Is Your Combinatorial Search Algorithm Telling the Truth?

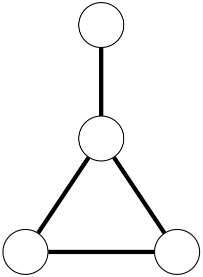
Ciaran McCreesh

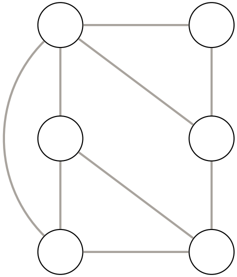
With numerous co-conspirators, including Bart Bogaerts, Jan Elffers, Stephan Gocht, Ross McBride, Matthew McIlree, Jakob Nordström, Andy Oertel, Patrick Prosser, and James Trimble

University of Glasgow

Royal Academy of Engineering

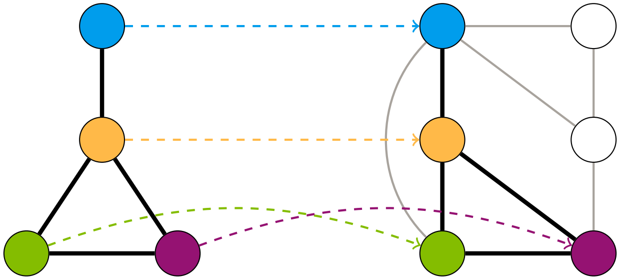
Subgraph Isomorphism





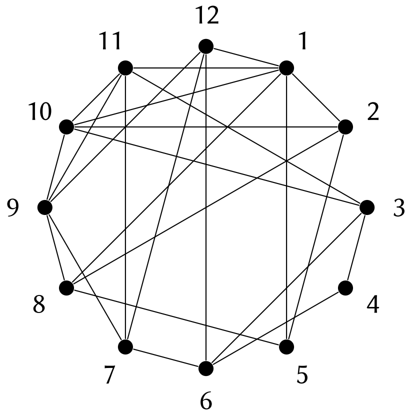
- Find the *pattern* inside the *target*.
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find *all* matches.

Subgraph Isomorphism

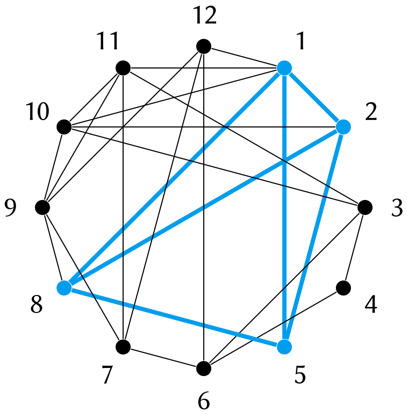


- Find the *pattern* inside the *target*.
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find *all* matches.

The Maximum Clique Problem



The Maximum Clique Problem



Constraint Programming

- We have a set of *variables*.
- Each variable has a finite *domain*.
- We have *constraints* between variables.
- Give each variable a value from its domain, satisfying all constraints (and maybe maximise some objective).
- Solve using inference and intelligent backtracking search.

Worst-Case Complexity vs Practice

- These problems are NP-hard, hard to approximate, etc.
- We can solve maximum clique on larger graphs than all-pairs shortest path.
- We don’t have a deep understanding as to why.

The Slight Problem...

- State of the art solvers occasionally produce incorrect answers.

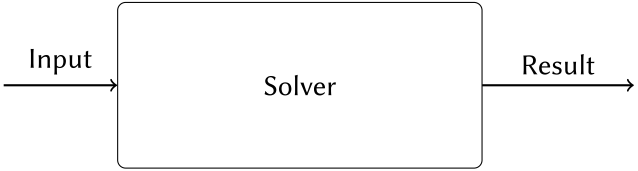
The Slight Problem…

- State of the art solvers occasionally produce incorrect answers.
- Extensive testing?
    - Only uncovers superficial bugs.
    - Empirically unsuccessful, even if people try really hard.
    - Even if you're sure, why should anyone believe you?

The Slight Problem…

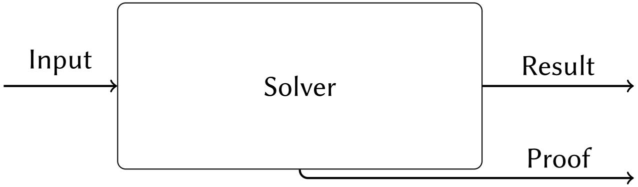
- State of the art solvers occasionally produce incorrect answers.
- Extensive testing?
  - Only uncovers superficial bugs.
  - Empirically unsuccessful, even if people try really hard.
  - Even if you're sure, why should anyone believe you?
- Formal methods?
  - Far from being able to handle state of the art algorithms and solvers.

