Certified Symmetry and Dominance Breaking for Combinatorial Optimisation

Bart Bogaerts, **Stephan Gocht**, Ciaran McCreesh, Jakob Nordström

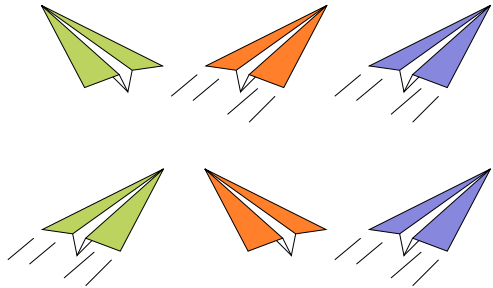
February 2022

Symmetry Breaking — Example

- airline scheduling: have 3 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)

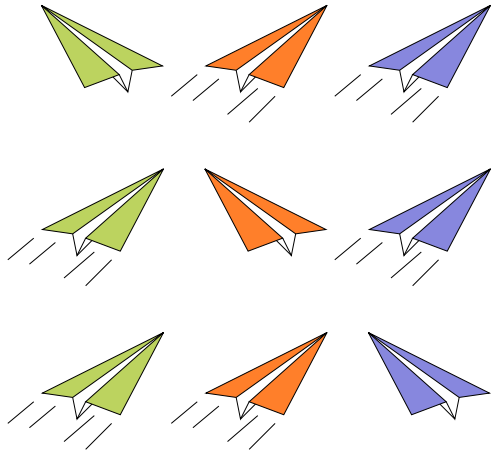
Symmetry Breaking — Example

- airline scheduling: have 3 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)



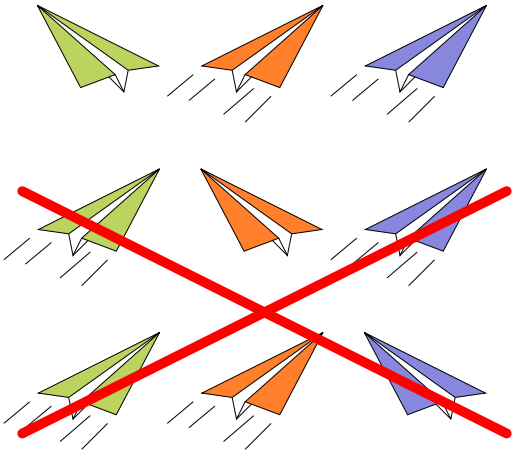
Symmetry Breaking — Example

- airline scheduling: have 3 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)



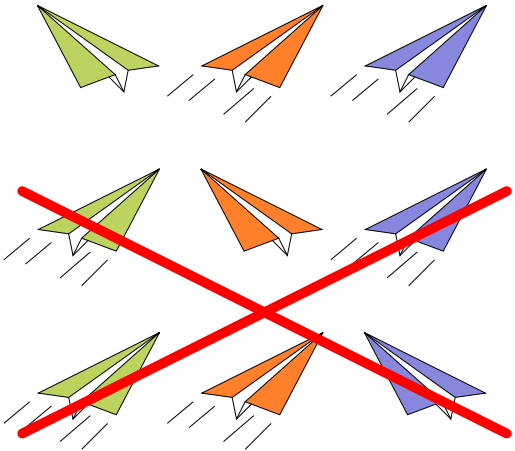
Symmetry Breaking — Example

- airline scheduling: have 3 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)
- idea: make scheduling easier
  by removing symmetric assignments



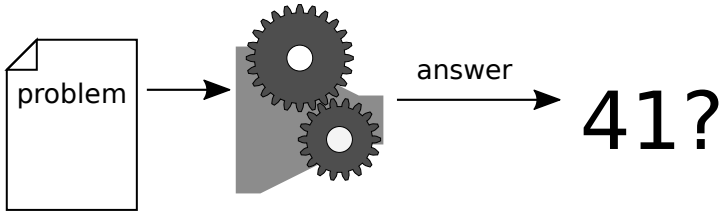
Symmetry Breaking — Example

- airline scheduling: have 3 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)
- idea: make scheduling easier
  by removing symmetric assignments

