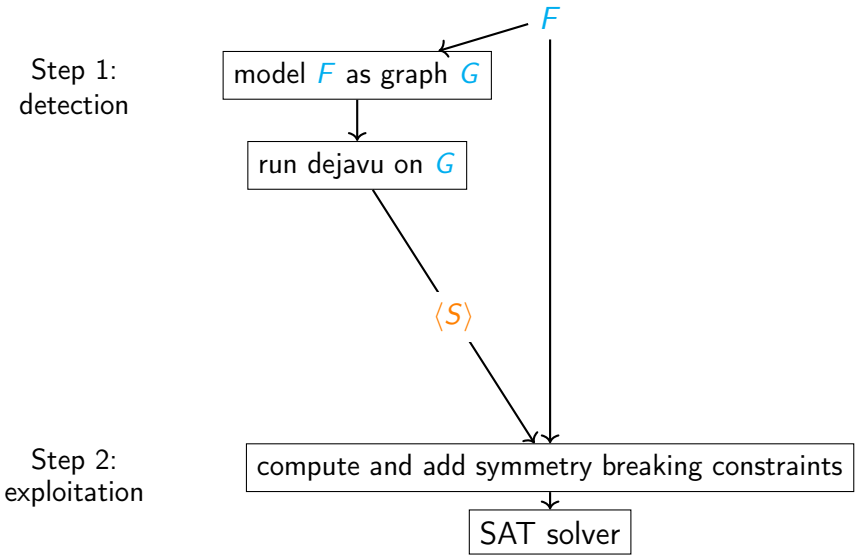


Symmetry Detection: dejavu

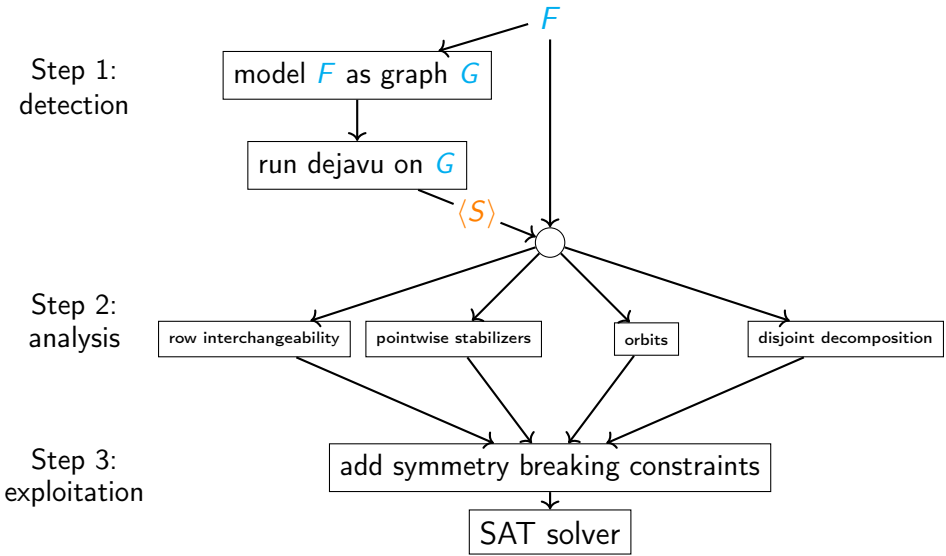
- I've been building the symmetry detect … past 5 years
- it's fast
- it's fastest on SAT graphs
- has one-sided bounded error (does not matter for most applications)
- randomness is *inherent* to the design



Symmetry Breaking: Refined Picture



Symmetry Breaking: Refined Picture II



Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- structural analysis of permutation group

Symmetry Breaking Sub-Tasks

- structural analysis of permutation group
- enable efficient production of symmetry breaking clauses

Symmetry Breaking Sub-Tasks

- structural analysis of permutation group
- enable efficient production of symmetry breaking clauses
- implementation issues in sate-of-the-art symmetry breaking (BreakID, SCIP, ...)