Proof Logging



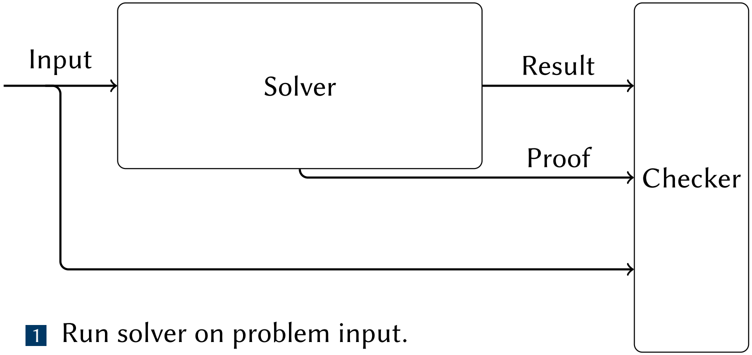
1. Run solver on problem input.

Proof Logging



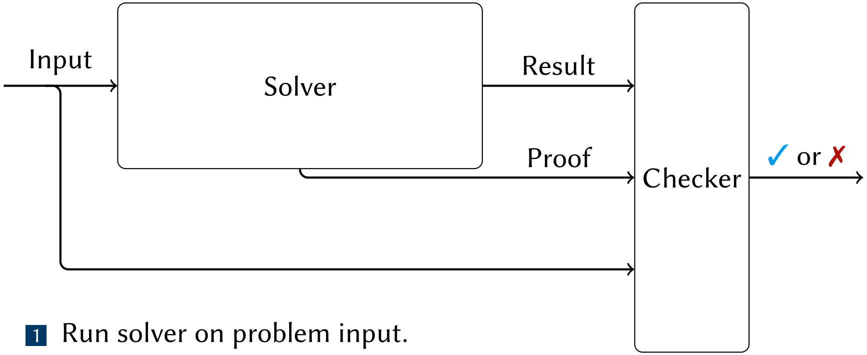
1. Run solver on problem input.
2. Get as output not only result but also proof.

Proof Logging



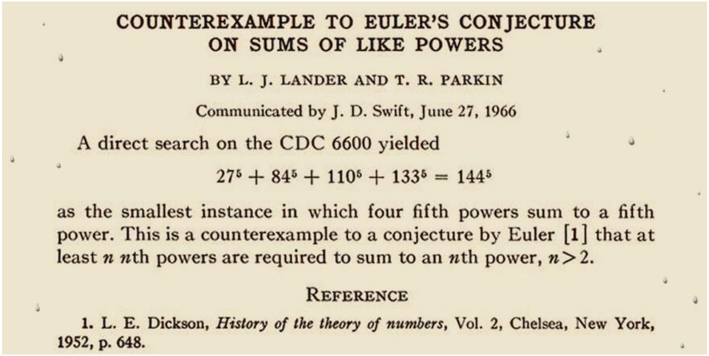
1. Run solver on problem input.
2. Get as output not only result but also proof.
3. Feed input + result + proof to proof checker.

Proof Logging



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.

What Is A Proof?


COUNTEREXAMPLE TO EULER'S CONJECTURE ON SUMS OF LIKE POWERS

BY L. J. LANDER AND T. R. PARKIN

Communicated by J. D. Swift, June 27, 1966

A direct search on the CDC 6600 yielded

$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

as the smallest instance in which four fifth powers sum to a fifth power. This is a counterexample to a conjecture by Euler [1] that at least n nth powers are required to sum to an nth power, $n > 2$.

REFERENCE

1. L. E. Dickson, *History of the theory of numbers*, Vol. 2, Chelsea, New York, 1952, p. 648.

The SAT Problem

- Variable x: takes value **true** (=1) or **false** (=0)
- Literal ℓ: variable x or its negation $\overline{x}$
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

The SAT Problem

Given a CNF formula F, is it satisfiable?

For instance, what about:

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

Proofs for SAT

For satisfiable instances: just specify a satisfying assignment.

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

- Each clause follows “obviously” from everything we know so far.
- Final clause is empty, meaning contradiction (written $\perp$).
- Means original formula must be inconsistent.

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

- $p \vee \overline{u}$ propagates $u \mapsto 0$.

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

- $p \lor \overline{u}$ propagates $u \mapsto 0$.
- $q \lor r$ propagates $r \mapsto 1$.

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

- $p \vee \overline{u}$ propagates $u \mapsto 0$.
- $q \vee r$ propagates $r \mapsto 1$.
- Then $\overline{r} \vee w$ propagates $w \mapsto 1$.