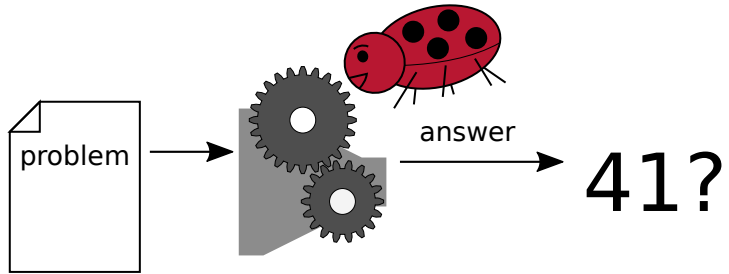


How can we know we didn't remove too many assignments?

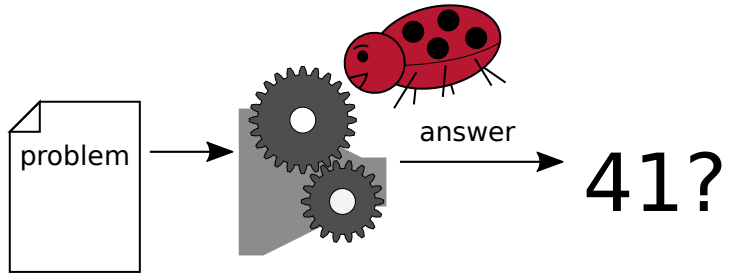
Detecting Bugs with Certifying Algorithms



Detecting Bugs with Certifying Algorithms

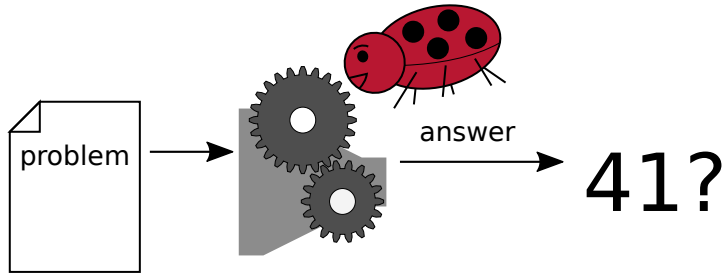


Detecting Bugs with Certifying Algorithms



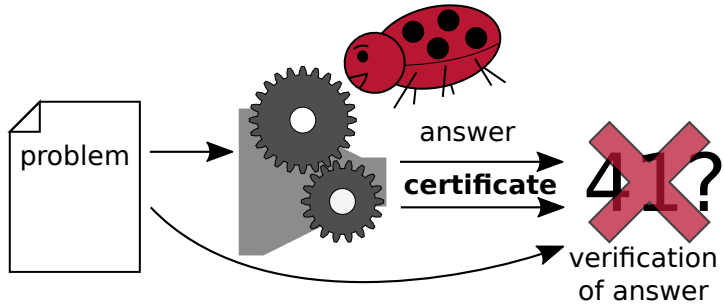
- formally verify solver?
  usually not feasible / too costly

Detecting Bugs with Certifying Algorithms



- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Detecting Bugs with Certifying Algorithms



- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Why Certifying Algorithms?

- during development
  - simplifies testing: no ground truth needed
  - find bugs even if result is correct
  - locate first unsound step

Why Certifying Algorithms?

- during development
  - simplifies testing: no ground truth needed
  - find bugs even if result is correct
  - locate first unsound step
- while solving
  - increase trust in solution
  - detect hardware errors

Why Certifying Algorithms?