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- structural analysis of permutation group
- enable efficient production of symmetry breaking clauses
- implementation issues in sate-of-the-art symmetry breaking (BreakID, SCIP, ...)
  - expensive, make up majority of runtime in some instances

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- structural analysis of permutation group
- enable efficient production of symmetry breaking clauses
- implementation issues in sate-of-the-art symmetry breaking (BreakID, SCIP, ...)
  - expensive, make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators

Symmetry Breaking Sub-Tasks

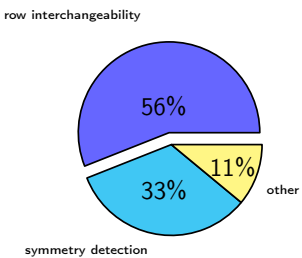
row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators

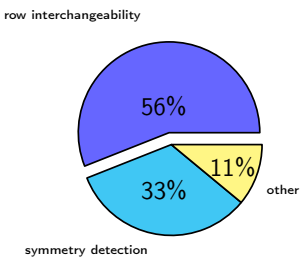


BreakID on PHP instances

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators



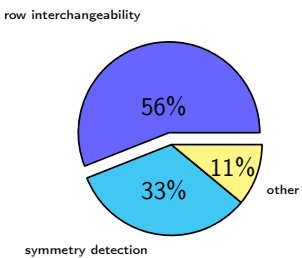
BreakID on PHP instances

Relies on "transpositions":
$S_1 = \{(12), (23), (34)\}$
$S_2 = \{(12), (1234)\}$
$\langle S_1 \rangle = \langle S_2 \rangle$

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators



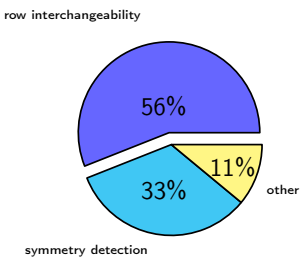
BreakID on PHP instances

Relies on "transpositions":
$S_1 = \{(12), (23), (34)\}$ ✓
$S_2 = \{(12), (1234)\}$
$\langle S_1 \rangle = \langle S_2 \rangle$

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators



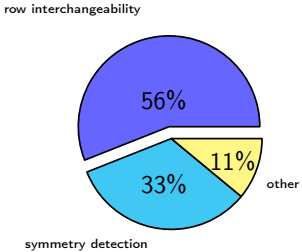
BreakID on PHP instances

Relies on "transpositions":
$S_1 = \{(12), (23), (34)\}$ ✓
$S_2 = \{(12), (1234)\}$ ✗
$\langle S_1 \rangle = \langle S_2 \rangle$

Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators



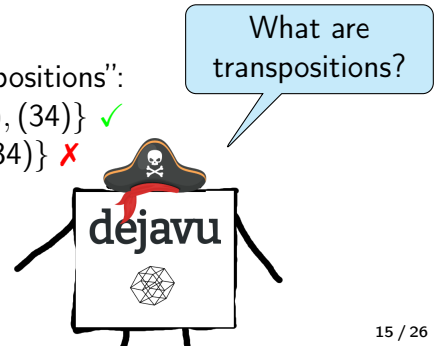
BreakID on PHP instances

Relies on "transpositions":

$S_1 = \{(12), (23), (34)\}$ ✓

$S_2 = \{(12), (1234)\}$ ✗

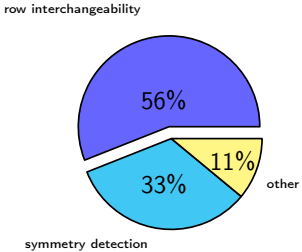
$\langle S_1 \rangle = \langle S_2 \rangle$



Symmetry Breaking Sub-Tasks

row interchangeability | pointwise stabilizers | orbits | disjoint decomposition

- implementation issues in sate-of-the-art symmetry exploitation (BreakID, SCIP, ...)
  - expensive, sometimes make up majority of runtime in some instances
  - not generic, rely on very specific properties of generators



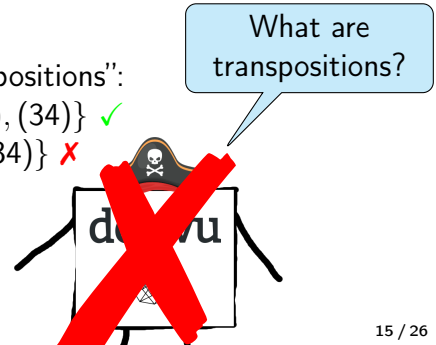
BreakID on PHP instances

Relies on "transpositions":

$S_1 = \{(12), (23), (34)\}$ ✓

$S_2 = \{(12), (1234)\}$ ✗

$\langle S_1 \rangle = \langle S_2 \rangle$



How could we solve them, exactly?