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

- $p \vee \overline{u}$ propagates $u \mapsto 0$.
- $q \vee r$ propagates $r \mapsto 1$.
- Then $\overline{r} \vee w$ propagates $w \mapsto 1$.
- No further unit propagations.

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ.

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

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

- $p \vee \overline{u}$ propagates $u \mapsto 0$.
- $q \vee r$ propagates $r \mapsto 1$.
- Then $\overline{r} \vee w$ propagates $w \mapsto 1$.
- No further unit propagations.

Proof checker should know how to unit propagate until saturation.

Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

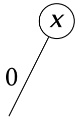
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

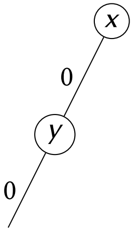


Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

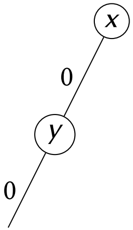


Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

“Proof trace”: when backtracking, write negation of guesses made.

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



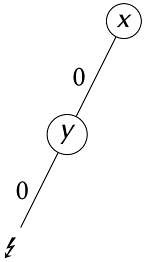
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

1. $x \vee y$



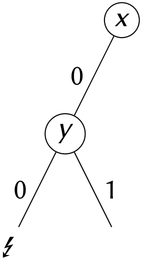
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

1. $x \vee y$



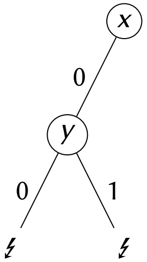
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

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



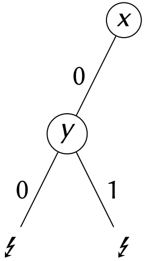
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

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



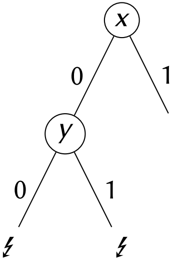
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

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



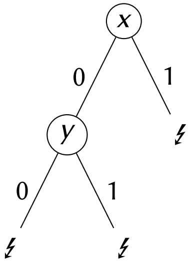
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

“Proof trace”: when backtracking, write negation of guesses made.

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

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



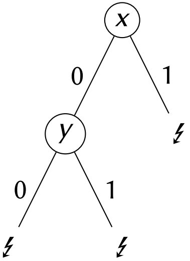
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

"Proof trace": when backtracking, write negation of guesses made.

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

1. $x \lor y$
2. $x \lor \overline{y}$
3. x
4. $\overline{x}$
5. $\bot$



Reverse Unit Propagation (RUP)

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

Reverse Unit Propagation (RUP)

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

Reverse unit propagation (RUP) clause

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

Reverse Unit Propagation (RUP)

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

Reverse unit propagation (RUP) clause

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

Backtrack clauses from DPLL solver generate a RUP proof.

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short 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 short proof of unsatisfiability for

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

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 short proof of unsatisfiability for

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

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 short proof of unsatisfiability for

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

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 short proof of unsatisfiability for

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

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 short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short 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. $\bot$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

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 short proof of unsatisfiability for

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

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 short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

Resolution Proofs

Fact

RUP proofs can be seen as shorthand for Resolution proofs.

Model axioms From the input

Resolution

$$\frac{x_1 \lor x_2 \lor \ldots \lor x_i \lor c \qquad \overline{c} \lor y_1 \lor y_2 \lor \ldots y_j}{x_1 \lor x_2 \lor \ldots \lor x_i \lor y_1 \lor y_2 \lor \ldots \lor y_j}$$

- To prove unsatisfiability: resolve until you reach the empty clause.

Resolution Can’t Count

- In subgraph isomorphism, can’t map a pattern vertex with n vertices into a target graph with $n - 1$ vertices.
- This requires exponential length proofs in resolution!

From CNF to Pseudo-Boolean

- A set of $\{0, 1\}$-valued variables x_i, 1 means true.
- Constraints are linear inequalities

$$\sum_i c_i x_i \geq C$$

- Write $\overline{x}_i$ to mean $1 - x_i$.
- Can rewrite CNF to pseudo-Boolean directly,

$$x_1 \vee \overline{x}_2 \vee x_3 \qquad \leftrightarrow \qquad x_1 + \overline{x}_2 + x_3 \geq 1$$

Cutting Planes Proofs

Model axioms From the input

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

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 $c \in \mathbb{N}^+$
$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

Interleaving RUP and Cutting Planes

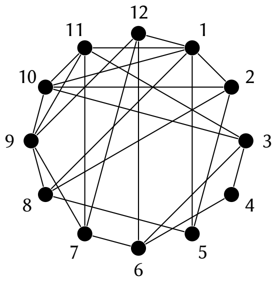
- Can define RUP similarly for pseudo-Boolean constraints.
- It does the same thing on clauses.
- Idea: use RUP for backtracking, and include explicit cutting planes steps to justify reasoning.