- during development
  - simplifies testing: no ground truth needed
  - find bugs even if result is correct
  - locate first unsound step
- while solving
  - increase trust in solution
  - detect hardware errors
- after solving
  - analyse certificate to understand and improve solving process
  - could use certificate to audit solution afterwards

Requirements for Certifying Algorithms

- certificate production
  - easy to implement in any solver
  - small performance overhead

Requirements for Certifying Algorithms

- certificate production
  - easy to implement in any solver
  - small performance overhead

- certificate verification
  - simple, easy to verify steps
  - efficiently machine-verifiable
  - ideally so simple that proof checker can be formally verified

Requirements for Certifying Algorithms

- certificate production
  - easy to implement in any solver
  - small performance overhead

- certificate verification
  - simple, easy to verify steps
  - efficiently machine-verifiable
  - ideally so simple that proof checker can be formally verified

But how?

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)

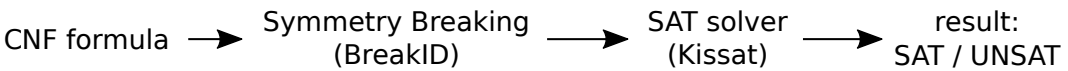
SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CFMSSK17], LRAT [CFHH$^+$17]; DRAT [WHH14] has become standard

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CFMSSK17], LRAT [CFHH$^+$17]; DRAT [WHH14] has become standard

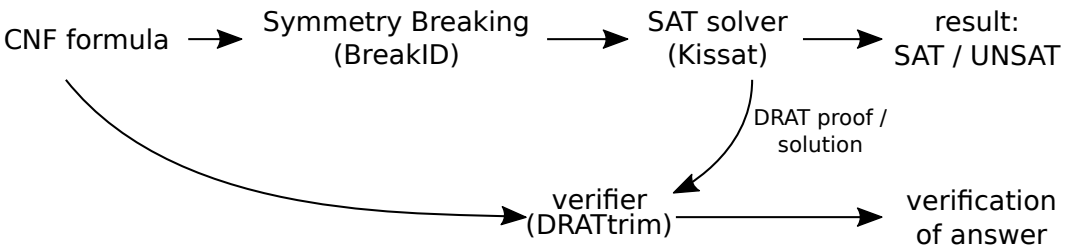
- lets try using SAT technology



SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CFMSSK17], LRAT [CFHH+17]; DRAT [WHH14] has become standard

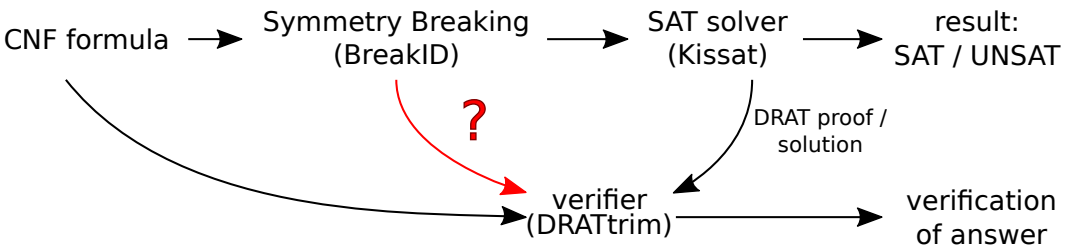
- lets try using SAT technology



SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CFMSSK17], LRAT [CFHH+17]; DRAT [WHH14] has become standard

- lets try using SAT technology



. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - not workable in practice
  - in particular, large or overlapping symmetries are problematic

. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - not workable in practice
  - in particular, large or overlapping symmetries are problematic

- without symmetry breaking $\Rightarrow$ exponential loss in reasoning power / performance

. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - not workable in practice
  - in particular, large or overlapping symmetries are problematic
- without symmetry breaking $\Rightarrow$ exponential loss in reasoning power / performance
- How about practical proof logging for other solving paradigms?
  - MaxSAT solving
  - constraint programming (CP)
  - mixed integer programming (MIP)
  - algebraic reasoning / Gröbner basis computations
  - pseudo-Boolean satisfiability and optimization

. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - not workable in practice
  - in particular, large or overlapping symmetries are problematic
- without symmetry breaking $\Rightarrow$ exponential loss in reasoning power / performance
- How about practical proof logging for other solving paradigms?
  - MaxSAT solving
  - constraint programming (CP)
  - mixed integer programming (MIP)
  - algebraic reasoning / Gröbner basis computations
  - pseudo-Boolean satisfiability and optimization

Need to look beyond DRAT!

