Solving with Provably Correct Results: Beyond Satisfiability, and Towards Constraint Programming

Bart Bogaerts Ciaran McCreesh
Jakob Nordström

Solvers Are Awesome!

We're here because we all know how good solvers for CP, SAT, MIP, etc have become.

The Controversial Slide

Last year's MiniZinc challenge: for 1.28% of instances, wrong solutions were claimed.

- False claims of unsatisfiability.
- False claims of optimality.
- Infeasible solutions produced.

This problem is worth taking seriously.

- Not limited to a single solver, problem, or constraint.
- Not even consistent—same solver on same hardware and same instance can give different results on different runs.

Obviously, *your* solver doesn't have this problem, but how do you convince others of this?

Testing?

Various domain-specific testing methods [BLB10, AGJ$^+$18, GSD19].

Definitely better than nothing, but is it enough?

Testing?

Various domain-specific testing methods [BLB10, AGJ$^+$18, GSD19].

Definitely better than nothing, but is it enough?

- Do you still think it's enough if a solver is making a decision that will affect your life or livelihood?

Formal Methods?

Prove that solver implementation adheres to formal specification.

Current techniques cannot scale to this level of complexity.

- Even an inefficient implementation of all-different is pushing the limits [Dub20].

This One Simple Trick Fixes Everything!

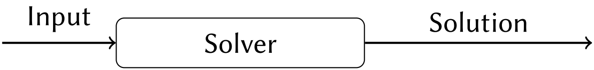
For SAT solvers: proof logging.

- A particular kind of certifying algorithm [ABM+11, MMNS11].

Solvers output a proof in a standard format, which can be verified independently.

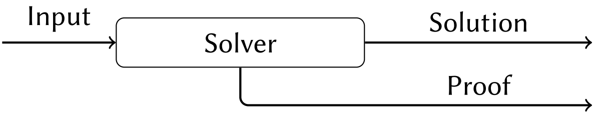
- A variety of formats, including
  DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17],
  LRAT [CHH+17], …

Proof Logging Workflow



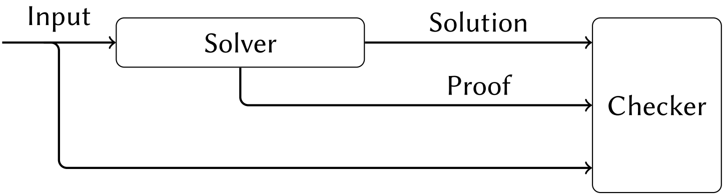
1. Run solver on problem input.

Proof Logging Workflow



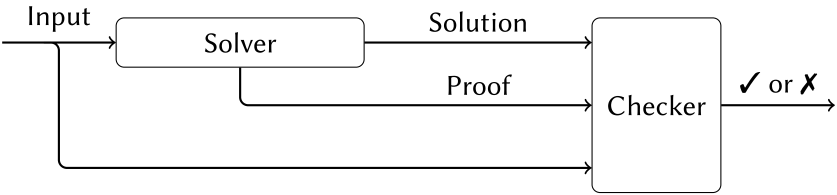
1. Run solver on problem input.
2. Get as output not only solution but also proof.

Proof Logging Workflow



1. Run solver on problem input.
2. Get as output not only solution but also proof.
3. Feed input + solution + proof to proof checker.

Proof Logging Workflow



1. Run solver on problem input.
2. Get as output not only solution but also proof.
3. Feed input + solution + proof to proof checker.
4. Verify that proof checker says solution is correct.

Requirements

Proofs produced by certifying solver should:

- Be powerful enough to allow proof logging with minimal overhead.
- Be based on very simple rules.
- Not require knowledge of inner workings of solver.
- Allow verification by stand-alone proof checker.

Much easier to trust a small, simple checker than a full solver.

- Should even be simple enough to be formally verified.

Does not prove solver correct, but proves solution correct.

The Sales Pitch For Proof Logging

1. Certifies correctness of solutions.
2. Detects errors even if due to compiler bugs, hardware failures, or cosmic rays.
3. Provides debugging support during development [EG21, GMM+20, KM21].
4. Facilitates performance analysis.
5. Helps identify potential for further improvements.
6. Enables auditability.
7. Serves as stepping stone towards explainability.

The Rest of This Tutorial

VeriPB (`https://gitlab.com/MIAOresearch/VeriPB`)

Versatile proof logging system that can handle

- Subgraph algorithms (we'll show lots of examples).
- Constraint programming (we'll give an overview).
- Symmetries and dominance (time and interest dependent).

in a unified way.

The Rest of This Tutorial

VeriPB (`https://gitlab.com/MIAOresearch/VeriPB`)

Versatile proof logging system that can handle

- Subgraph algorithms (we'll show lots of examples).
- Constraint programming (we'll give an overview).
- Symmetries and dominance (time and interest dependent).

in a unified way.

But first we need to tell you about:

- Proof logging for SAT.
- Pseudo-Boolean reasoning and cutting planes.

Proofs for SAT

For satisfiable instances: just specify a satisfying assignment.

For unsatisfiability: a proof is a sequence of clauses (CNF constraints).

- Each clause follows "obviously" from everything we know so far.
- Final clause is empty, meaning contradiction (written $\perp$).

Forward Checking (DPLL)

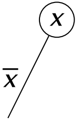
We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Forward Checking (DPLL)

We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

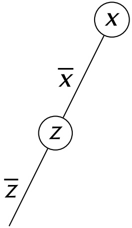
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Forward Checking (DPLL)

We could write a “proof” of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

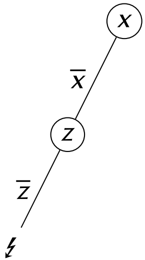


Forward Checking (DPLL)

We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1 $x \vee z$

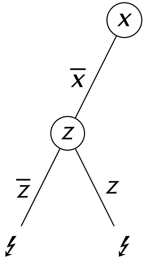


Forward Checking (DPLL)

We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee z$
2. $x \vee \overline{z}$

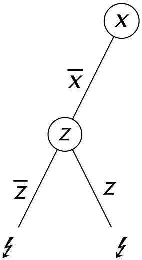


Forward Checking (DPLL)

We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee z$
2. $x \vee \overline{z}$
3. x

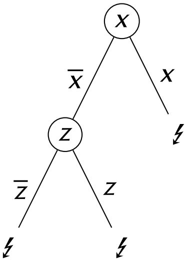


Forward Checking (DPLL)

We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee z$
2. $x \vee \overline{z}$
3. x
4. $\overline{x}$



Forward Checking (DPLL)

We could write a "proof" of unsatisfiability by writing a step whenever a forward-checker backtracks asserting the negation of the guesses we made. For example,

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

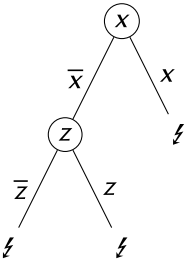
1 $x \vee z$

2 $x \vee \overline{z}$

3 x

4 $\overline{x}$

5 $\perp$



Reverse Unit Propagation (RUP)

To make this a proof, need each backtrack clause to be easily verifiable.