The VeriPB System

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

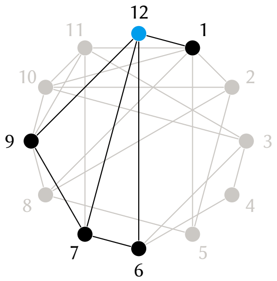
- MIT licence, written in Python with parsing in C++.
- Useful features like tracing and proof debugging.

(Roughly) How (Some) Maximum Clique Solvers Work



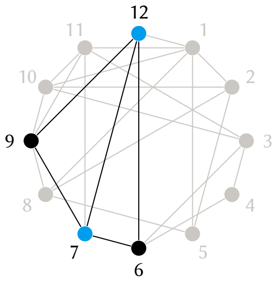
- Pick a vertex v, and branch on whether or not to include it. When acccepting, reject any vertices not adjacent to v.
- Remember largest clique found so far, and only look for bigger cliques.

(Roughly) How (Some) Maximum Clique Solvers Work



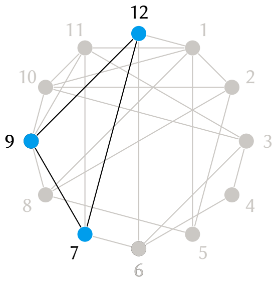
- Pick a vertex v, and branch on whether or not to include it. When acccepting, reject any vertices not adjacent to v.
- Remember largest clique found so far, and only look for bigger cliques.

(Roughly) How (Some) Maximum Clique Solvers Work



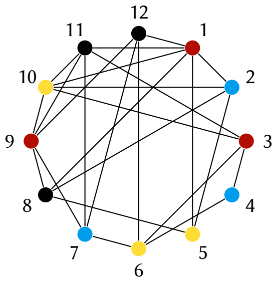
- Pick a vertex v, and branch on whether or not to include it. When acccepting, reject any vertices not adjacent to v.
- Remember largest clique found so far, and only look for bigger cliques.

(Roughly) How (Some) Maximum Clique Solvers Work



- Pick a vertex v, and branch on whether or not to include it. When acccepting, reject any vertices not adjacent to v.
- Remember largest clique found so far, and only look for bigger cliques.

(Roughly) How (Some) Maximum Clique Solvers Work



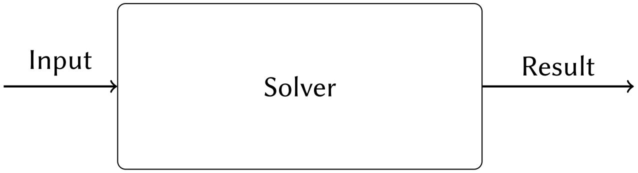
Given a *k*-colouring of a subgraph, that subgraph cannot have a clique of more than *k* vertices.

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

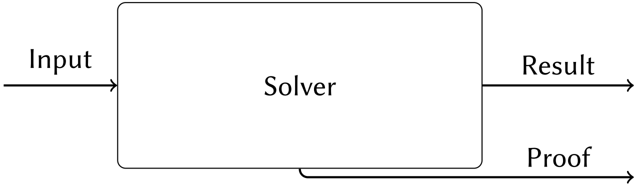
Making a Proof-Logging Clique Solver

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.