New Proof Systems on the Rise

many new proof systems with implemented proof checkers:

- delete symmetry reverse unit propagation (DSRUP) [TD20]
- propagation redundancy (PR) [HKB17]
- practical polynomial calculus (PAC) [RBK18, KFB20]
- propagation redundancy for BDDs [BB21]
- Max-SAT resolution [PCH21]

Our Approach

- general-purpose proof format: pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- allows easy proof logging for
  - reasoning with 0-1 linear inequalities (by design)
  - all-different constraints [EGMN20]
  - subgraph isomorphism [GMN20]
  - clique and maximum common (connected) subgraph [GMM+20]
  - parity/ XOR reasoning [GN21]

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Our Approach

- general-purpose proof format: pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- allows easy proof logging for
  - reasoning with 0-1 linear inequalities (by design)
  - all-different constraints [EGMN20]
  - subgraph isomorphism [GMN20]
  - clique and maximum common (connected) subgraph [GMM+20]
  - parity/ XOR reasoning [GN21]

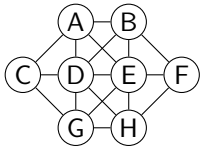
This Work

- proof logging for symmetry and dominance breaking
- applied to SAT, constraint programming and max clique solving
- full support for optimization

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Supported Applications – Symmetry Breaking

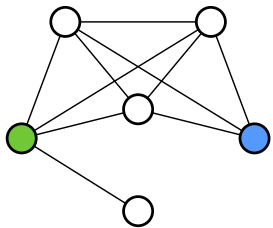
- SAT
  challenge: translate breaking constraint from PB to CNF
- Constraint Programming
  challenge: integer domains instead of 0-1



example: The Crystal Maze puzzle. Place numbers 1 to 8 without repetition, so that adjacent circles do not have consecutive numbers.

Supported Applications – Dominance Breaking

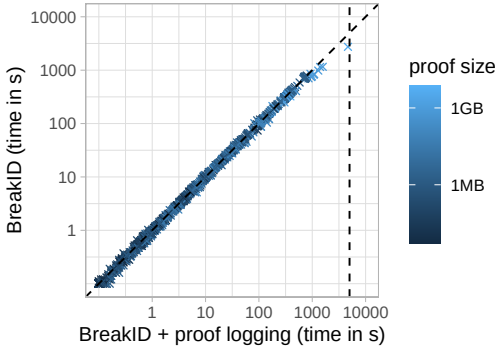
- maximum clique solving (find largest fully connected component)
  challenge: lazy breaking

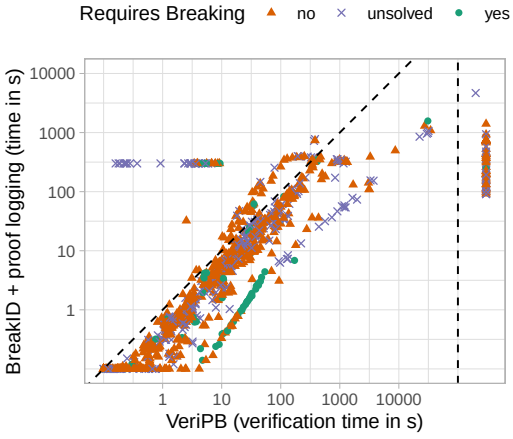


example: consider green but not blue node (every neighbour of blue is also neighbour of green)

Experiments

- evaluated on SAT competition benchmarks
- used BreakID[2] to find and break symmetries





[2] https://bitbucket.org/krr/breakid/

Running Example

$$\begin{aligned} \text{min: } & 4x_1 + 2x_2 + x_3 \\ \text{s.t. } & x_1 + x_2 + x_3 \geq 2 \end{aligned}$$

- boolean variable x is 0 (false) or 1 (true)
  (e.g. $x_1 = 1$ means green aircraft flies)
- pseudo-Boolean constraint:
  linear inequality over variables
- formula F: set of constraints
- objective function f to be minimized

Goal: find assignment minimizing objective and satisfying all constraints

Running Example

$$\text{min: } 4x_1 + 2x_2 + x_3$$
$$\text{s.t. } x_1 + x_2 + x_3 \geq 2$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0$$

(to make solving easier)

- boolean variable x is 0 (false) or 1 (true)
  (e.g. $x_1 = 1$ means green aircraft flies)
- pseudo-Boolean constraint:
  linear inequality over variables
- formula F: set of constraints
- objective function f to be minimized

Goal: find assignment minimizing objective and satisfying all constraints

Running Example

$$\text{min: } 4x_1 + 2x_2 + x_3$$
$$\text{s.t. } x_1 + x_2 + x_3 \geq 2$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0$$

(to make solving easier)

- boolean variable x is 0 (false) or 1 (true)
  (e.g. $x_1 = 1$ means green aircraft flies)
- pseudo-Boolean constraint:
  linear inequality over variables
- formula F: set of constraints
- objective function f to be minimized

Goal: find assignment minimizing objective and satisfying all constraints

How can we know this claim is correct?

Naive Attempt: Proof by Truth Table

$$\text{min: } 4x_1 + 2x_2 + x_3 \quad (1)$$
$$\text{s.t. } x_1 + x_2 + x_3 \geq 2 \quad (2)$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0 \quad (3)$$

(to make solving easier)

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

Naive Attempt: Proof by Truth Table

$$\text{min: } 4x_1 + 2x_2 + x_3 \quad (1)$$

$$\text{s.t. } x_1 + x_2 + x_3 \geq 2 \quad (2)$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0 \quad (3)$$

(to make solving easier)

objective value	x_1	x_2	x_3	
0	0	0	0	falsifies (2)
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	
6	1	1	0	
7	1	1	1	

Naive Attempt: Proof by Truth Table

$$\text{min: } 4x_1 + 2x_2 + x_3 \quad (1)$$

$$\text{s.t. } x_1 + x_2 + x_3 \geq 2 \quad (2)$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0 \quad (3)$$

(to make solving easier)

objective value	x_1	x_2	x_3	
0	0	0	0	falsifies (2)
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	optimum
5	1	0	1	
6	1	1	0	
7	1	1	1	

Naive Attempt: Proof by Truth Table

$$\text{min: } 4x_1 + 2x_2 + x_3 \quad (1)$$

$$\text{s.t. } x_1 + x_2 + x_3 \geq 2 \quad (2)$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0 \quad (3)$$

(to make solving easier)

objective value	x_1	x_2	x_3	
0	0	0	0	falsifies (2)
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	optimum
5	1	0	1	falsifies (3)
6	1	1	0	
7	1	1	1	

Naive Attempt: Proof by Truth Table

$$\text{min: } 4x_1 + 2x_2 + x_3 \quad (1)$$
$$\text{s.t. } x_1 + x_2 + x_3 \geq 2 \quad (2)$$

Claim:

- without loss of generality, add

$$3x_1 - 2x_2 - x_3 \leq 0 \quad (3)$$

(to make solving easier)

objective value	x_1	x_2	x_3	
0	0	0	0	falsifies (2)
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	optimum
5	1	0	1	falsifies (3)
6	1	1	0	
7	1	1	1	

Not practical, need something more efficient!

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker
- output:
  - for permutation
    $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	π
4	1	0	0	
3	0	1	1	
5	1	0	1	π
6	1	1	0	
7	1	1	1	

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker
- output:
  - for permutation
    $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
  - (syntactic) symmetry detected (ignoring objective)
    $(x_1 + x_2 + x_3 \geq 2)_{\restriction \pi} = x_2 + x_3 + x_1 \geq 2$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	π
4	1	0	0	
3	0	1	1	
5	1	0	1	π
6	1	1	0	
7	1	1	1	

Notation: $C_{\restriction \pi}$ substitutes variables in C as specified by π

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker
- output:
  - for permutation
    $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
  - (syntactic) symmetry detected (ignoring objective)
    $(x_1 + x_2 + x_3 \geq 2)_{\restriction \pi} = x_2 + x_3 + x_1 \geq 2$
  - breaking constraint
    $3x_1 - 2x_2 - x_3 \leq 0$ (falsified by purple assignments)

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	π
4	1	0	0	
3	0	1	1	
5	1	0	1	π
6	1	1	0	
7	1	1	1	

Notation: $C_{\restriction \pi}$ substitutes variables in C as specified by π

The Dominance Rule

- idea (from dominance breaking):
  - allow to add constraint C, e.g, $3x_1 - 2x_2 - x_3 \leq 0$
  - if for every ρ falsifying C but satisfying F (purple)
  - we find ρ' that satisfies F and $f(\rho) > f(\rho')$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	)
6	1	1	0	)
7	1	1	1	

The Dominance Rule

- idea (from dominance breaking):
  - allow to add constraint C, e.g, $3x_1 - 2x_2 - x_3 \leq 0$
  - if for every ρ falsifying C but satisfying F (purple)
  - we find ρ' that satisfies F and $f(\rho) > f(\rho')$

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{ \neg C \} \models F_{\restriction \omega} \cup \{ f > f_{\restriction \omega} \}$$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	↻
6	1	1	0	↻
7	1	1	1	

Notation: $F \models F'$: satisfying assignment to F is also satisfying assignment to F'

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- to prove: $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \geq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- to prove: $F \cup \{ \neg C \} \models F_{\restriction \omega} \cup \{ f > f_{\restriction \omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \geq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- to prove: $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive $(x_1 + x_2 + x_3 \geq 2)_{\restriction\omega}$
  $\equiv \quad x_2 + x_3 + x_1 \geq 2$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- to prove: $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive $(x_1 + x_2 + x_3 \geq 2)_{\restriction\omega}$
  $\equiv \quad x_2 + x_3 + x_1 \geq 2$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- to prove: $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive
$$(x_1 + x_2 + x_3 \geq 2)_{\restriction\omega}$$
$$\equiv \quad x_2 + x_3 + x_1 \geq 2$$
- and
$$4x_1 + 2x_2 + x_3 > (4x_1 + 2x_2 + x_3)_{\restriction\omega}$$
$$\equiv 4x_1 + 2x_2 + x_3 > \quad 4x_2 + 2x_3 + x_1$$
$$\equiv 3x_1 - 2x_2 - x_3 > 0$$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \geq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- to prove: $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive $(x_1 + x_2 + x_3 \geq 2)_{\restriction\omega}$
  $\equiv \quad x_2 + x_3 + x_1 \geq 2$
- and $4x_1 + 2x_2 + x_3 > (4x_1 + 2x_2 + x_3)_{\restriction\omega}$
  $\equiv 4x_1 + 2x_2 + x_3 > \quad 4x_2 + 2x_3 + x_1$
  $\equiv 3x_1 - 2x_2 - x_3 > 0$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Further Remarks

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{ \neg C \} \models F_{|\omega} \cup \{ f > f_{|\omega} \}$$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Further Remarks

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{ \neg C \} \models F_{\restriction \omega} \cup \{ f > f_{\restriction \omega} \}$$

- "$\models$" replaced by **efficiently machine-verifiable** proof system (cutting planes)

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Further Remarks

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{ \neg C \} \models F_{\restriction \omega} \cup \{ f > f_{\restriction \omega} \}$$

- "$\models$" replaced by **efficiently machine-verifiable** proof system (cutting planes)
- in paper: any strict order instead of $f > f_{\restriction \omega}$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

Conclusion

- proof logging is well-established standard for SAT solving
- so far, prohibitively expensive for some techniques (e.g. symmetry breaking)

3

4

Conclusion

- proof logging is well-established standard for SAT solving
- so far, prohibitively expensive for some techniques (e.g. symmetry breaking)

our work: proof logging for symmetry breaking (BreakID[3]) + verification (VeriPB[4])

- simple to implement + efficient proof checking
- fully evaluated for symmetry breaking on SAT competition benchmarks
- proof of concept for
  - symmetry breaking in constraint programming
  - dynamic dominance breaking for maximum clique

[3] `https://bitbucket.org/krr/breakid/`

[4] `https://gitlab.com/MIAOresearch/VeriPB`

Conclusion

- proof logging is well-established standard for SAT solving
- so far, prohibitively expensive for some techniques (e.g. symmetry breaking)

our work: proof logging for symmetry breaking (BreakID[3]) + verification (VeriPB[4])

- simple to implement + efficient proof checking
- fully evaluated for symmetry breaking on SAT competition benchmarks
- proof of concept for
  - symmetry breaking in constraint programming
  - dynamic dominance breaking for maximum clique

future work:

- proof logging for more techniques (e.g., in MaxSAT, CP or MIP)
- formally verified verifier

[3] `https://bitbucket.org/krr/breakid/`

[4] `https://gitlab.com/MIAOresearch/VeriPB`

References I

[BB21] Lee A. Barnett and Armin Biere.
Non-clausal redundancy properties.
In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 252–272. Springer, 2021.

[Bie06] Armin Biere.
Tracecheck.
`http://fmv.jku.at/tracecheck/`, 2006.

[CFHH+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp.
Efficient certified rat verification.
In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.

[CFMSSK17] Luís Cruz-Filipe, Joao Marques-Silva, and Peter Schneider-Kamp.
Efficient certified resolution proof checking.
In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 118–135. Springer Berlin Heidelberg, 2017.

References II

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström.
Justifying all differences using pseudo-boolean reasoning.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1486–1494. AAAI Press, 2020.

[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble.
Certifying solvers for clique and maximum common (connected) subgraph problems.
In Helmut Simonis, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, 2020.

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström.
Subgraph isomorphism meets cutting planes: Solving with certified solutions.
In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1134–1140, 2020.

[GN03] Evguenii I. Goldberg and Yakov Novikov.
Verification of proofs of unsatisfiability for CNF formulas.
In *Design, Automation and Test in Europe Conference (DATE)*, pages 10886–10891. IEEE Computer Society, 2003.

References III

[GN21] Stephan Gocht and Jakob Nordström.
Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3768–3777, 2021.

[HHW15] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler.
Expressing symmetry breaking in DRAT proofs.
In *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, August 2015.

[HKB17] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere.
Short proofs without new variables.
In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2017.

[KFB20] Daniela Kaufmann, Mathias Fleury, and Armin Biere.
The proof checkers pacheck and pastÃ¨que for the practical algebraic calculus.
In Ofer Strichman and Alexander Ivrii, editors, *Formal Methods in Computer-Aided Design, FMCAD 2020.*, volume 1, pages 264–269. TU Vienna Academic Press, 2020.

References IV

[PCH21] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet.
A proof builder for max-sat.
In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 488–498, Cham, 2021. Springer International Publishing.

[RBK18] Daniela Ritirc, Armin Biere, and Manuel Kauers.
A practical polynomial calculus for arithmetic circuit verification.
In Anna M. Bigatti and Martin Brain, editors, *3rd International Workshop on Satisfiability Checking and Symbolic Computation (SC2'18)*, pages 61–76. CEUR-WS, 2018.

[TD20] Rodrigue Konan Tchinda and Clémentin Tayou Djamégni.
On certifying the UNSAT result of dynamic symmetry-handling-based SAT solvers.
Constraints An Int. J., 25(3-4):251–279, 2020.

[WHH14] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr.
DRAT-trim: Efficient checking and trimming using expressive clausal proofs.
In *Proceedings of the 17th Internatjuional Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.