Reverse unit propagation (RUP) clause [GN03, Van08]

C is a reverse unit propagation (RUP) clause with respect to F if

- assigning C to false,
- then unit propagating on F until saturation
- leads to contradiction

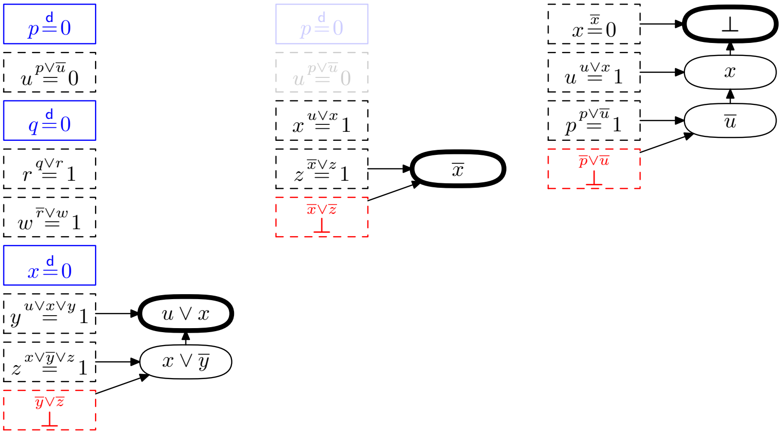
If so, F clearly implies C, and condition easy to verify efficiently

Fact

Backtracks from DPLL solver generate a RUP proof.

What About CDCL?

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So shorter proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

Writing Proofs in the DRAT Format

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Writing Proofs in the DRAT Format

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

Writing Proofs in the DRAT Format

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee z$

$x \vee \overline{z}$

x

$\overline{x}$

$\perp$

Writing Proofs in the DRAT Format

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee z$

$x \vee \overline{z}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

```
6 8 0
6 -8 0
6 0
-6 0
0
```

Writing Proofs in the DRAT Format

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee z$

$x \vee \overline{z}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

```
6 8 0
6 -8 0
6 0
-6 0
0
```

CDCL Proof as RUP

$u \vee x$

$\overline{x}$

$\perp$

Writing Proofs in the DRAT Format

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee z$

$x \vee \overline{z}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

```
6 8 0
6 -8 0
6 0
-6 0
0
```

CDCL Proof as RUP

$u \vee x$

$\overline{x}$

$\perp$

CDCL Proof in DRAT

```
4 6 0
-6 0
0
```

Extension Variables, Part 1

Fact

RUP proofs are short-hand for so-called Resolution proofs.

RUP and Resolution aren't enough for preprocessing, inprocessing, or some kinds of reasoning.

Suppose we want new, fresh variable a encoding

$$a \leftrightarrow (x \land y)$$

Extended Resolution: introduce clauses

$$a \lor \overline{x} \lor \overline{y} \qquad \overline{a} \lor x \qquad \overline{a} \lor y$$

Should be fine, so long as a doesn't appear anywhere previously.

Deleting Clauses

In practice, important to erase lines to save memory and time.

Very easy to deal with from the point of view of proof logging.

So ignored in this tutorial for simplicity and clarity.

Why Aren't We Done?

Practical limitations of SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently.
- Clausal proofs can't easily reflect what other algorithms do.

Surprising claim: a slight change to 0-1 integer linear inequalities does the job!

- Can justify graph reasoning without knowing what a graph is.
- Can justify constraint programming inference without knowing what an integer variable is.
- This even helps justify advanced SAT techniques (cardinality reasoning, Gaussian elimination, symmetry elimination).

Pseudo-Boolean Constraints

0-1 integer linear inequalities or pseudo-Boolean constraints:

$$\sum_i a_i \ell_i \geq A$$

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

Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

2. Cardinality constraints

$$x_1 + x_2 + x_3 + x_4 \geq 2$$

3. General pseudo-Boolean constraints

$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

RUP Revisited

Can define (reverse) unit propagation in a pseudo-Boolean setting.

Confusing terminology: in CP, we'd call it (reverse) integer bounds consistency.

- Does the same thing if we're working with clauses.
- More interesting for general pseudo-Boolean constraints.

SAT people beware: constraints can propagate multiple variables and multiple times.

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i) \ell_i \geq A + B}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i) \ell_i \geq A + B}$$

Multiplication
for any $c \in \mathbb{N}^+$

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

Multiplication
for any $c \in \mathbb{N}^+$

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

Division
for any $c \in \mathbb{N}^+$

$$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i) \ell_i \geq A + B}$$

Multiplication
for any $c \in \mathbb{N}^+$

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

Division
for any $c \in \mathbb{N}^+$

$$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

Literal axioms

$$\frac{}{\ell_i \geq 0}$$

Extension Variables, Part 2

Suppose we want new, fresh variable a encoding

$$a \leftrightarrow (x \land y)$$

This time, introduce constraints

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

Again, needs support from the proof system.

Proof Logs for Extended Cutting Planes

For satisfiable instances: just specify a satisfying assignment.

For unsatisfiability: a proof is a sequence of pseudo-Boolean constraints.

- Each constraint follows "obviously" from everything we know so far.
- Either implicitly, by RUP...
- Or an explicit cutting planes derivation...
- Or an extension variable reifying a new constraint*
- Final clause is $0 \geq 1$.

(*) Not actually implemented this way: details later.

Satisfiable, Enumeration, and Optimisation Problems

When a solution is found, can log it.

- Introduces a new axiom constraint saying "not this solution".
- So the proof semantics are "unsatisfiable, except for all the solutions I told you about".

For optimisation,

- Define an objective $f = \sum_i w_i \ell_i$, $w \in \mathbb{Z}$, to minimise in the pseudo-Boolean model.
- Log a solution α, get a solution-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \alpha(\ell_i)$.

The VeriPB Format and Tool

`https://gitlab.com/MIAOresearch/VeriPB`

MIT Licence.

Various features to help development:

- Extended variable name syntax allowing human-readable names.
- Proof tracing.
- "Trust me" assertions.

Full details: Stephan Gocht's PhD thesis [Goc22].

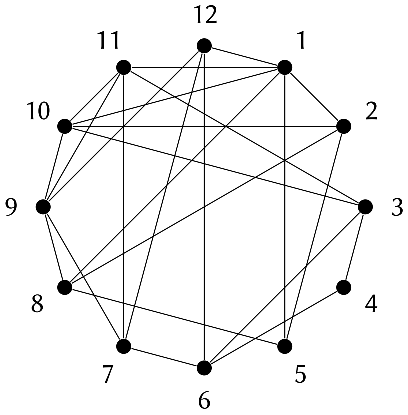
Progress So Far

We've seen proof logging, and how it works for SAT.