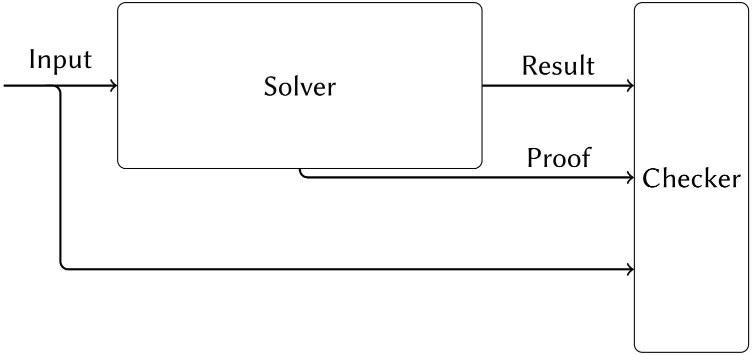
A Slightly Different Workflow



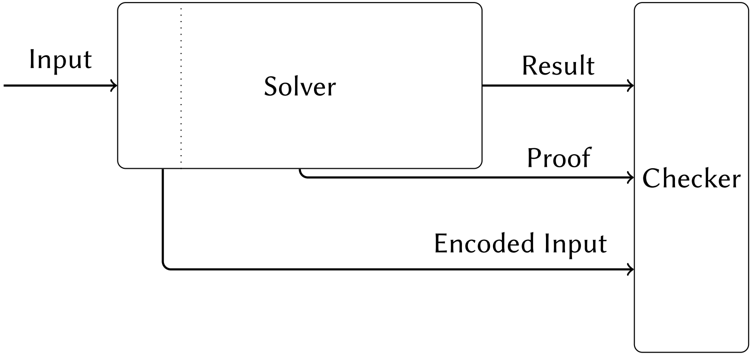
A Slightly Different Workflow



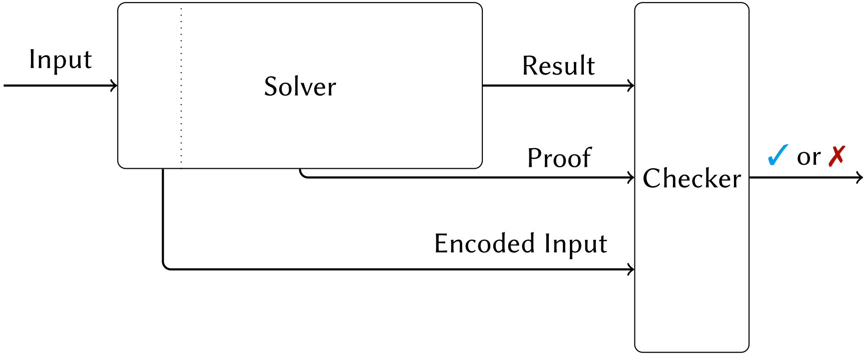
A Slightly Different Workflow



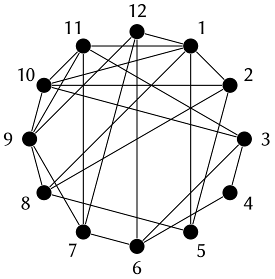
A Slightly Different Workflow



A Slightly Different Workflow



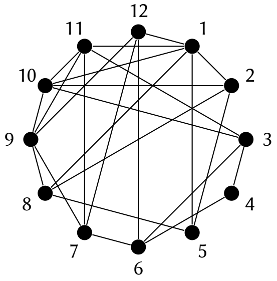
A Pseudo-Boolean Encoding for Clique (in OPB Format)



```
* #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
```

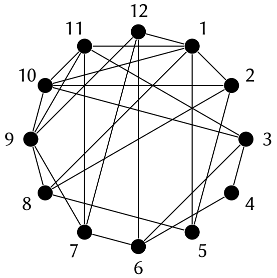
First Attempt at a Proof

```
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

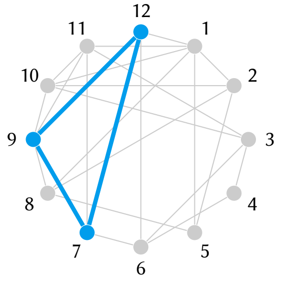
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Start with a header.
Load the 41 problem axioms.

First Attempt at a Proof

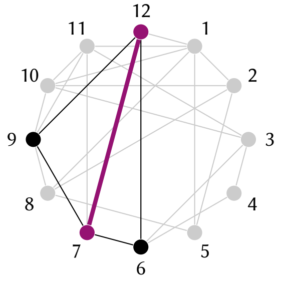
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Branch on 12, 7, 9.
Find a new incumbent.
Now looking for a ≥ 4 vertex clique.

First Attempt at a Proof

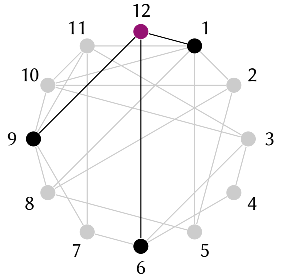
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 12, 7.
Only 6 and 9 feasible.
No ≥ 4 vertex clique possible.
Effectively this deletes the 7–12 edge.

First Attempt at a Proof

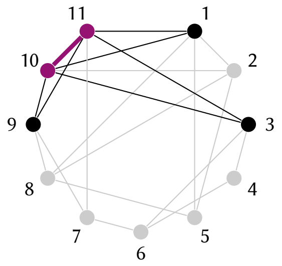
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 12.
Only 1, 6 and 9 feasible.
No ≥ 4 vertex clique possible.
Effectively this deletes vertex 12.

First Attempt at a Proof

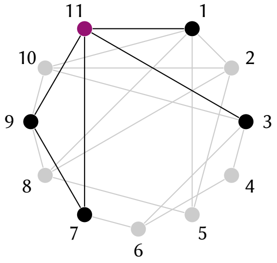
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Branch on 11 then 10.
Only 1, 3 and 9 feasible.
No ≥ 4 vertex clique possible.
Backtrack, deleting the edge.

First Attempt at a Proof

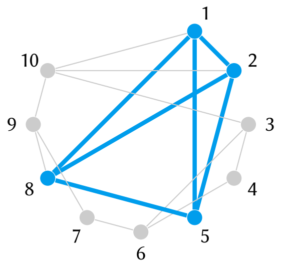
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 11.
Clearly no ≥ 4 clique.
Delete the vertex.

First Attempt at a Proof

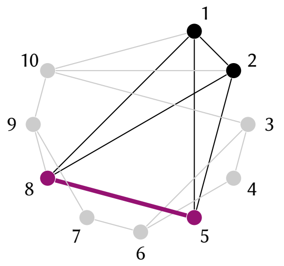
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Branch on 8, 5, 1, 2.
Find a new incumbent.
Now looking for a ≥ 5 vertex clique.

First Attempt at a Proof

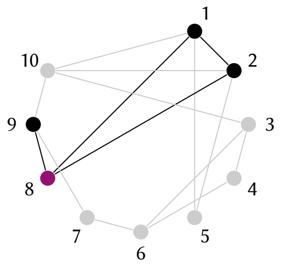
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 8, 5.
Only 4 vertices, can't have a ≥ 5 clique.
Delete the edge.

First Attempt at a Proof

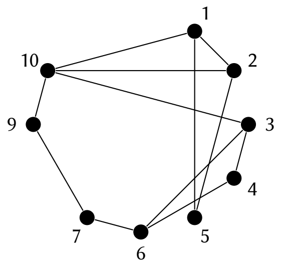
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 8.
Still not enough vertices.
Delete the vertex.

First Attempt at a Proof

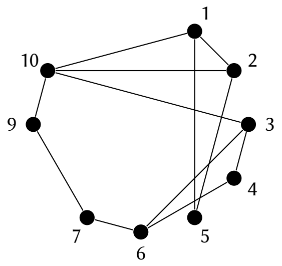
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Now obvious to solver that claim of ≥ 5 clique is contradictory (remaining vertices are 3-colourable).

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
```