How could we solve them, exactly?

How could we solve them, exactly?

row interchangeability

no algorithm

pointwise stabilizers

orbits

disjoint decomposition

How could we solve them, exactly?

row interchangeability

no algorithm

pointwise stabilizers

Schreier-Sims

orbits

disjoint decomposition

How could we solve them, exactly?

row interchangeability	pointwise stabilizers	orbits	disjoint decomposition
no algorithm	Schreier-Sims	well-known algorithm	

How could we solve them, exactly?

row interchangeability	pointwise stabilizers	orbits	disjoint decomposition
no algorithm	Schreier-Sims	well-known algorithm	recent [Chang, Jefferson, '20]

How could we solve them, exactly?

Permutation Group Algorithms: Observations

Permutation Group Algorithms: Observations

- algorithms are "linear-time", but not linear-time

Permutation Group Algorithms: Observations

- algorithms are "linear-time", but not linear-time
  - running time is measured in terms of dense permutation representations

Permutation Group Algorithms: Observations

- algorithms are "linear-time", but not linear-time
  - running time is measured in terms of dense permutation representations
  - assume existence of a "strong" generating set

Permutation Group Algorithms: Observations

- algorithms are "linear-time", but not linear-time
  - running time is measured in terms of dense permutation representations
  - assume existence of a "strong" generating set
  - assume production of random elements...

Permutation Group Algorithms: Observations

- algorithms are "linear-time", but not linear-time
  - running time is measured in terms of dense permutation representations
  - assume existence of a "strong" generating set
  - assume production of random elements...
- doesn't make use of graphs or SAT formulas (obviously!)

Permutation Group Algorithms: Observations

- algorithms are "linear-time", but not linear-time
  - running time is measured in terms of dense permutation representations
  - assume existence of a "strong" generating set
  - assume production of random elements...
- doesn't make use of graphs or SAT formulas (obviously!)
- So... also not *quite* what we want?

SAT-Symmetry Algorithms

Our computational setting:

SAT-Symmetry Algorithms

Our computational setting:

1. we have access to both a graph (SAT formula) *and* corresponding group

SAT-Symmetry Algorithms

Our computational setting:

1. we have access to both a graph (SAT formula) *and* corresponding group
2. we want to measure running time in the "encoding size" of graphs and groups

SAT-Symmetry Algorithms

Our computational setting:

1. we have access to both a graph (SAT formula) *and* corresponding group
2. we want to measure running time in the "encoding size" of graphs and groups

Definition (Joint Graph-Group Pairs)

A graph G and generating S is called a joint graph-group pair, whenever $\langle S \rangle = \mathrm{Aut}(G)$.

SAT-Symmetry Algorithms

Our computational setting:

1. we have access to both a graph (SAT formula) *and* corresponding group
2. we want to measure running time in the "encoding size" of graphs and groups

Definition (Joint Graph-Group Pairs)

A graph G and generating S is called a joint graph-group pair, whenever $\langle S \rangle = \mathrm{Aut}(G)$.

Definition (Instance-linear Running Time)

Given a SAT formula F, graph $G = (V, E)$, we call algorithms that run in time $\mathcal{O}(|F| + |V| + |E| + \mathrm{enc}(S))$ *instance-linear*, where $\mathrm{enc}(S) := \Sigma_{p \in S} |\mathrm{supp}(p)|$.

SAT-Symmetry Algorithms

Our computational setting:

1. we have access to both a graph (SAT formula) *and* corresponding group
2. we want to measure running time in the "encoding size" of graphs and groups

Definition (Joint Graph-Group Pairs)

A graph G and generating S is called a joint graph-group pair, whenever $\langle S \rangle = \mathrm{Aut}(G)$.

Definition (Instance-linear Running Time)

Given a SAT formula F, graph $G = (V, E)$, we call algorithms that run in time $\mathcal{O}(|F| + |V| + |E| + \mathrm{enc}(S))$ *instance-linear*, where $\mathrm{enc}(S) := \Sigma_{p \in S} |\mathrm{supp}(p)|$.