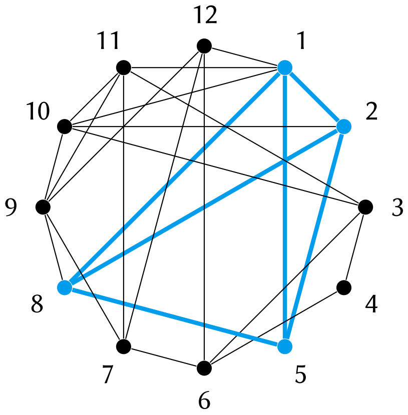
We've learned about cutting planes and VeriPB.

Coming next, some worked examples from dedicated graph solvers.

The Maximum Clique Problem



The Maximum Clique Problem



Maximum Clique Solvers

There are a lot of dedicated solvers for clique problems.

But there are issues:

- "State of the art" solvers have been buggy.
- Often undetected: error rate of around 0.1% [MPP19].

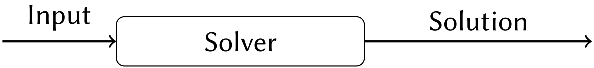
Often used inside other solvers.

- An off-by-one result can cause much larger errors.

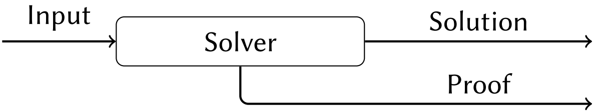
Making a Proof-Logging Clique Solver

1. Output a pseudo-Boolean encoding of the problem.
   - Clique problems have several standard file formats.
2. Make the solver log its search tree.
   - Output a small header.
   - Output something on every backtrack.
   - Output something every time a solution is found.
   - Output a small footer.
3. Figure out how to log the bound function.

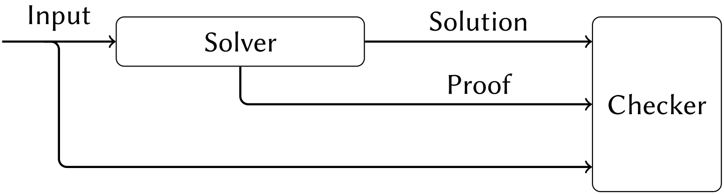
A Slightly Different Workflow



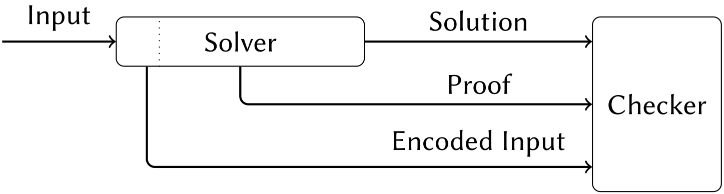
A Slightly Different Workflow



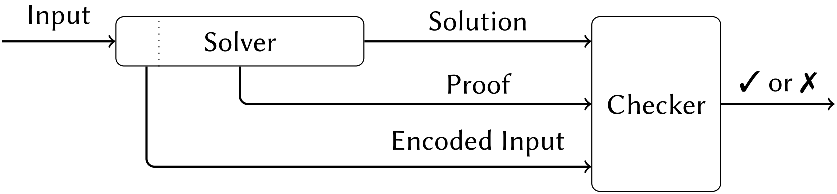
A Slightly Different Workflow



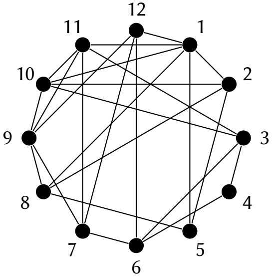
A Slightly Different Workflow



A Slightly Different Workflow



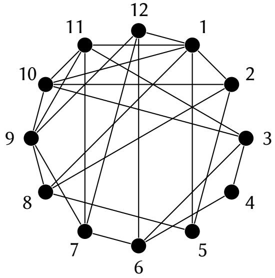
A Pseudo-Boolean Encoding for Clique (in OPB Format)




```
* #variable= 12 #constraint= 41
min: -1 x1 -1 x2 -1 x3 -1 x4 . . . and so on. . .  -1 x11 -1 x12 ;
1 ~x3 1 ~x1 >= 1 ;
1 ~x3 1 ~x2 >= 1 ;
1 ~x4 1 ~x1 >= 1 ;
* . . . and a further 38 similar lines for the remaining non-edges
```

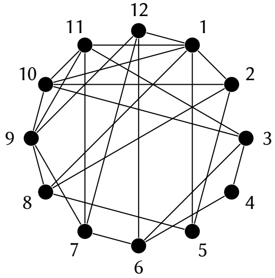
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



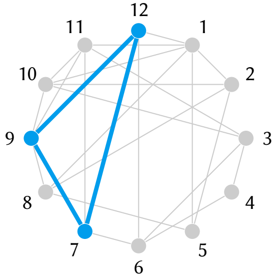
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



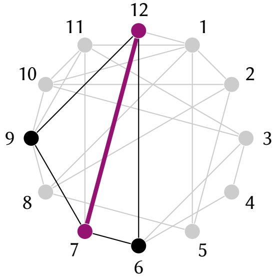
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



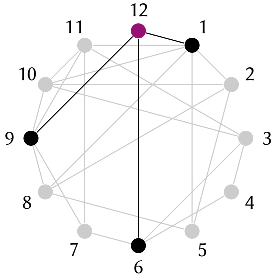
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



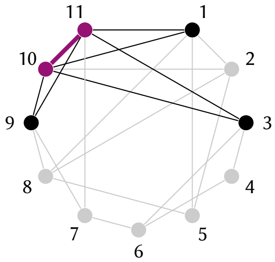
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



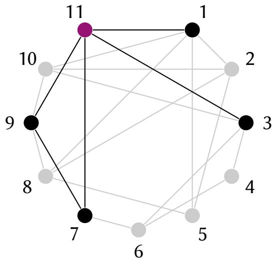
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



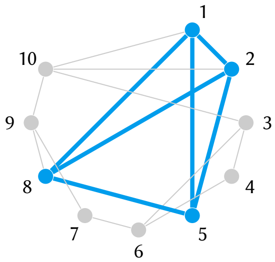
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



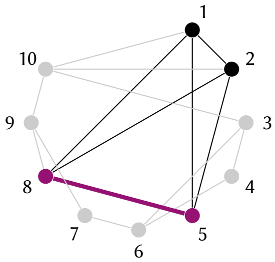
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



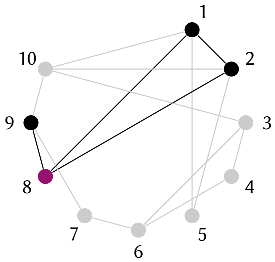
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



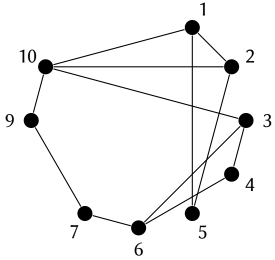
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



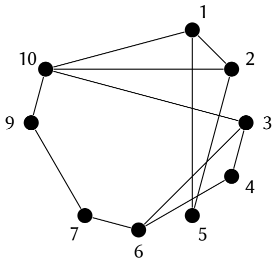
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```

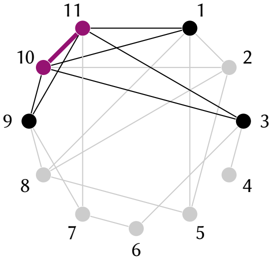


Verifying This Proof (Or Not...)

```
$ veripb clique.opb clique-attempt-one.veripb
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```

Verifying This Proof (Or Not…)

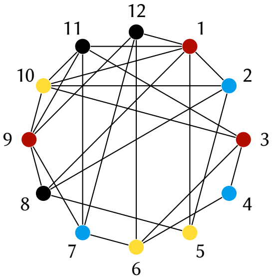
```
$ veripb clique.opb clique-attempt-one.veripb
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```



Verifying This Proof (Or Not...)

```
$ veripb --trace clique.opb clique-attempt-one.veripb
  ...
  ConstraintId 040: 1 ~x10 1 ~x12 >= 1
  ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: o x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
  ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x1
line 004: u 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: u 1 ~x12 >= 1 ;
  ConstraintId 044: 1 ~x12 >= 1
line 006: u 1 ~x11 1 ~x10 >= 1 ;
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```

Bound Functions



Given a *k*-colouring of a subgraph, that subgraph cannot have a clique of more than *k* vertices.

- Each colour class describes an at-most-one constraint.

This does *not* follow by reverse unit propagation.

Recovering At-Most-One Constraints

Can't list every colour class we *might* use in the pseudo-Boolean input.

We can use cutting planes to recover colour classes lazily!

Recovering At-Most-One Constraints

Can't list every colour class we *might* use in the pseudo-Boolean input.

We can use cutting planes to recover colour classes lazily!

$$
\begin{aligned}
(\overline{x}_1 + \overline{x}_6 \geq 1) \times 2 \quad &= \quad 2\overline{x}_1 + 2\overline{x}_6 \geq 2 \\
+ (\overline{x}_1 + \overline{x}_9 \geq 1) \quad &= \quad 3\overline{x}_1 + 2\overline{x}_6 + \overline{x}_9 \geq 6 \\
+ (\overline{x}_6 + \overline{x}_9 \geq 1) \quad &= \quad 3\overline{x}_1 + 3\overline{x}_6 + 2\overline{x}_9 \geq 4 \\
/\, 3 \quad &= \quad \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
&\text{i.e. } x_1 + x_6 + x_9 \leq 1
\end{aligned}
$$

Recovering At-Most-One Constraints

Can't list every colour class we *might* use in the pseudo-Boolean input.

We can use cutting planes to recover colour classes lazily!

$$
\begin{aligned}
(\overline{x}_1 + \overline{x}_6 \geq 1) \times 2 \quad &= \quad 2\overline{x}_1 + 2\overline{x}_6 \geq 2 \\
+ (\overline{x}_1 + \overline{x}_9 \geq 1) \quad &= \quad 3\overline{x}_1 + 2\overline{x}_6 + \overline{x}_9 \geq 6 \\
+ (\overline{x}_6 + \overline{x}_9 \geq 1) \quad &= \quad 3\overline{x}_1 + 3\overline{x}_6 + 2\overline{x}_9 \geq 4 \\
/\, 3 \quad &= \quad \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
&\text{i.e. } x_1 + x_6 + x_9 \leq 1
\end{aligned}
$$

This generalises for arbitrarily large colour classes.

- Each non-edge is used exactly once, $v(v-1)$ additions.
- $v-2$ multiplications and divisions.

Solvers don't need to “understand” cutting planes to write this out.

What This Looks Like (More or Less)

```
pseudo-Boolean proof version 1.2
f 41 0
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
* at most one [ x1 x6 x9 ]
p nonadj1_6 2 * nonadj1_9 + nonadj6_9 + 3 d                    ⇝ tmp1
p obj1 tmp1 +
u 1 ~x11 1 ~x10 >= 1 ;                                          ⇝ b3
* at-most-one [ x1 x3 x9 ]
p nonadj1_3 2 * nonadj1_9 + nonadj3_9 + 3 d                    ⇝ tmp2
p obj1 tmp2 +
u 1 ~x11 >= 1 ;                                                 ⇝ b4
o x1 x2 x5 x8                                                   ⇝ obj2
u 1 ~x8 1 ~x5 >= 1 ;                                            ⇝ b5
p obj2 nonadj1_9 +
u 1 ~x8 >= 1 ;                                                  ⇝ b6
* at-most-one [ x1 x3 x7 ] [ x2 x4 x9 ] [ x5 x6 x10 ]
p nonadj1_3 2 * nonadj1_7 + nonadj3_7 + 3 d                    ⇝ tmp3
p obj2 tmp3 +
p nonadj2_4 2 * nonadj2_9 + nonadj4_9 + 3 d                    ⇝ tmp4
p obj2 tmp3 + tmp4 +
p nonadj5_6 2 * nonadj5_10 + nonadj6_10 + 3 d                  ⇝ tmp5
p obj2 tmp3 + tmp4 + tmp5 +
u >= 1 ;                                                        ⇝ done
c done
```

Verifying This Proof (For Real, This Time)

```
$ veripb --trace clique.opb clique-attempt-two.veripb
  ...
line 005: u 1 ~x12 >= 1 ;
  ConstraintId 044: 1 ~x12 >= 1
line 006: p 7 2 * 19 + 24 + 3 d
  ConstraintId 045: 1 ~x1 1 ~x6 1 ~x9 >= 2
line 007: p -1 42 +
  ConstraintId 046: 1 x2 1 x3 1 x4 1 x5 1 x7 1 x8 1 x10 1 x11 1 x12 >= 3
  ...
line 020: p 51 -1 + -3 + -5 +
  ConstraintId 059: 1 x8 1 x11 1 x12 >= 2
line 021: u >= 1 ;
  ConstraintId 060: >= 1
line 022: c -1
=== end trace ===

Verification succeeded.
```

Different Clique Algorithms

Different search orders?

✓ Irrelevant for proof logging.

Using local search to initialise?

✓ Just log the incumbent.

Different bound functions?

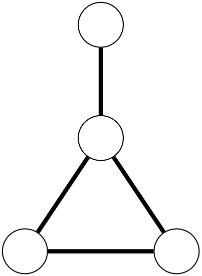
- Is cutting planes strong enough to justify every useful bound function ever invented?
- So far, seems like it…

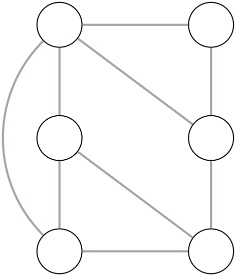
Weighted cliques?

✓ Multiply a colour class by its largest weight.

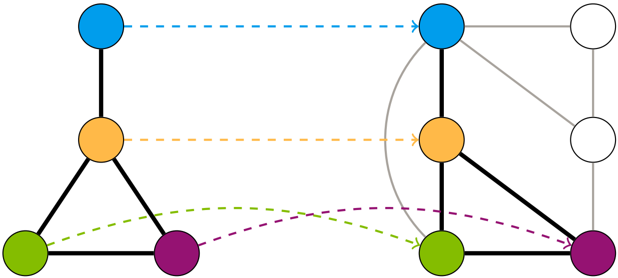
✓ Also works for vertices "split between colour classes".

Subgraph Isomorphism





Subgraph Isomorphism



Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex must be mapped to exactly one target vertex:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex must be mapped to exactly one target vertex:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

Injectivity, each target vertex may be used at most once:

$$\sum_{p \in \mathrm{V}(P)} -x_{p,t} \geq -1 \qquad\qquad t \in \mathrm{V}(T)$$

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex must be mapped to exactly one target vertex:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

Injectivity, each target vertex may be used at most once:

$$\sum_{p \in \mathrm{V}(P)} -x_{p,t} \geq -1 \qquad\qquad t \in \mathrm{V}(T)$$

Adjacency constraints, if a vertex p is mapped to a vertex t, then every vertex in the neighbourhood of p must be mapped to a vertex in the neighbourhood of t:

$$\overline{x}_{p,t} + \sum_{u \in \mathrm{N}(t)} x_{q,u} \geq 1 \qquad p \in \mathrm{V}(P),\ q \in \mathrm{N}(p),\ t \in \mathrm{V}(T)$$

Degree Reasoning in Cutting Planes

A pattern vertex p of degree $\deg(p)$ can never be mapped to a target vertex t of degree $\deg(p) - 1$ or lower in any subgraph isomorphism.

Suppose $\mathrm{N}(p) = \{q, r, s\}$ and $\mathrm{N}(t) = \{u, v\}$.

We wish to derive $\overline{x}_{p,t} \geq 1$.

Degree Reasoning in Cutting Planes

We have the three adjacency constraints,

$$\overline{x}_{p,t} + x_{q,u} + x_{q,v} \geq 1$$
$$\overline{x}_{p,t} + x_{r,u} + x_{r,v} \geq 1$$
$$\overline{x}_{p,t} + x_{s,u} + x_{s,v} \geq 1$$

Their sum is

$$3\overline{x}_{p,t} + x_{q,u} + x_{q,v} + x_{r,u} + x_{r,v} + x_{s,u} + x_{s,v} \geq 3$$

Degree Reasoning in Cutting Planes

Remember, $\mathsf{N}(p) = \{q, r, s\}$ and $\mathsf{N}(t) = \{u, v\}$.

Continuing with the sum

$$3\overline{x}_{p,t} + x_{q,u} + x_{q,v} + x_{r,u} + x_{r,v} + x_{s,u} + x_{s,v} \geq 3$$

Due to injectivity,

$$\sum_{\alpha \in \mathsf{V}(P)} -x_{\alpha,u} \geq -1 \quad \text{and} \quad \sum_{\alpha \in \mathsf{V}(P)} -x_{\alpha,v} \geq -1$$

Add all these together, getting

$$3\overline{x}_{p,t} + \sum_{\alpha \in \mathsf{V}(P) \setminus \{q,r,s\}} -x_{\alpha,u} + \sum_{\alpha \in \mathsf{V}(P) \setminus \{q,r,s\}} -x_{\alpha,v} \geq 1$$

Degree Reasoning in Cutting Planes

Remember, $\mathsf{N}(p) = \{q, r, s\}$ and $\mathsf{N}(t) = \{u, v\}$.

Continuing with the sum of sums

$$3\overline{x}_{p,t} + \sum_{\alpha \in \mathsf{V}(P) \setminus \{q,r,s\}} -x_{\alpha,u} + \sum_{\alpha \in \mathsf{V}(P) \setminus \{q,r,s\}} -x_{\alpha,v} \geq 1$$

Add the literal axioms

$$\sum_{\alpha \in \mathsf{V}(P) \setminus \{q,r,s\}} x_{\alpha,u} + \sum_{\alpha \in \mathsf{V}(P) \setminus \{q,r,s\}} x_{\alpha,v} \geq 0$$

to get

$$3\overline{x}_{p,t} \geq 1$$

Divide by 3 to get the desired

$$\overline{x}_{p,t} \geq 1$$

Degree Reasoning in VeriPB

```
p 18 19 + 20 +       * sum adj constraints
  12 + 13 +          * sum inj constraints
  xp_u + xp_v +      * cancel stray xp_*
  xo_u + xo_v +      * cancel stray xo_*
  3 d 0              * divide, and we're done
e -1 1 ~xp_t >= 1 ; * check what we just did
```

Degree Reasoning in VeriPB

```
p 18 19 + 20 +       * sum adj constraints
  12 + 13 + 0        * sum inj constraints
j -1 1 ~xp_t >= 1 ; * and simplify the above
```

All-Different Failures

$$
\begin{array}{rl}
v \in & \{\ 1 \quad\quad\quad\quad 4\ \} \\
w \in & \{\ 1 \quad 2 \quad 3 \quad\ \ \} \\
x \in & \{\ \quad\ \ 2 \quad 3 \quad\ \ \} \\
y \in & \{\ 1 \quad\quad\ \ 3 \quad\ \ \} \\
z \in & \{\ 1 \quad\quad\ \ 3 \quad\ \ \}
\end{array}
$$

All-Different Failures

$$
\begin{array}{rl}
v \in & \{\,1 \quad\quad\quad 4\,\} \\
w \in & \{\,1 \quad 2 \quad 3 \quad\,\} \\
x \in & \{\quad\;\; 2 \quad 3 \quad\,\} \\
y \in & \{\,1 \quad\quad 3 \quad\,\} \\
z \in & \{\,1 \quad\quad 3 \quad\,\}
\end{array}
$$

All-Different Failures

$$
\begin{array}{rl}
v \in \{\, 1 & \ 4 \,\} \\
w \in \{\, 1 & 2\ \ 3 \,\} \qquad x_{w,1} + x_{w,2} + x_{w,3} \geq 1 \\
x \in \{\, & 2\ \ 3 \,\} \\
y \in \{\, 1 & \ \ 3 \,\} \\
z \in \{\, 1 & \ \ 3 \,\}
\end{array}
$$

All-Different Failures

$$\begin{array}{rlllll}
v \in \{1 & & & 4\} & & \\
w \in \{1 & 2 & 3 & \} & x_{w,1} + x_{w,2} + x_{w,3} & \geq 1 \\
x \in \{ & 2 & 3 & \} & x_{x,2} + x_{x,3} & \geq 1 \\
y \in \{1 & & 3 & \} & x_{y,1} + x_{y,3} & \geq 1 \\
z \in \{1 & & 3 & \} & x_{z,1} + x_{z,3} & \geq 1
\end{array}$$

All-Different Failures

$$
\begin{array}{rcl}
v &\in& \{\,1 \qquad\quad 4\,\} \\
w &\in& \{\,1 \;\; 2 \;\; 3 \;\;\,\} \quad x_{w,1} + x_{w,2} + x_{w,3} \geq 1 \\
x &\in& \{\;\;\; 2 \;\; 3 \;\;\,\} \quad x_{x,2} + x_{x,3} \geq 1 \\
y &\in& \{\,1 \;\;\;\;\; 3 \;\;\,\} \quad x_{y,1} + x_{y,3} \geq 1 \\
z &\in& \{\,1 \;\;\;\;\; 3 \;\;\,\} \quad x_{z,1} + x_{z,3} \geq 1
\end{array}
$$

$$\rightarrow \quad -x_{v,1} + -x_{w,1} + -x_{y,1} + -x_{z,1} \geq -1$$

$$\rightarrow \quad -x_{w,2} + -x_{x,2} \geq -1$$

$$\rightarrow \quad -x_{w,3} + -x_{x,3} + -x_{y,3} + -x_{z,3} \geq -1$$

All-Different Failures

$$
\begin{array}{rlllllllll}
v \in \{1 & & & 4\} & & & & & & \\
w \in \{1 & 2 & 3 & \} & x_{w,1} + & x_{w,2} + & x_{w,3} & & & \geq 1 \\
x \in \{ & 2 & 3 & \} & & x_{x,2} + & x_{x,3} & & & \geq 1 \\
y \in \{1 & & 3 & \} & x_{y,1} & + & x_{y,3} & & & \geq 1 \\
z \in \{1 & & 3 & \} & x_{z,1} & + & x_{z,3} & & & \geq 1
\end{array}
$$

$$
\begin{array}{ll}
\rightarrow & -x_{v,1} + -x_{w,1} + \qquad -x_{y,1} + -x_{z,1} \geq -1 \\
\quad\rightarrow & \qquad -x_{w,2} + -x_{x,2} \geq -1 \\
\qquad\rightarrow & \qquad -x_{w,3} + -x_{x,3} + -x_{y,3} + -x_{z,3} \geq -1
\end{array}
$$

$$-x_{v,1} \geq 1$$

All-Different Failures

$$
\begin{array}{rcl}
v & \in & \{\, 1 \qquad\quad 4 \,\} \\
w & \in & \{\, 1 \;\; 2 \;\; 3 \quad \} \\
x & \in & \{\, \quad 2 \;\; 3 \quad \} \\
y & \in & \{\, 1 \quad\;\; 3 \quad \} \\
z & \in & \{\, 1 \quad\;\; 3 \quad \}
\end{array}
\qquad
\begin{array}{rcl}
x_{w,1} + x_{w,2} + x_{w,3} & \geq & 1 \\
x_{x,2} + x_{x,3} & \geq & 1 \\
x_{y,1} + x_{y,3} & \geq & 1 \\
x_{z,1} + x_{z,3} & \geq & 1
\end{array}
$$

$$
\begin{array}{lrcl}
\rightarrow & -x_{v,1} + -x_{w,1} + -x_{y,1} + -x_{z,1} & \geq & -1 \\
\quad\rightarrow & -x_{w,2} + -x_{x,2} & \geq & -1 \\
\qquad\rightarrow & -x_{w,3} + -x_{x,3} + -x_{y,3} + -x_{z,3} & \geq & -1
\end{array}
$$

$$
\begin{array}{rcl}
-x_{v,1} & \geq & 1 \\
x_{v,1} & \geq & 0
\end{array}
$$

All-Different Failures

$$
\begin{array}{lllllll}
v \in \{\, 1 \quad\quad\quad 4 \,\} & & & & & & \\
w \in \{\, 1 \;\; 2 \;\; 3 \;\; \} & x_{w,1} + & x_{w,2} + & x_{w,3} & & & \geq 1 \\
x \in \{\, \;\; 2 \;\; 3 \;\; \} & & x_{x,2} + & x_{x,3} & & & \geq 1 \\
y \in \{\, 1 \;\; \;\; 3 \;\; \} & x_{y,1} & + & x_{y,3} & & & \geq 1 \\
z \in \{\, 1 \;\; \;\; 3 \;\; \} & x_{z,1} & + & x_{z,3} & & & \geq 1
\end{array}
$$

$$
\begin{array}{lllllll}
\rightarrow & -x_{v,1} + & -x_{w,1} + & & -x_{y,1} + & -x_{z,1} & \geq -1 \\
\quad \rightarrow & & -x_{w,2} + & -x_{x,2} & & & \geq -1 \\
\quad\quad \rightarrow & & -x_{w,3} + & -x_{x,3} + & -x_{y,3} + & -x_{z,3} & \geq -1
\end{array}
$$

$$
\begin{array}{ll}
-x_{v,1} & \geq 1 \\
x_{v,1} & \geq 0
\end{array}
$$

$$
0 \geq 1
$$

Other Forms of Reasoning

We can also do:

- All-different filtering.
- Distance filtering.
- Neighbourhood degree sequences.
- Path filtering.
- Supplemental graphs.

Proof steps are "efficient" using cutting planes.

- The length of the proof steps are no worse than the time complexity of the reasoning algorithms.
- Most proof steps require only trivial additional computations.
  - Worst case: all-different requires finding one additional alternating path.

Limitations

Why trust the encoding?

- Here we can use formal verification! Work in progress…

Proof logging can introduce large slowdowns

- Writing to disk is much slower than bit-parallel algorithms.

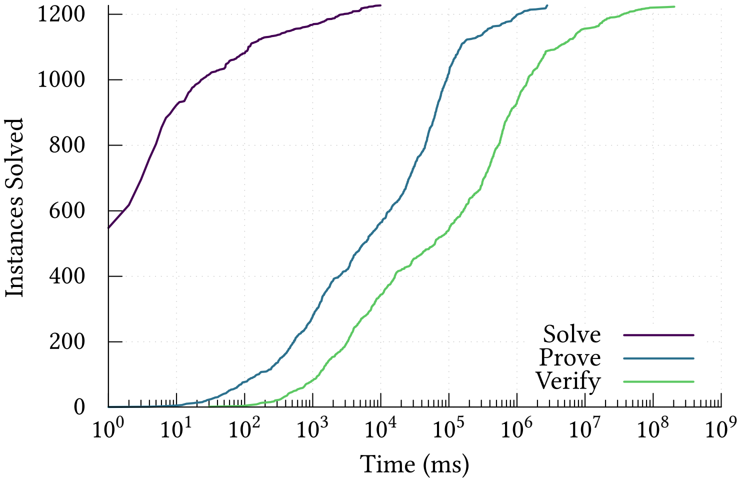
Verification can be even slower

- Unit propagation is much slower than bit-parallel algorithms.

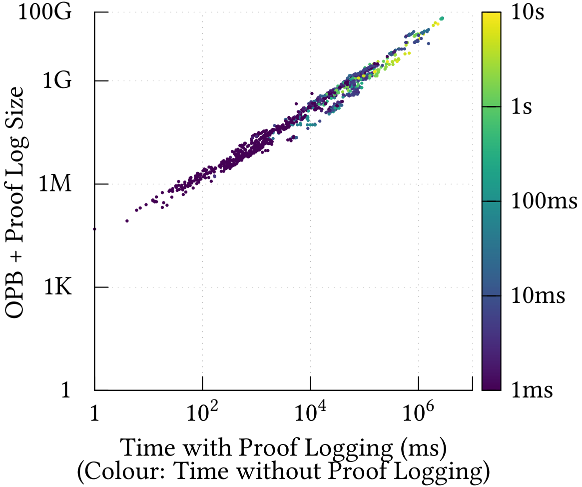
Works up to moderately-sized hard instances

- Even an $O(n^3)$ encoding is painful.
- Particularly bad when the psuedo-Boolean encoding talks about "non-edges" but large sparse graphs are "easy".

Performance for Subgraph Isomorphism



Performance for Subgraph Isomorphism



Code

`https://github.com/ciaranm/glasgow-subgraph-solver`

MIT Licence.

What About CP?

Non-Boolean variables?

Constraints?

- Encoding constraints as Pseudo-Boolean constraints?
- Justifying inference?

Reformulation?

Work in progress: more on Friday.

Variables, Take One

$$A \in \{1, 2, 3, 4, 5\}$$

becomes

$$a_{=1} + a_{=2} + a_{=}3 + a_{=4} + a_{=5} = 1$$

But this is unusable for large domains.

Variables, Take Two

$$A \in \{-3 \ldots 9\}$$

becomes

$$-32a_{\mathsf{neg}} + 1a_{\mathsf{b0}} + 2a_{\mathsf{b1}} + 4a_{\mathsf{b2}} + 8a_{\mathsf{b3}} + 16a_{\mathsf{b4}} \geq -3$$

and

$$32a_{\mathsf{neg}} + -1a_{\mathsf{b0}} + -2a_{\mathsf{b1}} + -4a_{\mathsf{b2}} + -8a_{\mathsf{b3}} + -16a_{\mathsf{b4}} \geq -9$$

Weakly propagating, but that doesn't matter!

Really annoying for proofs, though…

Lazily Introducing Direct Variables

Whenever we propagate a value or bounds, introduce $x_{\geq i}$ and $x_{=i}$ as extension variables.

This works because for large domains, most values are never used.

Encoding Propagators

We already know how to do it for any propagator that has a sane encoding using some combination of

- CNF,
- Integer linear inequalities,
- Table constraints,
- Auxiliary variables.

Simplicity is important, propagation strength isn't.

Justifying Search

Nothing new to say.

Restarts are easy.

Justifying Inference

If it follows from unit propagation, nothing needed.

Some propagators and encodings need RUP steps for inferences.

A few need explicit cutting planes justifications.

What about inference during search?

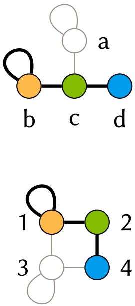
- Roughly speaking, you can pretend guessed assignments aren't there.

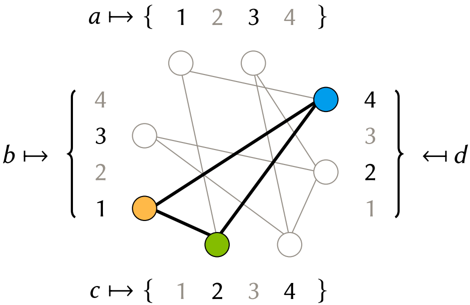
Reformulation

Auto-tabulation is possible.

- Heavy use of extension variables.

Can re-encode maximum common subgraph as a clique problem, without changing the pseudo-Boolean model.





High Level Modelling Languages?

There are formally verified compilers, maybe these can be inspirational?

Edge-case semantics of constraints aren't obvious!

Experience so far: at least two bugs in my code that turns XCSP into a low level model.

Code

`https://github.com/ciaranm/glasgow-constraint-solver`

MIT Licence.

All-different, integer linear inequality (including for variables with very large domains), table, minimum / maximum of an array, element, absolute value.

More on Friday at 12:00 in Taub 7.

What's Left?

Symmetries!

But first, some more about extension variables…

The Truth About Extension Variables

Recall: we want new, fresh variable a encoding

$$a \leftrightarrow (x \wedge y)$$

Introduce clauses

$$a \vee \overline{x} \vee \overline{y} \qquad \overline{a} \vee x \qquad \overline{a} \vee y$$

Or constraints

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

Resolution and cutting planes proof system inherently cannot certify such derivations: they are not implied!

Redundance-Based Strengthening

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

Adding redundant clauses should be OK

Redundance-Based Strengthening

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

Adding redundant clauses should be OK

Redundance-based strengthening [BT19, GN21] (extending DRAT)

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

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$$

Redundance-Based Strengthening

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

Adding redundant clauses should be OK

Redundance-based strengthening [BT19, GN21] (extending DRAT)

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

$$F \land \neg C \models (F \land C)\restriction_{\omega}$$

Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \land C$

Redundance-Based Strengthening

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

Adding redundant clauses should be OK

Redundance-based strengthening [BT19, GN21] (extending DRAT)

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

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$$

Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \wedge C$

Implication should be efficiently verifiable (which is the case, e.g., if all clauses in $(F \wedge C)\restriction_{\omega}$ are RUP)

Deriving $a \leftrightarrow (x \land y)$ Using the Redundance Rule

Want to derive

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

using condition $F \land \neg C \models (F \land C)\restriction_\omega$

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

using condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(2\overline{a} + x + y \geq 2) \models (F \wedge (2\overline{a} + x + y \geq 2))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new constraint satisfied

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

using condition $F \wedge \neg C \models (F \wedge C)\restriction_\omega$

1. $F \wedge \neg(2\overline{a} + x + y \geq 2) \models (F \wedge (2\overline{a} + x + y \geq 2))\restriction_\omega$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new constraint satisfied
2. $F \wedge (2\overline{a} + x + y \geq 2) \wedge \neg(a + \overline{x} + \overline{y} \geq 1) \models$
   $(F \wedge (2\overline{a} + x + y \geq 2) \wedge (a + \overline{x} + \overline{y} \geq 1))\restriction_\omega$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(a + \overline{x} + \overline{y} \geq 1)$ forces $x \mapsto 1$ and $y \mapsto 1$, hence $2\overline{a} + x + y \geq 2$ remains satisfied after forcing a to be true

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega} \wedge f\restriction_{\omega} \leq f$$

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega \wedge f\restriction_\omega \leq f$$

Can be more aggressive if witness ω strictly improves solution.

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega} \wedge f\restriction_{\omega} \leq f$$

Can be more aggressive if witness ω strictly improves solution.

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_{\omega} \wedge f\restriction_{\omega} < f$$

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

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

Can be more aggressive if witness ω strictly improves solution.

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

- Applying ω should strictly decrease f.
- If so, don't need to show that $C\restriction_{\omega}$ holds!

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

Why is this sound?

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

Why is this sound?

1. Suppose α 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 ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \ \wedge \ f\restriction_\omega < f$$

Why is this sound?

1. Suppose α 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 ω s.t.

$$F \wedge \neg C \models F\restriction_{\omega} \wedge f\restriction_{\omega} < f$$

Why is this sound?

1. Suppose α 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 ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \;\wedge\; f\restriction_\omega < f$$

Why is this sound?

1. Suppose α 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 $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_{\omega} \wedge f\restriction_{\omega} < f$$

Why is this sound?

1. Suppose α 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 $f\big((\alpha \circ \omega) \circ \omega\big) < f(\alpha \circ \omega)$.
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 ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \ \wedge \ f\restriction_\omega < f$$

Why is this sound?

1. Suppose α 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 $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.
6. Otherwise $\big((\alpha \circ \omega) \circ \omega\big) \circ \omega$ satisfies F and $f\big(((\alpha \circ \omega) \circ \omega) \circ \omega\big) < f\big((\alpha \circ \omega) \circ \omega\big)$.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \ \wedge \ f\restriction_\omega < f$$

Why is this sound?

1. Suppose α 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 $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.
6. Otherwise $\big((\alpha \circ \omega) \circ \omega\big) \circ \omega$ satisfies F and $f\big(((\alpha \circ \omega) \circ \omega) \circ \omega\big) < f\big((\alpha \circ \omega) \circ \omega\big)$.
7. …

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Why is this sound?

1. Suppose α 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 $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.
6. Otherwise $\big((\alpha \circ \omega) \circ \omega\big) \circ \omega$ satisfies F and $f\big(((\alpha \circ \omega) \circ \omega) \circ \omega\big) < f\big((\alpha \circ \omega) \circ \omega\big)$.
7. ...
8. Can't go on forever, so finally reach α' satisfying $F \wedge C$.

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Now why is *this* sound?

- Same inductive proof as before, but nested.

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Now why is *this* sound?

- Same inductive proof as before, but nested.
- Or pick solution α minimizing f and argue by contradiction.

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Now why is *this* sound?

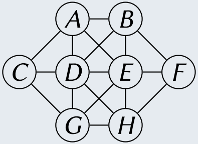
- Same inductive proof as before, but nested.
- Or pick solution α minimizing f and argue by contradiction.

Further extensions:

- Define dominance rule w.r.t. order independent of objective.
- Switch between different orders in same proof.
- See [BGMN22] for details.

Symmetry Elimination Constraints

The Crystal Maze Puzzle



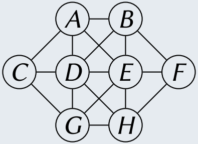
Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

Symmetry Elimination Constraints

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

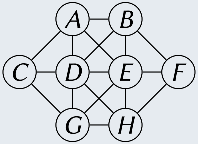
Symmetry Elimination Constraints

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

Are all of the above valid simultaneously?

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

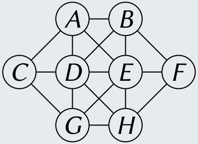
Symmetry Elimination Constraints

We can introduce these constraints inside the proof, rather than as part of the pseudo-Boolean model!

- Can use permutation of variable-values as the witness ω.
- The constraints give us the order.
- No group theory required!

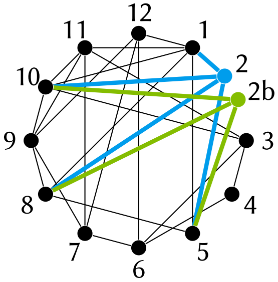
Research challenge: a CP toolchain supporting this.

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

Lazy Global Domination



Can ignore vertex 2b.

- Every neighbour of 2b is also a neighbour of 2.
- Not a symmetry, but a *dominance*.

Dominance rule can justify this.

- Even when detected dynamically during search.

Making Your Solver Output Proofs

`https://gitlab.com/MIAOresearch/VeriPB`

It's documented!

Worked examples in [GMM+20, EGMN20], and even more in Stephan Gocht's PhD thesis [Goc22].

More on

- SAT on Thursday at 15:00 in Benjamin Auditorium,
- CP on Friday at 12:00 in Taub 7.

We're happy to collaborate with you. We even have money for this!

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- "Lemmas", or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Things to proof log:

- Every single dedicated solving algorithm ever.
- The 400 remaining global constraints not implemented yet
- CP symmetries, dynamic symmetry handling, ...
- MaxSAT, MIP, SMT, ...

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- "Lemmas", or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Things to proof log:

- Every single dedicated solving algorithm ever.
- The 400 remaining global constraints not implemented yet
- CP symmetries, dynamic symmetry handling, …
- MaxSAT, MIP, SMT, …

The end.

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- "Lemmas", or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Things to proof log:

- Every single dedicated solving algorithm ever.
- The 400 remaining global constraints not implemented yet
- CP symmetries, dynamic symmetry handling, …
- MaxSAT, MIP, SMT, …

The end. Or rather, the beginning!

References I

[ABM+11] Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6):287–293, December 2011.

[AGJ+18] Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Metamorphic testing of constraint solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *Lecture Notes in Computer Science*, pages 727–736. Springer, August 2018.

[BGMN22] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, pages 3698–3707, February 2022.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

[BT19] Samuel R. Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, July 2019.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[CHH+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, August 2017.

[CMS17] Luís Cruz-Filipe, João P. Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135. Springer, April 2017.

References II

[Dub20] Catherine Dubois. Formally verified constraints solvers: a guided tour. CICM. Invited talk, 2020.

[EG21] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. In *Proceedings of the 22nd International Conference on Integer Programming and Combinatorial Optimization (IPCO '21)*, volume 12707 of *Lecture Notes in Computer Science*, pages 163–177. Springer, May 2021.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 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 *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

[GN03] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, March 2003.

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

[Goc22] Stephan Gocht. *Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning*. PhD thesis, Lund University, Lund, Sweden, June 2022. Available at `https://stephangocht.github.io/`.

[GSD19] Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, October 2019.

References III

[HHW13a] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.

[HHW13b] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[KM21] Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.

[MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

[MPP19] Ciaran McCreesh, William Pettersson, and Patrick Prosser. Understanding the empirical hardness of random optimisation problems. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 333–349. Springer, September 2019.

[Van08] Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM '08)*, 2008. Available at `http://isaim2008.unl.edu/index.php?page=proceedings`.

[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 International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.

https://gitlab.com/MIAOresearch/VeriPB

https://github.com/ciaranm/glasgow-constraint-solver

https://github.com/ciaranm/glasgow-subgraph-solver

https://bitbucket.org/krr/breakid