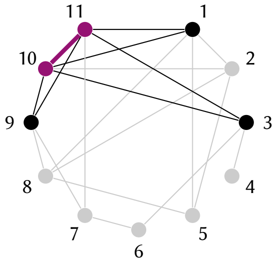
Assert previous line has derived contradiction, ending proof.

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
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
  ConstraintId 002: 1 ~x2 1 ~x3 >= 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.
```

Recovering At-Most-One Constraints

At-most-one constraints do not follow from reverse unit propagation.

Infeasible to list every colour class we *might* use in the input.

Recovering At-Most-One Constraints

At-most-one constraints do not follow from reverse unit propagation.

Infeasible to list every colour class we *might* use in the input.

But we can use cutting planes to recover colour classes lazily!

Recovering At-Most-One Constraints

At-most-one constraints do not follow from reverse unit propagation.

Infeasible to list every colour class we *might* use in the input.

But we can use cutting planes to recover colour classes lazily!

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

Recovering At-Most-One Constraints

At-most-one constraints do not follow from reverse unit propagation.

Infeasible to list every colour class we *might* use in the input.

But we can use cutting planes to recover colour classes lazily!

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

This generalises for arbitrarily large colour classes.

- Each non-edge is used exactly once, $v(v-1)$ additions.
- $v-3$ multiplications and $v-2$ divisions.

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

What This Looks Like

```
pseudo-Boolean proof version 1.2
f 41
o x12 x7 x9
u 1 ~x12 1 ~x7 >= 1 ;
* bound, colour classes [ x1 x6 x9 ]
p 7_{1≁6} 19_{1≁9} + 24_{6≁9} + 2 d
p 42_{obj} -1 +
u 1 ~x12 >= 1 ;
* bound, colour classes [ x1 x3 x9 ]
p 1_{1≁3} 19_{1≁9} + 21_{3≁9} + 2 d
p 42_{obj} -1 +
u 1 ~x11 1 ~x10 >= 1 ;
* bound, colour classes [ x1 x3 x7 ] [ x9 ]
p 1_{1≁3} 10_{1≁7} + 12_{3≁7} + 2 d
p 42_{obj} -1 +
u 1 ~x11 >= 1 ;
o x8 x5 x2 x1
u 1 ~x8 1 ~x5 >= 1 ;
* bound, colour classes [ x1 x9 ] [ x2 ]
p 53_{obj} 19_{1≁9} +
u 1 ~x8 >= 1 ;
* bound, colour classes [ x1 x3 x7 ] [ x2 x4 x9 ] [ x5 x6 x10 ]
p 1_{1≁3} 10_{1≁7} + 12_{3≁7} + 2 d
p 53_{obj} -1 +
p 4_{2≁4} 20_{2≁9} + 22_{4≁9} + 2 d
p 53_{obj} -3 + -1 +
p 9_{5≁6} 26_{5≁10} + 27_{6≁10} + 2 d
p 53_{obj} -5 + -3 + -1 +
u >= 1 ;
c -1
```

Verifying This Proof (For Real, This Time)