Are there better algorithms or heuristics in this setting?

SAT-Symmetry Algorithms

Our computational setting:

1. we have access to both a graph (SAT formula) *and* corresponding group
2. we want to measure running time in the "encoding size" of graphs and groups

Definition (Joint Graph-Group Pairs)

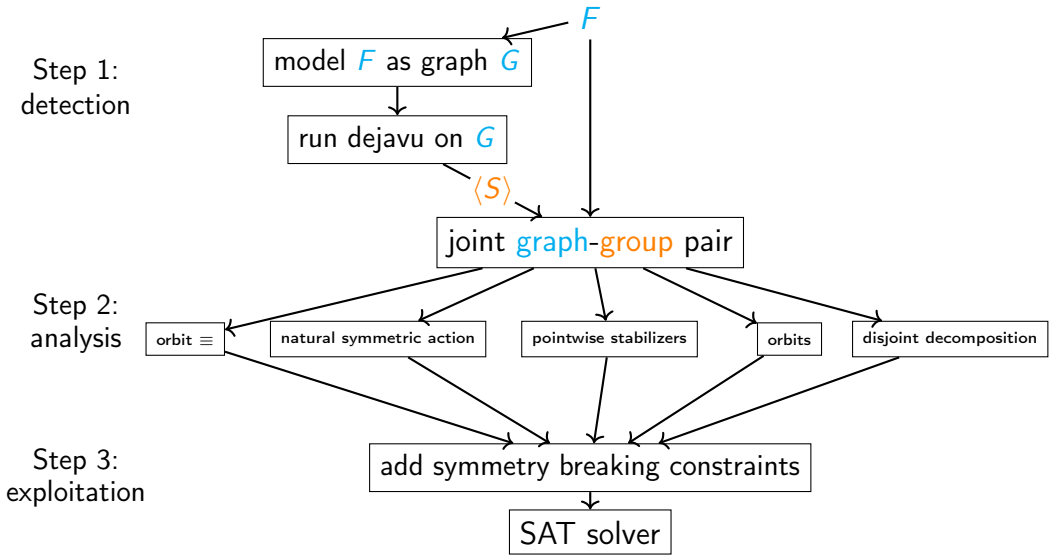
A graph G and generating S is called a joint graph-group pair, whenever $\langle S \rangle = \mathrm{Aut}(G)$.

Definition (Instance-linear Running Time)

Given a SAT formula F, graph $G = (V, E)$, we call algorithms that run in time $\mathcal{O}(|F| + |V| + |E| + \mathrm{enc}(S))$ *instance-linear*, where $\mathrm{enc}(S) := \Sigma_{p \in S} |\operatorname{supp}(p)|$.

Are there better algorithms or heuristics in this setting? **Yes!**

Symmetry Breaking: Refined Picture III



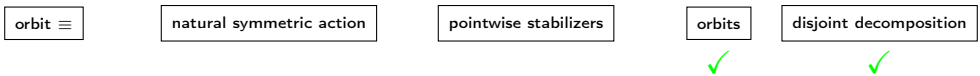
Algorithms in the Paper

orbit $\equiv$ | natural symmetric action | pointwise stabilizers | orbits | disjoint decomposition

Algorithms in the Paper

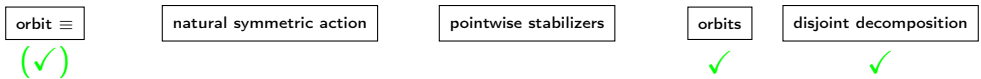
orbit $\equiv$ | natural symmetric action | pointwise stabilizers | orbits ✓ | disjoint decomposition

Algorithms in the Paper



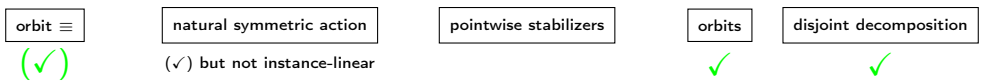
- finest disjoint decomposition in instance-quasi-linear time (instance-linear if orbits available)

Algorithms in the Paper



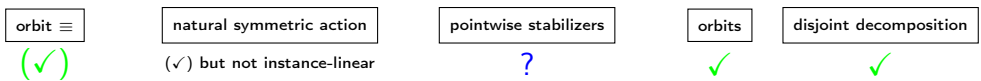
- finest disjoint decomposition in instance-quasi-linear time (instance-linear if orbits available)
- equivalent symmetric orbits in instance-linear time under "unique cycle assumption"

Algorithms in the Paper



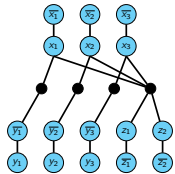
- finest disjoint decomposition in instance-quasi-linear time (instance-linear if orbits available)
- equivalent symmetric orbits in instance-linear time under "unique cycle assumption"
- algorithm for natural symmetric action based on known computational group theory (point out instance-linear graph-based heuristics)

Algorithms in the Paper

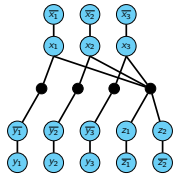


- finest disjoint decomposition in instance-quasi-linear time (instance-linear if orbits available)
- equivalent symmetric orbits in instance-linear time under "unique cycle assumption"
- algorithm for natural symmetric action based on known computational group theory (point out instance-linear graph-based heuristics)

Finest Disjoint Direct Decomposition: What it's about

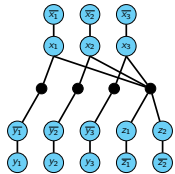


Finest Disjoint Direct Decomposition: What it's about



$$\langle S \rangle = \mathrm{Aut}(G)$$

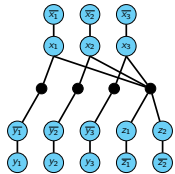
Finest Disjoint Direct Decomposition: What it's about



$\langle S \rangle = \mathrm{Aut}(G)$

$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$

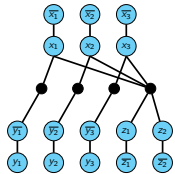
Finest Disjoint Direct Decomposition: What it's about



$\langle S \rangle = \mathrm{Aut}(G)$

$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$
x's and y's are independent of z's

Finest Disjoint Direct Decomposition: What it's about

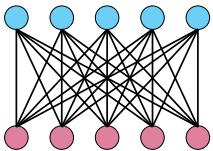


$\langle S \rangle = \mathrm{Aut}(G)$

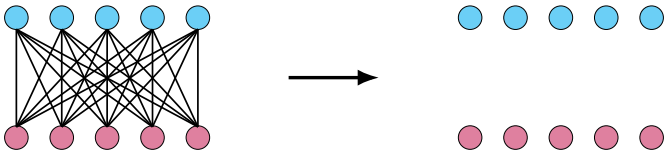
$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$
x's and y's are independent of z's

$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})(z_1z_2)(\overline{z_1z_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$
Here they are not?

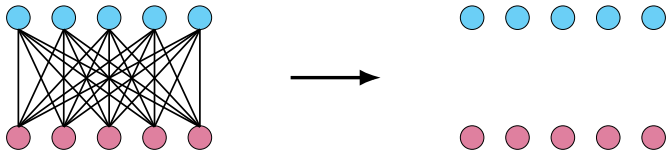
Finest Disjoint Direct Decomposition: Flip edges

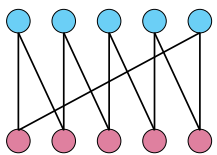


Finest Disjoint Direct Decomposition: Flip edges

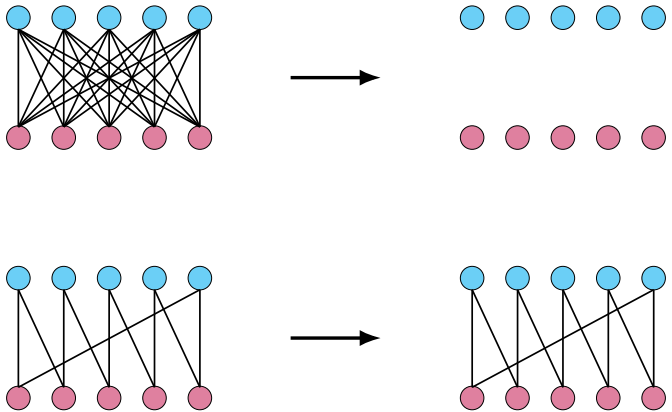


Finest Disjoint Direct Decomposition: Flip edges



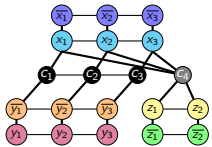


Finest Disjoint Direct Decomposition: Flip edges



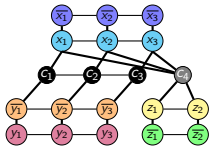
Finest Disjoint Direct Decomposition: Algorithm

1. color vertices with "orbits", connect vertices of orbit with a path

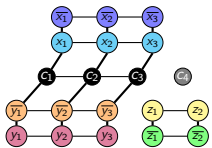


Finest Disjoint Direct Decomposition: Algorithm

1. color vertices with "orbits", connect vertices of orbit with a path

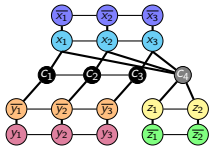


2. flip edges between colors

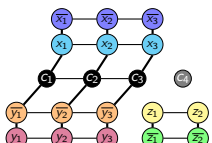


Finest Disjoint Direct Decomposition: Algorithm

1. color vertices with "orbits", connect vertices of orbit with a path



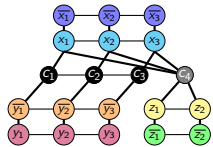
2. flip edges between colors



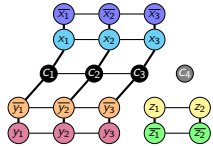
3. compute connected components

Finest Disjoint Direct Decomposition: Algorithm

1. color vertices with "orbits", connect vertices of orbit with a path



2. flip edges between colors



3. compute connected components

Lemma

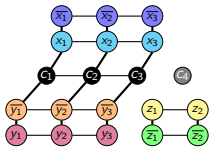
Vertices are in the same connected component if and only if they are in the same factor of the finest direct disjoint decomposition.

Finest Disjoint Direct Decomposition: Algorithm

- we can split generators according to connected components

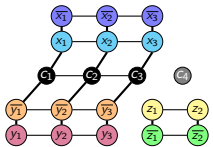
Finest Disjoint Direct Decomposition: Algorithm

- we can split generators according to connected components



Finest Disjoint Direct Decomposition: Algorithm

- we can split generators according to connected components

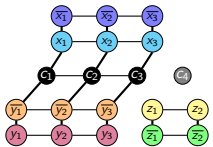


$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})(z_1z_2)(\overline{z_1z_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$

not independent

Finest Disjoint Direct Decomposition: Algorithm

- we can split generators according to connected components

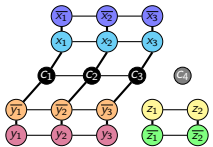


$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})(z_1z_2)(\overline{z_1z_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$ $\rightarrow$

not independent

Finest Disjoint Direct Decomposition: Algorithm

- we can split generators according to connected components



$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2})(z_1z_2)(\overline{z_1z_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$

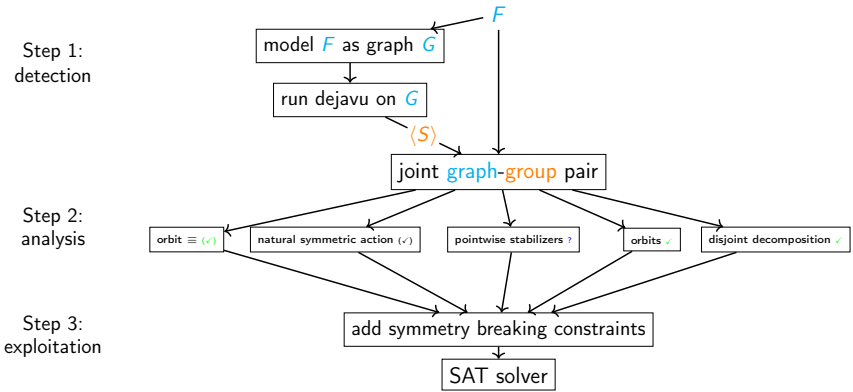
not independent

$\rightarrow$

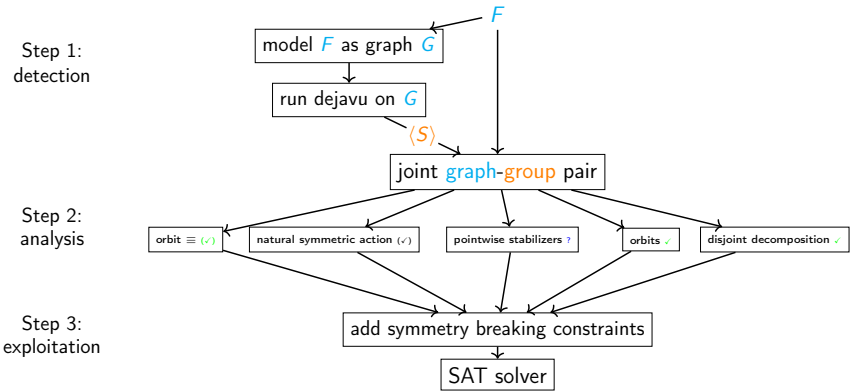
$S = \{$
$(x_1x_2x_3)(\overline{x_1x_2x_3})(y_1y_2y_3)(\overline{y_1y_2y_3}),$
$(x_1x_2)(\overline{x_1x_2})(y_1y_2)(\overline{y_1y_2}),$
$(z_1z_2)(\overline{z_1z_2}),$
$(z_1z_2)(\overline{z_1z_2})\}$

independent

Summary

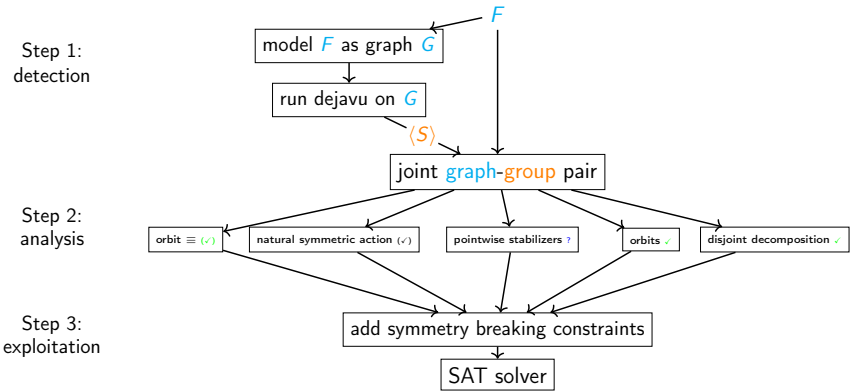

Step 1:
detection
Step 2:
analysis
Step 3:
exploitation
F
model F as graph G
run dejavu on G
⟨S⟩
joint graph-group pair
orbit ≡ (✓)
natural symmetric action (✓)
pointwise stabilizers ?
orbits ✓
disjoint decomposition ✓
add symmetry breaking constraints
SAT solver

Summary



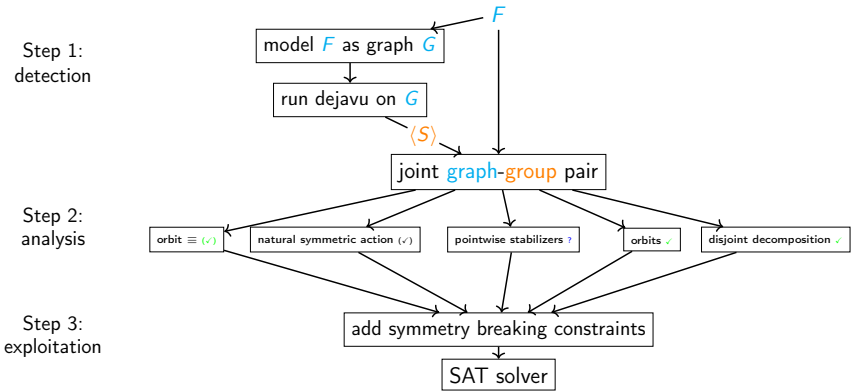
- make use of joint graph-group pairs to get fast, generic algorithms and heuristics

Summary



- make use of joint graph-group pairs to get fast, generic algorithms and heuristics
- work to do to put this in practice (no 1:1 replacement for heuristics!)

Summary



- make use of joint graph-group pairs to get fast, generic algorithms and heuristics
- work to do to put this in practice (no 1:1 replacement for heuristics!)
- want to detect more involved group structures