Certified Symmetry and Dominance Breaking for Combinatorial Optimisation

Jakob Nordström
University of Copenhagen and Lund University

Swedish Operations Research Conference
Stockholm, Sweden
October 24, 2022

Joint work with Bart Bogaerts, Stephan Gocht, and Ciaran McCreesh
Combinatorial Solving and Optimisation

- Revolution last couple of decades in \textit{combinatorial solvers} for
  - Boolean satisfiability (SAT) solving [BHvMW21]*
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]

*See end of slides for all references with bibliographic details
Combinatorial Solving and Optimisation

- Revolution last couple of decades in **combinatorial solvers** for
  - Boolean satisfiability (SAT) solving [BHvMW21]*
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]

- Solve NP problems (or worse) very successfully in practice!

*See end of slides for all references with bibliographic details*
Combinatorial Solving and Optimisation

- Revolution last couple of decades in combinatorial solvers for
  - Boolean satisfiability (SAT) solving [BHvMW21]*
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]

- Solve NP problems (or worse) very successfully in practice!

- Except solvers are sometimes wrong… (Even best commercial ones) [BLB10, CKSW13, AGJ+18, GSD19, GS19]

*See end of slides for all references with bibliographic details
Combinatorial Solving and Optimisation

- Revolution last couple of decades in **combinatorial solvers** for
  - Boolean satisfiability (SAT) solving [BHvMW21]*
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]

- Solve NP problems (or worse) very successfully in practice!

- **Except solvers are sometimes wrong...** (Even best commercial ones) [BLB10, CKSW13, AGJ+18, GSD19, GS19]

- **Software testing** doesn’t suffice to resolve this problem

*See end of slides for all references with bibliographic details*
Combinatorial Solving and Optimisation

- Revolution last couple of decades in **combinatorial solvers** for
  - Boolean satisfiability (SAT) solving [BHvMW21]*
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]

- Solve NP problems (or worse) very successfully in practice!

- Except solvers are sometimes wrong... (Even best commercial ones) [BLB10, CKSW13, AGJ+18, GSD19, GS19]

- **Software testing** doesn’t suffice to resolve this problem

- **Formal verification** techniques cannot deal with level of complexity of modern solvers

---

*See end of slides for all references with bibliographic details*
Certified Results with Proof Logging

Solution: Design certifying algorithms [ABM⁺11, MMNS11] that
- not only solve problem but also
- do proof logging to certify that result is correct
Certified Results with Proof Logging

Solution: Design certifying algorithms \([\text{ABM}^+11, \text{MMNS}11]\) that
- not only solve problem but also
- do proof logging to certify that result is correct

Workflow:
1. Run solver on problem input
Certified Results with Proof Logging

Solution: Design **certifying algorithms** \([ABM^+11, MMNS11]\) that
- not only **solve problem** but also
- do **proof logging** to certify that result is correct

**Workflow:**
1. Run solver on problem input
2. Get as output not only result but also proof
Certified Results with Proof Logging

Solution: Design certifying algorithms [ABM$^+$11, MMNS11] that
- not only solve problem but also
- do proof logging to certify that result is correct

Workflow:
1. Run solver on problem input
2. Get as output not only result but also proof
3. Feed input + result + proof to proof checker
Certified Results with Proof Logging

Solution: Design certifying algorithms [ABM⁺11, MMNS11] that

- not only solve problem but also
- do proof logging to certify that result is correct

Workflow:
1. Run solver on problem input
2. Get as output not only result but also proof
3. Feed input + result + proof to proof checker
4. Verify that proof checker says result is correct
Yet Another SAT Success Story

Many proof logging formats for SAT solving using CNF clausal format:

- DRAT [HHW13a, HHW13b, WHH14]
- GRIT [CMS17]
- LRAT [CHH+17]
- …

Well established — required in main track of SAT competitions
Crucial for unsatisfiable formulas
Yet Another SAT Success Story

Many proof logging formats for SAT solving using CNF clausal format:
- DRAT [HHW13a, HHW13b, WHH14]
- GRIT [CMS17]
- LRAT [CHH+17]

... 

Well established — required in main track of SAT competitions
Crucial for unsatisfiable formulas

But efficient proof logging has remained out of reach for stronger paradigms
Yet Another SAT Success Story(?)

Many proof logging formats for SAT solving using CNF clausal format:

- DRAT [HHW13a, HHW13b, WHH14]
- GRIT [CMS17]
- LRAT [CHH+17]
- ...

Well established — required in main track of SAT competitions
Crucial for unsatisfiable formulas

But efficient proof logging has remained out of reach for stronger paradigms

And, in fact, even for some advanced SAT solving techniques:

- cardinality reasoning
- Gaussian elimination
- symmetry handling
Our Work: Efficient Proof Logging for Symmetry Breaking

Paper *Certified Symmetry and Dominance Breaking for Combinatorial Optimisation* at AAAI ’22 [BGMN22]:

Implementation in proof checker VeriPB [Ver]
Our Work: Efficient Proof Logging for Symmetry Breaking

Paper *Certified Symmetry and Dominance Breaking for Combinatorial Optimisation* at AAAI ’22 [BGMN22]:

Implementation in proof checker VERIPB [Ver]

- First general & efficient proof logging method for *symmetry breaking*
Our Work: Efficient Proof Logging for Symmetry Breaking

Paper *Certified Symmetry and Dominance Breaking for Combinatorial Optimisation* at AAAI ’22 [BGMN22]:

Implementation in proof checker VERIPB [Ver]

- First general & efficient proof logging method for symmetry breaking
- Supports also pseudo-Boolean reasoning and Gaussian elimination
Our Work: Efficient Proof Logging for Symmetry Breaking

Paper *Certified Symmetry and Dominance Breaking for Combinatorial Optimisation* at AAAI ’22 [BGMN22]:

Implementation in proof checker **VeriPB** [Ver]

- First general & efficient proof logging method for symmetry breaking
- Supports also pseudo-Boolean reasoning and Gaussian elimination
- Based on 0-1 integer linear constraints instead of clauses
Our Work: Efficient Proof Logging for Symmetry Breaking

Paper *Certified Symmetry and Dominance Breaking for Combinatorial Optimisation* at *AAAI ’22* [BGMN22]:

Implementation in proof checker \textsc{VeriPB} [Ver]

- First general & efficient proof logging method for symmetry breaking
- Supports also pseudo-Boolean reasoning and Gaussian elimination
- Based on 0-1 integer linear constraints instead of clauses
- Uses cutting planes method [CCT87] with additional rules
Outline of Presentation

What I hope to cover in the rest of this presentation:

- Basics of proof logging with 0-1 linear constraints
- New rule for symmetry and dominance breaking
- Application to symmetry breaking for SAT solving (also other applications, but focus here on SAT)
- Some future research directions
Outline of Presentation

What I hope to cover in the rest of this presentation:

- Basics of proof logging with 0-1 linear constraints
- New rule for symmetry and dominance breaking
- Application to symmetry breaking for SAT solving (also other applications, but focus here on SAT)
- Some future research directions

Caveat: Only exact problems in this talk but:
- This is already very challenging
- Ideas seem likely to generalize
Pseudo-Boolean (PB) constraints are 0-1 integer linear constraints

\[ C \doteq \sum_i a_i \ell_i \geq A \]

- \( a_i, A \in \mathbb{Z} \)
- literals \( \ell_i: x_i \) or \( \overline{x}_i \) (where \( x_i + \overline{x}_i = 1 \))
- variables \( x_i \) take values \( 0 = false \) or \( 1 = true \)
0-1 Integer Linear (a.k.a. Pseudo-Boolean) Constraints

Pseudo-Boolean (PB) constraints are 0-1 integer linear constraints

\[ C \equiv \sum_{i} a_i \ell_i \geq A \]

- \( a_i, A \in \mathbb{Z} \)
- literals \( \ell_i: x_i \) or \( \overline{x_i} \) (where \( x_i + \overline{x_i} = 1 \))
- variables \( x_i \) take values \( 0 = \text{false} \) or \( 1 = \text{true} \)

Negation of constraint

\[ \neg C \equiv \sum_{i} a_i \ell_i \leq A - 1 \]
Pseudo-Boolean (PB) constraints are 0-1 integer linear constraints

\[ C \doteq \sum_i a_i \ell_i \geq A \]

- \( a_i, A \in \mathbb{Z} \)
- literals \( \ell_i: x_i \) or \( \overline{x_i} \) (where \( x_i + \overline{x_i} = 1 \))
- variables \( x_i \) take values \( 0 = false \) or \( 1 = true \)

Negation of constraint

\[ \neg C \doteq \sum_i a_i \ell_i \leq A - 1 \]

Pseudo-Boolean formulas \( F \doteq \bigwedge_{i=1}^m C_i \) are conjunctions of pseudo-Boolean constraints (a.k.a. 0-1 integer linear programs)
Some Types of Pseudo-Boolean Constraints

1. Clauses

\[ x \lor \bar{y} \lor z \iff x + \bar{y} + z \geq 1 \]
Some Types of Pseudo-Boolean Constraints

1. **Clauses**

\[ x \lor \bar{y} \lor z \iff x + \bar{y} + z \geq 1 \]

2. **Cardinality constraints**

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]
Some Types of Pseudo-Boolean Constraints

1. Clauses

\[ x \lor \overline{y} \lor z \iff x + \overline{y} + z \geq 1 \]

2. Cardinality constraints

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]

3. General pseudo-Boolean constraints

\[ x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**

\[ \ell_i \geq 0 \]

**Linear combination**

\[
\begin{align*}
\sum_i a_i \ell_i &\geq A \\
\sum_i b_i \ell_i &\geq B \\
\sum_i (c_A a_i + c_B b_i) \ell_i &\geq c_A A + c_B B \\
\end{align*}
\]

\([c_A, c_B \in \mathbb{N}]\)

**Division**

\[
\begin{align*}
\sum_i c a_i \ell_i &\geq A \\
\sum_i a_i \ell_i &\geq \lceil A/c \rceil \\
\end{align*}
\]

\([c \in \mathbb{N}^+]\)
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

### Literal axioms
\[ \ell_i \geq 0 \]

### Linear combination
\[
\begin{align*}
\sum_i a_i \ell_i &\geq A \\
\sum_i b_i \ell_i &\geq B \\
\sum_i (c_A a_i + c_B b_i) \ell_i &\geq c_A A + c_B B
\end{align*}
\]
\([c_A, c_B \in \mathbb{N}]\)

### Division
\[
\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil}
\]
\([c \in \mathbb{N}^+]\)

### Toy example:

| Lin comb | \(2x + 4y + 2z + w \geq 5\) | \(2x + y + w \geq 2\) |
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**
\[ \ell_i \geq 0 \]

**Linear combination**
\[
\frac{\sum_i a_i \ell_i \geq A}{\sum_i b_i \ell_i \geq B} \quad \sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B
\]
\[ [c_A, c_B \in \mathbb{N}] \]

**Division**
\[
\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+] \]

**Toy example:**
Lin comb
\[ 2x + 4y + 2z + w \geq 5 \quad 2x + y + w \geq 2 \]
\[ (2x + 4y + 2z + w) + 2 \cdot (2x + y + w) \geq 5 + 2 \cdot 2 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Literal axioms \( \ell_i \geq 0 \)

Linear combination

\[
\sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \\
\sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B 
\]

[\(c_A, c_B \in \mathbb{N}\)]

Division

\[
\sum_i c a_i \ell_i \geq A \\
\sum_i a_i \ell_i \geq \lceil A/c \rceil 
\]

[c \in \mathbb{N}^+] 

Toy example:

Lin comb

\[
2x + 4y + 2z + w \geq 5 \\
6x + 6y + 2z + 3w \geq 9
\]

\[
2x + y + w \geq 2
\]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**
\[ l_i \geq 0 \]

**Linear combination**
\[ \sum_i a_i l_i \geq A \quad \sum_i b_i l_i \geq B \]
\[ \sum_i (c_A a_i + c_B b_i) l_i \geq c_A A + c_B B \]
\[ [c_A, c_B \in \mathbb{N}] \]

**Division**
\[ \frac{\sum_i c a_i l_i}{\sum_i a_i l_i} \geq \left\lceil \frac{A}{c} \right\rceil \]
\[ [c \in \mathbb{N}^+] \]

**Toy example:**

**Lin comb**
\[ 2x + 4y + 2z + w \geq 5 \quad 2x + y + w \geq 2 \]
\[ 6x + 6y + 2z + 3w \geq 9 \quad \bar{z} \geq 0 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**

\[ l_i \geq 0 \]

**Linear combination**

\[
\begin{align*}
\sum_i a_i l_i &\geq A \\
\sum_i b_i l_i &\geq B \\
\sum_i (c_A a_i + c_B b_i) l_i &\geq c_A A + c_B B
\end{align*}
\]

\([c_A, c_B \in \mathbb{N}]\)

**Division**

\[
\begin{align*}
\sum_i c a_i l_i &\geq A \\
\sum_i c a_i l_i &\geq \lceil A/c \rceil
\end{align*}
\]

\([c \in \mathbb{N}^+]\)

**Toy example:**

Lin comb

\[
\begin{align*}
2x + 4y + 2z + w &\geq 5 \\
2x + y + w &\geq 2 \\
6x + 6y + 2z + 3w &\geq 9 \\
6x + 6y + 2z + 3w + 2 \cdot \bar{z} &\geq 9 + 2 \cdot 0
\end{align*}
\]
Pseudo-Boolean Reasoning: Cutting Planes \([\text{CCT87}]\)

**Literal axioms**
\[ \ell_i \geq 0 \]

**Linear combination**
\[ \sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \]
\[ \sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B \]
\([c_A, c_B \in \mathbb{N}]\)

**Division**
\[ \sum_i c a_i \ell_i \geq A \]
\[ \sum_i a_i \ell_i \geq \lceil A/c \rceil \]
\([c \in \mathbb{N}^+]\)

**Toy example:**
Lin comb
\[ 2x + 4y + 2z + w \geq 5 \quad 2x + y + w \geq 2 \]
Lin comb
\[ 6x + 6y + 2z + 3w \geq 9 \]
Lin comb
\[ 6x + 6y + 3w + 2 \geq 9 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**
\[ \ell_i \geq 0 \]

**Linear combination**
\[ \sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \]
\[ \sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B \]
\[ [c_A, c_B \in \mathbb{N}] \]

**Division**
\[ \sum_i c a_i \ell_i \geq A \]
\[ \sum_i a_i \ell_i \geq \lceil A/c \rceil \]
\[ [c \in \mathbb{N}^+] \]

**Toy example:**

Lin comb
\[ 2x + 4y + 2z + w \geq 5 \quad 2x + y + w \geq 2 \]
\[ 6x + 6y + 2z + 3w \geq 9 \]
\[ 6x + 6y + 3w \geq 7 \]

Lin comb
\[ \bar{z} \geq 0 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**
\[ l_i \geq 0 \]

**Linear combination**
\[
\begin{align*}
\sum_i a_i l_i & \geq A \\
\sum_i b_i l_i & \geq B
\end{align*}
\]
\[
\sum_i (c_A a_i + c_B b_i) l_i \geq c_A A + c_B B
\]
\[ [c_A, c_B \in \mathbb{N}] \]

**Division**
\[
\begin{align*}
\sum_i c a_i l_i & \geq A \\
\sum_i a_i l_i & \geq \lceil A/c \rceil
\end{align*}
\]
\[ [c \in \mathbb{N}^+] \]

**Toy example:**
\[
\begin{align*}
\text{Lin comb} & \quad 2x + 4y + 2z + w \geq 5 \\
\text{Lin comb} & \quad 6x + 6y + 2z + 3w \geq 9
\end{align*}
\]
\[
\begin{align*}
\text{Lin comb} & \quad 2x + y + w \geq 2 \\
\text{Lin comb} & \quad 6x + 6y + 3w \geq 9
\end{align*}
\]
\[
\begin{align*}
\text{Div} & \quad 2x + 2y + w \geq 2 \frac{1}{3}
\end{align*}
\]
\[ z \geq 0 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Literal axioms**
\[ l_i \geq 0 \]

**Linear combination**
\[
\sum_i a_i l_i \geq A \\
\sum_i b_i l_i \geq B \\
\sum_i (c_A a_i + c_B b_i) l_i \geq c_A A + c_B B
\]
\[ [c_A, c_B \in \mathbb{N}] \]

**Division**
\[
\sum_i c a_i l_i \geq A \\
\sum_i a_i l_i \geq \lceil A/c \rceil
\]
\[ [c \in \mathbb{N}^+] \]

**Toy example:**

Lin comb
\[
2x + 4y + 2z + w \geq 5 \\
6x + 6y + 2z + 3w \geq 9
\]
Lin comb
\[
2x + y + w \geq 2 \\
6x + 6y + 3w \geq 7
\]
Div
\[
2x + 2y + w \geq 3
\]
\[ z \geq 0 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Literal axioms
\[ \ell_i \geq 0 \]

Linear combination
\[ \sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \]
\[ \sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B \]
\[ [c_A, c_B \in \mathbb{N}] \]

Division
\[ \sum_i c \ell_i \geq \lceil A/c \rceil \]
\[ \sum_i a_i \ell_i \geq \lceil A/c \rceil \]
\[ [c \in \mathbb{N}^+] \]

Toy example:
\[
\begin{align*}
\text{Lin comb} & \quad 2x + 4y + 2z + w \geq 5 & 2x + y + w \geq 2 \\
\text{Lin comb} & \quad 6x + 6y + 2z + 3w \geq 9 & 6x + 6y + 3w \geq 7 \\
\text{Div} & \quad 2x + 2y + w \geq 3 & z \geq 0
\end{align*}
\]

(See [BN21] for more details about cutting planes)
Proof Logging for SAT with Pseudo-Boolean Reasoning

- View clauses as pseudo-Boolean constraints
Proof Logging for SAT with Pseudo-Boolean Reasoning

- View clauses as pseudo-Boolean constraints
- Operate on constraints with cutting planes rules
Proof Logging for SAT with Pseudo-Boolean Reasoning

- View clauses as pseudo-Boolean constraints
- Operate on constraints with cutting planes rules
- Prove unsatisfiability by deriving $0 \geq 1$
Proof Logging for SAT with Pseudo-Boolean Reasoning

- View clauses as pseudo-Boolean constraints
- Operate on constraints with cutting planes rules
- Prove unsatisfiability by deriving $0 \geq 1$
- **Fact:** Fully sufficient for proof logging for so-called conflict-driven clause learning [BS97, MS99, MMZ$^+$01]
Proof Logging for SAT with Pseudo-Boolean Reasoning

- View clauses as pseudo-Boolean constraints
- Operate on constraints with cutting planes rules
- Prove unsatisfiability by deriving $0 \geq 1$
- **Fact:** Fully sufficient for proof logging for so-called conflict-driven clause learning [BS97, MS99, MMZ+01]
- Also need **extension** rule (analogue of RAT [JHB12] used in SAT proof logging) to deal with, e.g., preprocessing/presolving
Extension Rule: Redundance-Based Strengthening

$C$ is **redundant** with respect to $F$ if $F$ and $F \land C$ are **equisatisfiable**

Want to allow adding redundant constraints
Extension Rule: Redundance-Based Strengthening

$C$ is redundant with respect to $F$ if $F$ and $F \land C$ are equisatisfiable.

Want to allow adding redundant constraints.

Redundance-based strengthening [BT19, GN21]

$C$ is redundant with respect to $F$ if and only if there is a substitution $\omega$ (mapping variables to truth values or literals), called a witness, for which

$$F \land \neg C \models (F \land C) \upharpoonright \omega$$
Extension Rule: Redundance-Based Strengthening

*C* is redundant with respect to *F* if *F* and *F ∧ C* are equisatisfiable

Want to allow adding redundant constraints

Redundance-based strengthening [BT19, GN21]

*C* is redundant with respect to *F* if and only if there is a substitution *ω* (mapping variables to truth values or literals), called a witness, for which

\[ F ∧ ¬C \models (F ∧ C)↾ω \]

- Proof sketch for interesting direction: If *α* satisfies *F* but falsifies *C*, then *α ◦ ω* satisfies *F ∧ C*
Extension Rule: Redundance-Based Strengthening

$C$ is redundant with respect to $F$ if $F$ and $F \land C$ are equisatisfiable
Want to allow adding redundant constraints

Redundance-based strengthening [BT19, GN21]

$C$ is redundant with respect to $F$ if and only if there is a substitution $\omega$ (mapping variables to truth values or literals), called a witness, for which

$$F \land \neg C \models (F \land C)^\omega$$

- Proof sketch for interesting direction: If $\alpha$ satisfies $F$ but falsifies $C$, then $\alpha \circ \omega$ satisfies $F \land C$
- Witness $\omega$ should be specified and implication efficiently verifiable by very simple checks (technical details omitted)
The Power of Proof Logging with Extended Cutting Planes

0-1 linear inequalities convenient to capture SAT reasoning (with clauses)
0-1 linear inequalities convenient to capture SAT reasoning (with clauses)

And yields efficient proof logging for wider range of problems/algorithms:
- Pre- and inprocessing [GN21] (since redundancy rule subsumes RAT)
- pseudo-Boolean reasoning (by design)
- Gaussian elimination [GN21]
The Power of Proof Logging with Extended Cutting Planes

0-1 linear inequalities convenient to capture SAT reasoning (with clauses)
And yields efficient proof logging for wider range of problems/algorithms:
- Pre- and inprocessing [GN21] (since redundancy rule subsumes RAT)
- pseudo-Boolean reasoning (by design)
- Gaussian elimination [GN21]
- subgraph problems [GMN20, GMM+20]
The Power of Proof Logging with Extended Cutting Planes

0-1 linear inequalities convenient to capture SAT reasoning (with clauses)

And yields efficient proof logging for wider range of problems/algorithms:
- Pre- and inprocessing [GN21] (since redundance rule subsumes RAT)
- pseudo-Boolean reasoning (by design)
- Gaussian elimination [GN21]
- subgraph problems [GMN20, GMM+20]
- solving pseudo-Boolean formulas via translation to CNF [GMNO22]
The Power of Proof Logging with Extended Cutting Planes

0-1 linear inequalities convenient to capture SAT reasoning (with clauses)

And yields efficient proof logging for wider range of problems/algorithms:
- Pre- and inprocessing [GN21] (since redundancy rule subsumes RAT)
- pseudo-Boolean reasoning (by design)
- Gaussian elimination [GN21]
- subgraph problems [GMN20, GMM+20]
- solving pseudo-Boolean formulas via translation to CNF [GMNO22]
- (basic) constraint programming [EGMN20, GMN22]
The Power of Proof Logging with Extended Cutting Planes

0-1 linear inequalities convenient to capture SAT reasoning (with clauses)

And yields efficient proof logging for wider range of problems/algorithms:

- Pre- and inprocessing [GN21] (since redundancy rule subsumes RAT)
- pseudo-Boolean reasoning (by design)
- Gaussian elimination [GN21]
- subgraph problems [GMN20, GMM+20]
- solving pseudo-Boolean formulas via translation to CNF [GMNO22]
- (basic) constraint programming [EGMN20, GMN22]
- **This talk:** extend to symmetry and dominance breaking [BGMN22]
The Power of Proof Logging with Extended Cutting Planes

0-1 linear inequalities convenient to capture SAT reasoning (with clauses)

And yields efficient proof logging for wider range of problems/algorithms:
- Pre- and inprocessing [GN21] (since redundance rule subsumes RAT)
- pseudo-Boolean reasoning (by design)
- Gaussian elimination [GN21]
- subgraph problems [GMN20, GMM+20]
- solving pseudo-Boolean formulas via translation to CNF [GMNO22]
- (basic) constraint programming [EGMN20, GMN22]
- **This talk**: extend to symmetry and dominance breaking [BGMN22]

Zoom tutorial on all of these developments **Mon Nov 28 at 14:00 CET**

*Combinatorial Solving with Provably Correct Results*

See [http://www.jakobnordstrom.se/miao-seminars](http://www.jakobnordstrom.se/miao-seminars)
The Challenge of Symmetries

(Syntactic) symmetry: substitution $\sigma$ preserving $F$ ($F^\sigma \equiv F'$)

- Show up in some hard SAT benchmarks
- Can play crucial role in CP and MIP problems [AW13, GSVW14]
The Challenge of Symmetries

(Syntactic) symmetry: substitution $\sigma$ preserving $F$ ($F\upharpoonright_\sigma \equiv F'$)

- Show up in some hard SAT benchmarks
- Can play crucial role in CP and MIP problems [AW13, GSVW14]

Symmetry breaking in SAT
Add constraints filtering out symmetric solutions [ASM06, DBBD16]

Symmetric learning in SAT
Allow to add all symmetric versions of learned constraint [DBB17]
The Challenge of Symmetries

(Syntactic) symmetry: substitution $\sigma$ preserving $F$ ($F^\sigma = F'$)
- Show up in some hard SAT benchmarks
- Can play crucial role in CP and MIP problems [AW13, GSVW14]

Symmetry breaking in SAT
Add constraints filtering out symmetric solutions [ASM06, DBBD16]

Symmetric learning in SAT
Allow to add all symmetric versions of learned constraint [DBB17]

Not supported by standard SAT proof logging!
Optimisation Problems

Deal with *symmetry breaking* by switching focus to *optimisation* (which the title of the talk kind of promised anyway)
Optimisation Problems

Deal with symmetry breaking by switching focus to optimisation (which the title of the talk kind of promised anyway)

Pseudo-Boolean optimisation

Minimize $f = \sum_i w_i \ell_i$ (for $w_i \in \mathbb{N}^+$) subject to constraints in $F$
Optimisation Problems

Deal with symmetry breaking by switching focus to optimisation (which the title of the talk kind of promised anyway)

Pseudo-Boolean optimisation

Minimize $f = \sum_i w_i \ell_i$ (for $w_i \in \mathbb{N}^+$) subject to constraints in $F$

Proof of optimality:

- $F$ satisfied by $\alpha$
- $F \land (\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i))$ is infeasible
Optimisation Problems

Deal with symmetry breaking by switching focus to optimisation (which the title of the talk kind of promised anyway)

Pseudo-Boolean optimisation

Minimize \( f = \sum_i w_i l_i \) (for \( w_i \in \mathbb{N}^+ \)) subject to constraints in \( F \)

Proof of optimality:

- \( F \) satisfied by \( \alpha \)
- \( F \land (\sum_i w_i l_i < \sum_i w_i \cdot \alpha(l_i)) \) is infeasible

[Note that \( \sum_i w_i l_i < \sum_i w_i \cdot \alpha(l_i) \) means \( \sum_i w_i l_i \leq -1 + \sum_i w_i \cdot \alpha(l_i) \)]
Symmetry and Dominance

Dealing with Symmetries

Optimisation Problems

Deal with symmetry breaking by switching focus to optimisation (which the title of the talk kind of promised anyway)

Pseudo-Boolean optimisation

Minimize \( f = \sum_i w_i \ell_i \) (for \( w_i \in \mathbb{N}^+ \)) subject to constraints in \( F \)

Proof of optimality:

- \( F \) satisfied by \( \alpha \)
- \( F \land (\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)) \) is infeasible

[Note that \( \sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i) \) means \( \sum_i w_i \ell_i \leq -1 + \sum_i w_i \cdot \alpha(\ell_i) \)]

Spoiler alert:
For decision problem, nothing stops us from inventing objective function (like \( \sum_{i=1}^{n} 2^{n-i} \cdot x_i \) minimized by lexicographic order)
Proof Logging for Optimisation Problems

How does proof system change?
Rules must **preserve** (at least one) **optimal solution**
Proof Logging for Optimisation Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment
Proof Logging for Optimisation Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment

2. Once solution $\alpha$ has been found, allow constraint $\sum_i w_i l_i < \sum_i w_i \cdot \alpha(l_i)$ to force search for better solutions
Proof Logging for Optimisation Problems

How does proof system change?
Rules must **preserve** (at least one) **optimal solution**

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment

2. Once solution $\alpha$ has been found, allow constraint $\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)$ to force search for better solutions

3. Redundance rule must not destroy good solutions
Proof Logging for Optimisation Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment

2. Once solution $\alpha$ has been found, allow constraint $\sum_i w_i l_i < \sum_i w_i \cdot \alpha(l_i)$ to force search for better solutions

3. Redundance rule must not destroy good solutions

Redundance-based strengthening, optimisation version

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models (F \land C)\rhd_\omega \land f\rhd_\omega \leq f$$
Redundance and Dominance Rules

Redundance-based strengthening, optimisation version

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models (F \land C)\omega \land f\omega \leq f$$

Can be more aggressive if witness $\omega$ strictly improves solution

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models (F \land C)\omega \land f\omega < f$$

Applying $\omega$ should strictly decrease $f$.

If so, don’t need to show that $C\omega$ implied!
Redundance and Dominance Rules

Redundance-based strengthening, optimisation version

Add constraint \( C \) to formula \( F \) if exists witness substitution \( \omega \) such that

\[
F \land \neg C \models (F \land C) \upharpoonright \omega \land f \upharpoonright \omega \leq f
\]

Can be more aggressive if witness \( \omega \) strictly improves solution
Redundance and Dominance Rules

**Redundance-based strengthening, optimisation version**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models (F \land C)\upharpoonright\omega \land f\upharpoonright\omega \leq f$$

Can be more aggressive if witness $\omega$ strictly improves solution

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F\upharpoonright\omega \land f\upharpoonright\omega < f$$
Redundance and Dominance Rules

Redundance-based strengthening, optimisation version
Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models (F \land C) \upharpoonright \omega \land f \upharpoonright \omega \leq f$$

Can be more aggressive if witness $\omega$ strictly improves solution

Dominance-based strengthening (simplified)
Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F \upharpoonright \omega \land f \upharpoonright \omega < f$$

- Applying $\omega$ should strictly decrease $f$
- If so, don’t need to show that $C \upharpoonright \omega$ implied!
Soundness of Dominance Rule

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F|_\omega \land f|_\omega < f$$

Why is this sound?
Soundness of Dominance Rule

Dominance-based strengthening (simplified)
Add constraint \( C \) to formula \( F \) if exists witness substitution \( \omega \) such that

\[
F \wedge \neg C \models F\upharpoonright\omega \wedge f\upharpoonright\omega < f
\]

Why is this sound?
1. Suppose \( \alpha \) satisfies \( F \) but falsifies \( C \) (i.e., satisfies \( \neg C \))
Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F\upharpoonright_\omega \land f\upharpoonright_\omega < f$$

Why is this sound?

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
Soundness of Dominance Rule

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F\lceil_\omega \land f\lceil_\omega < f$$

Why is this sound?

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies $C$, we’re done
Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F\upharpoonright\omega \land f\upharpoonright\omega < f$$

Why is this sound?

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies $C$, we’re done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $F$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F|_{\omega} \land f|_{\omega} < f$$

Why is this sound?
1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies $C$, we’re done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $F$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
Soundness of Dominance Rule

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F\upharpoonright_\omega \land f\upharpoonright_\omega < f$$

**Why is this sound?**

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies $C$, we’re done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $F$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
6. Otherwise $(((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $F$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f(((\alpha \circ \omega) \circ \omega)$
Soundness of Dominance Rule

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F\upharpoonright_\omega \land f\upharpoonright_\omega < f$$

**Why is this sound?**

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies $C$, we’re done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $F$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $F$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
7. ...
Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ such that

$$F \land \neg C \models F \upharpoonright \omega \land f \upharpoonright \omega < f$$

Why is this sound?

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $F$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies $C$, we’re done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $F$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $F$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
7. . .
8. Can’t go on forever, so finally reach $\alpha'$ satisfying $F \land C$
Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing $f = \sum_{i=1}^{n} 2^{n-i} \cdot x_i$
   (searching lexicographically smallest assignment satisfying formula)
Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing $f = \sum_{i=1}^{n} 2^{n-i} \cdot x_i$ (searching lexicographically smallest assignment satisfying formula)

2. Derive pseudo-Boolean lex-leader constraint

$$C_{\sigma} \equiv f \leq f\mid_{\sigma}$$

$$\equiv \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$
Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing \( f = \sum_{i=1}^{n} 2^{n-i} \cdot x_i \) (searching lexicographically smallest assignment satisfying formula)

2. Derive pseudo-Boolean lex-leader constraint

\[
C_\sigma \equiv f \leq f|_\sigma \\
\equiv \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0
\]

3. Derive CNF encoding of lex-leader constraints from PB constraint (in same spirit as PB-to-CNF translation in [GMNO22])

\[
y_0 \\
\overline{y}_{j-1} \lor \overline{x}_j \lor \sigma(x_j) \\
\overline{y}_j \lor \overline{y}_{j-1}
\]

\[
\overline{y}_j \lor \sigma(x_j) \lor x_j \\
y_j \lor \overline{y}_{j-1} \lor \overline{x}_j \\
y_j \lor \overline{y}_{j-1} \lor \sigma(x_j)
\]
Experimental Evaluation

- Evaluated on SAT competition benchmarks
- \texttt{BreakID} [DBBD16, Bre] used to find and break symmetries

- Proof logging overhead negligible
- Verification at most 20 times slower than solving for 95% of instances
Further Challenges

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging
- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (*work in progress*)

And more... Lots of challenging problems and interesting ideas

We're hiring! Talk to me to join the proof logging revolution!

Jakob Nordström (UCPH & LU)
Future Research Directions

Performance and reliability of pseudo-Boolean proof logging
- Trim proof while verifying (as in DRAT-trim [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (work in progress)

Proof logging for combinatorial optimization
- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) solving (work in progress [VDB22])
- Pseudo-Boolean optimization
- Mixed integer linear programming (work on SCIP in [CGS17, EG21])
Further Challenges

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging
- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (work in progress)

Proof logging for combinatorial optimization
- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) solving (work in progress [VDB22])
- Pseudo-Boolean optimization
- Mixed integer linear programming (work on SCIP in [CGS17, EG21])

And more...
- Lots of challenging problems and interesting ideas
Future Research Directions

Performance and reliability of pseudo-Boolean proof logging
- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (work in progress)

Proof logging for combinatorial optimization
- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) solving (work in progress [VDB22])
- Pseudo-Boolean optimization
- Mixed integer linear programming (work on SCIP in [CGS17, EG21])

And more...
- Lots of challenging problems and interesting ideas
- We’re hiring! Talk to me to join the proof logging revolution! 😊
Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **This work:** Efficient proof logging for symmetry and dominance breaking using cutting planes proof system with extensions
Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **This work:** Efficient proof logging for symmetry and dominance breaking using cutting planes proof system with extensions

*Thank you for your attention!*
References I


References II


References III


References IV


References V


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In *Proceedings of the 28th International Conference on Principles and</td>
<td>*Proceedings of the 28th International Conference on Principles and Practice of Constraint</td>
</tr>
<tr>
<td></td>
<td>In *Proceedings of the 25th International Conference on Theory and</td>
<td>*Proceedings of the 25th International Conference on Theory and Applications of Satisfiability</td>
</tr>
<tr>
<td></td>
<td>Presentation at KTH Royal Institute of Technology. Slides available</td>
<td>Presentation at KTH Royal Institute of Technology. Slides available at</td>
</tr>
</tbody>
</table>