```
$ veripb --trace clique.opb clique-attempt-two.veripb
=== begin trace ===
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
  ConstraintId 002: 1 ~x2 1 ~x3 >= 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 x11 1 x12 >= 4
line 004: u 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: * bound, colour classes [ x1 x6 x9 ]
line 006: p 7 19 + 24 + 2 d
  ConstraintId 044: 1 ~x1 1 ~x6 1 ~x9 >= 2
line 007: p 42 43 +
  ConstraintId 045: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x8 1 x9 1 x10 1 x11 >= 3
...
  ConstraintId 061: 1 ~x5 1 ~x6 1 ~x10 >= 2
line 028: p 53 57 + 59 + 61 +
  ConstraintId 062: 1 x8 1 x11 1 x12 >= 2
line 029: u >= 1 ;
  ConstraintId 063: >= 1
line 030: 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".

What About Subgraph Isomorphism?

Each pattern vertex gets a target vertex:

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

What About Subgraph Isomorphism?

Each pattern vertex gets a target vertex:

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

Each target vertex may be used at most once:

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

What About Subgraph Isomorphism?

Each pattern vertex gets a target vertex:

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

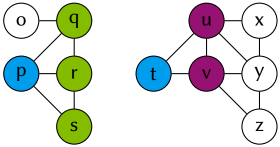
Each target vertex may be used at most once:

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

Adjacency constraints, if *p* is mapped to *t*, then *p*'s neighbours must be mapped to *t*'s neighbours:

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

Degree Reasoning in Cutting Planes

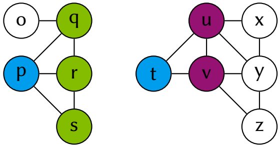


A pattern vertex p of degree $\deg(p)$ can never be mapped to a target vertex t of degree $\deg(p) - 1$ or lower in any subgraph isomorphism.

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

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

Degree Reasoning in Cutting Planes



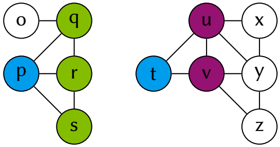
We have the three adjacency constraints,

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

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



Continuing with the sum

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

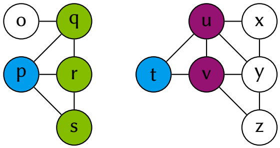
Due to injectivity,

$$-x_{o,u} + -x_{p,u} + -x_{q,u} + -x_{r,u} + -x_{s,u} \geq -1$$
$$-x_{o,v} + -x_{p,v} + -x_{q,v} + -x_{r,v} + -x_{s,v} \geq -1$$

Add all these together, getting

$$3\overline{x}_{p,t} + -x_{o,u} + -x_{o,v} + -x_{p,u} + -x_{p,v} \geq 1$$

Degree Reasoning in Cutting Planes



We're more or less there. We have:

$$3\overline{x}_{p,t} + -x_{o,u} + -x_{o,v} + -x_{p,u} + -x_{p,v} \geq 1$$

Add the literal axioms $x_{o,u} \geq 0$, $x_{o,v} \geq 0$, $x_{p,u} \geq 0$ and $x_{p,v} \geq 0$ 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$_{p\sim t:q}$ 19$_{p\sim t:r}$ + 20$_{p\sim t:s}$ +    * sum adjacency constraints
  12$_{inj(u)}$ + 13$_{inj(v)}$ +            * sum injectivity constraints
  xo_u + xo_v +                  * cancel stray xo_*
  xp_u + xp_v +                  * cancel stray xp_*
  3 d                            * divide, and we're done
```

Or we can ask VeriPB to do the last bit of simplification automatically:

```
p 18$_{p\sim t:q}$ 19$_{p\sim t:r}$ + 20$_{p\sim t:s}$ +    * sum adjacency constraints
  12$_{inj(u)}$ + 13$_{inj(v)}$ +            * sum injectivity constraints
j -1 1 ~xp_t >= 1 ;              * desired conclusion is implied
```

Other Forms of Reasoning

We can also log all of the other things state of the art subgraph solvers do:

- Injectivity reasoning and filtering.
- Distance filtering.
- Neighbourhood degree sequences.
- Path filtering.
- Supplemental graphs.

Other Forms of Reasoning

We can also log all of the other things state of the art subgraph solvers do:

- Injectivity reasoning and 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.

Extension Variables

Suppose we want new, fresh variable a encoding

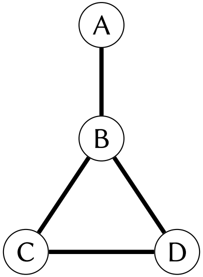
$$a \Leftrightarrow (3x + 2y + z + w \geq 3)$$

Introduce constraints

$$3\overline{a} + 3x + 2y + z + w \geq 3 \qquad 5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5$$

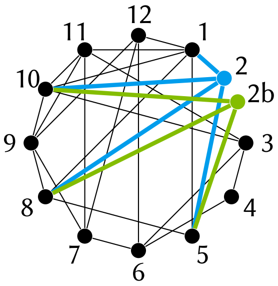
Should be fine, so long as a hasn't been used before.

Symmetries



- If a solution exists, a solution where $C < D$ exists.

Dominance



Can ignore vertex 2b.

- Every neighbour of 2b is also a neighbour of 2.

Progress So Far on World Domination

- SAT with symmetries, cardinality, XOR reasoning, MaxSAT.
  - Uncovered several undetected bugs in state of the art solvers.
  - Can't do MaxSAT hitting set solvers yet, MIP isn't proof logged.
- Certified translations from pseudo-Boolean to CNF.
- Clique, subgraph isomorphism, maximum common (connected) induced subgraph.
- Constraint programming.
  - Large integer variables.
  - Absolute value, all different, circuit, comparison, element, linear equality and inequality, minimum and maximum, regular, smart table constraints.
- In progress: MIP preprocessing for pseudo-Boolean problems, dynamic programming, the remaining 400 constraints for CP, …

What Reasoning Can We Justify?

- With extension variables, as strong as Extended Frege.
- So according to theorists, we can simulate pretty much everything.

What Reasoning Can We Justify?

- With extension variables, as strong as Extended Frege.
- So according to theorists, we can simulate pretty much everything.
    - Up to a polynomial factor…

What Reasoning Can We Justify?

- With extension variables, as strong as Extended Frege.
- So according to theorists, we can simulate pretty much everything.
    - Up to a polynomial factor…
- Except dominance is apparently even stronger?

What Reasoning Can We Justify Efficiently?

- Quadratic overheads are unpleasant.
- Cutting planes is very good at justifying combinatorial arguments.
- It's not really clear why.

Verifying the Verifier

- How do we know the encoding is correct?
- How do we know the verifier is correct?
- How do we know the proof system is sound?

Proof Trimming

- Proofs can be really really really big.
- Often many steps end up being redundant for the final proof.
- Could we make a tool that turns a really really really big proof into a really big proof?

Counting and Sampling without Enumerating

- The proof system deals with unsatisfiability.
- Satisfiability is easy, just give a solution.
- Optimisation is a solution and a proof there's nothing better.
- Enumeration is a solution list, and a proof there's nothing else.
- How do we provide a count without enumerating?

Going the Other Way

- Can we use proofs to understand solver behaviour?
  - Why solvers work so well when they shouldn't.
  - Why solvers perform so badly when they shouldn't.
- Explainability?

Where We're At

- Can verify *solutions* from state of the art combinatorial solving algorithms, in a unified proof system.
- Found many undetected bugs in widely used solvers.
  - Including in algorithms that have been "proved" correct.

Where We're At

- Can verify *solutions* from state of the art combinatorial solving algorithms, in a unified proof system.
- Found many undetected bugs in widely used solvers.
    - Including in algorithms that have been "proved" correct.
- Not being either proof logged or formally verified should be considered socially unacceptable.

Where We're At

- Can verify *solutions* from state of the art combinatorial solving algorithms, in a unified proof system.
- Found many undetected bugs in widely used solvers.
  - Including in algorithms that have been "proved" correct.
- Not being either proof logged or formally verified should be considered socially unacceptable.
- Perhaps studying proof logs can help explain why solvers work so well?

Getting Involved

- Glasgow has funding for PhD students starting this October.
- I will be hiring for a three year postdoc position as soon as the paperwork is finished.
- Install VeriPB: `https://gitlab.com/MIAOresearch/software/VeriPB`
- Documentation: `https://satcompetition.github.io/2023/downloads/proposals/veripb.pdf`
- Tutorial: `https://www.youtube.com/watch?v=s_5BIi4I22w`

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk