Proof Logging For Things That Aren't SAT

Ciaran McCreesh

And numerous unindicted co-conspirators, including
Bart Bogaerts, Stephan Gocht, Ross McBride,
James Trimble, Jakob Nordström, and Patrick Prosser

University of Glasgow

Royal Academy of Engineering

The Slide That Got Me Into Trouble

- For somewhere between 0.1% (my clique experiments) and 1.28% (MiniZinc challenge 2021) of instances, we get the wrong solution.
  - False claims of unsatisfiability.
  - False claims of optimality.
  - Infeasible solutions produced.
  - The same solver run on the same instance on the same hardware twice in a row can claim both unsatisfiability and satisfiability.
- This includes academic and commercial CP and MIP solvers.
- Extensive testing hasn't fixed this.
- Formal methods are far from being able to handle solvers.
- The situation for SAT solvers is somewhat better.

Proof Logging

- Certifying algorithms:
  - Must produce a proof alongside an output.
  - Verify outputs, not solvers.
  - Unsat is the hard part.
- A variety of formats for SAT: . . . , DRAT, FRAT, ...
- Huge success for SAT solving.

World Domination Plans

- Proof log all the things!
  - OK, we'll stick to NP decision and optimisation for now.
- Support both retrofitting and proof-driven development.
- Call it "auditable solving".

Opinionated Requirements

1. Work with what solvers actually do, not idealised algorithms.
2. No need for a new proof format for every new kind of algorithm.
   - At least a hundred subgraph-finding algorithms, each of which does a different kind of reasoning (colourings, neighbourhood degrees, paths, connectivity, supplemental graphs, …).
     - The "state of the art" is often buggy…
   - Constraint programming has 423 different global constraints, many of which have several different propagators.
     - Some of which are buggy, and at least one has faulty theory behind it…
3. Proof format must still be simple and well-founded.
   - Need to be able to trust the verifier.
   - Interactions between features can be subtle: even deletions aren't that easy to get right.

Reusing DRAT Isn't Feasible

- Closely tied to how MiniSAT works:
  - Proofs are (mostly) sequences of learned clauses.
  - Something special and strange happens to learned unit clauses.
- Stronger reasoning is hard in theory and in practice.
- Preprocessing is possible (sometimes), but not easy.
  - We need to do full-on reformulation, though.
- Not clear how to do optimisation, enumeration, counting, …

Unexpected and Remarkable Claim

- We can do everything we want with a proof format which is only slightly more sophisticated than DRAT.

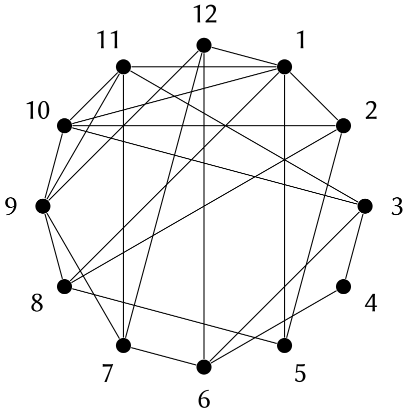
Unexpected and Remarkable Claim

- We can do everything we want with a proof format which is only slightly more sophisticated than DRAT.
- Using proof logs during development leads to faster development than not doing proof logging.

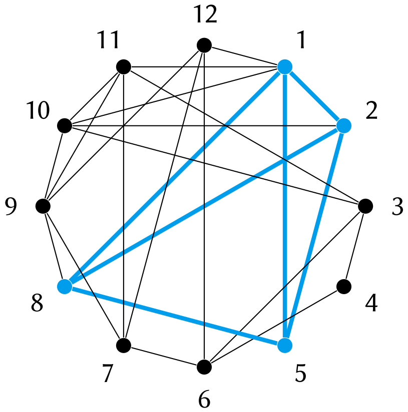
Unexpected and Remarkable Claim

- We can do everything we want with a proof format which is only slightly more sophisticated than DRAT.
- Using proof logs during development leads to faster development than not doing proof logging.
- You should make your students and postdocs adopt this technology right now.

The Maximum Clique Problem



The Maximum Clique Problem



The Certifying Process

- Express the problem in pseudo-Boolean form (0/1 integer linear program; a superset of CNF):
  - A set of $\{0, 1\}$-valued variables x_i.
  - We define $\overline{x}_i = 1 - x_i$.
  - Integer linear inequalities $\sum_i c_i x_i \geq C$.
  - Optionally, an objective min $\sum_i c_i x_i$.
- Write this out as an OPB file.
- Provide a proof log for this OPB file.
  - For unsat decision instances, prove $0 \geq 1$.
  - Can also log sat decision instances, enumeration, and optimisation.
- Feed the OPB file and the proof log to VeriPB.

In Action...

```
$ ./glasgow_clique_solver p_hat500-2.clq
nodes = 108217
clique = 37 59 63 68 71 102 124 133 137 150 160 186 206 222 231 238
runtime = 175ms
```

In Action. . .

```
$ ./glasgow_clique_solver p_hat500-2.clq
nodes = 108217
clique = 37 59 63 68 71 102 124 133 137 150 160 186 206 222 231 238
runtime = 175ms

$ ./glasgow_clique_solver p_hat500-2.clq --prove proof
runtime = 16,347ms
```

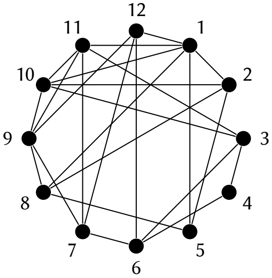
In Action...

```
$ ./glasgow_clique_solver p_hat500-2.clq
nodes = 108217
clique = 37 59 63 68 71 102 124 133 137 150 160 186 206 222 231 238
runtime = 175ms

$ ./glasgow_clique_solver p_hat500-2.clq --prove proof
runtime = 16,347ms

$ ls -lh proof.log proof.opb
-rw-rw-r-- 1 ciaranm ciaranm 558M Aug 23 21:43 proof.log
-rw-rw-r-- 1 ciaranm ciaranm 1.4M Aug 23 21:42 proof.opb
```

In Action...

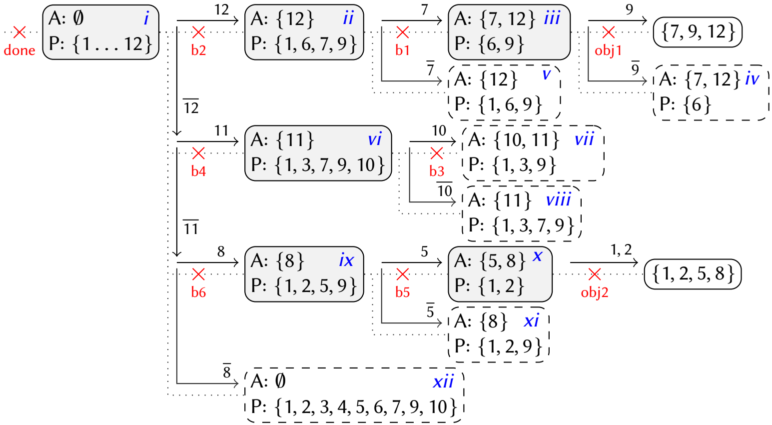
```
$ ./glasgow_clique_solver p_hat500-2.clq
nodes = 108217
clique = 37 59 63 68 71 102 124 133 137 150 160 186 206 222 231 238
runtime = 175ms

$ ./glasgow_clique_solver p_hat500-2.clq --prove proof
runtime = 16,347ms

$ ls -lh proof.log proof.opb
-rw-rw-r-- 1 ciaranm ciaranm 558M Aug 23 21:43 proof.log
-rw-rw-r-- 1 ciaranm ciaranm 1.4M Aug 23 21:42 proof.opb

$ veripb proof.opb proof.log
INFO:root:total time: 428.89s
maximal used database memory: 0.003 GB
Verification succeeded.
```

A Pseudo-Boolean Encoding




```
* #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
```

A Search Tree



A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

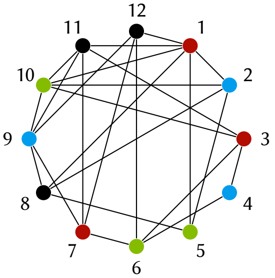
⇝ done

A Proof Describing This Search Tree

```
pseudo-Boolean proof version 1.0
f 41 0
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 done 0
```

⇝ done

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 from reverse unit propagation.

Bounds Using Cutting Planes

```
pseudo-Boolean proof version 1.0
f 41 0
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
* at most one [ x1 x3 x9 ]
p nonadj1_3 2 * nonadj1_9 + nonadj3_9 + 3 d                    ⇝ tmp1
p obj1 tmp1 +
u 1 ~x11 1 ~x10 >= 1 ;                                          ⇝ b3
* at-most-one [ x1 x3 x7 ]
p nonadj1_3 2 * nonadj1_7 + nonadj3_7 + 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 0
```

Results

- Implemented in the Glasgow Subgraph Solver.
  - Bit-parallel, can perform a colouring and recursive call in under a microsecond.
- 59 of the 80 DIMACS instances take under 1,000 seconds to solve without logging.
- Produced and verified proofs for 57 of these 59 instances (the other two reached 1TByte disk space).
- Mean slowdown from proof logging is 80.1 (due to disk I/O).
- Mean verification slowdown a further 10.1.
- Approximate implementation effort: one Masters student.

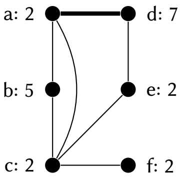
Maximum Clique in General

- There are a lot of maximum clique algorithms:
  - Different search orders.
  - Different bound functions.
  - Different data structures.
  - Priming using local search.
- Once you've implemented proof logging for one, the rest require very little effort.

Maximal Clique Enumeration

- There are contradictory results for several graphs in the literature…
- For proof logging:
  - Maximality property is easily expressed in PB ("either take *v*, or at least one of *v*'s neighbours").
  - Proof log every backtrack and every solution.
  - No need to proof log the "not set".
- This works for *all* maximal clique algorithms.
- Implementation effort: roughly one day for someone who had never implemented any kind of proof logging before.
- Works for standard benchmark graphs of up to 10,000 vertices.

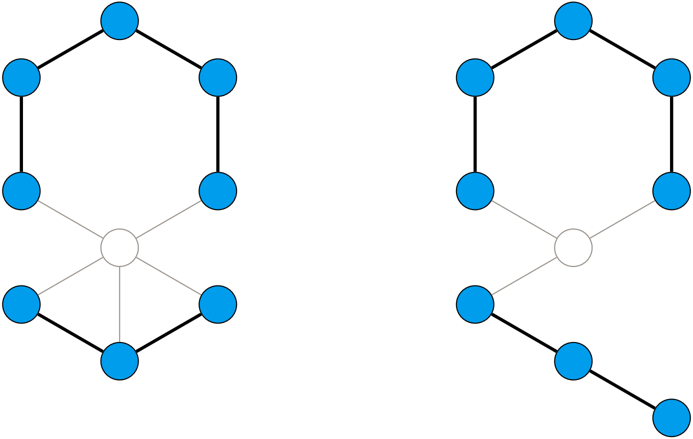
Maximum Weight Clique



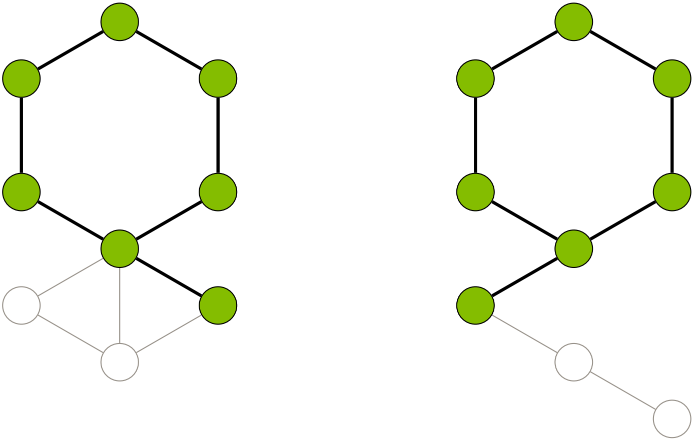

```
pseudo-Boolean proof version 1.0
f 8 0
o xa xd                                                  ⇝ obj
p nonadja_e 2 * nonadja_f + nonadje_f + 3 d 2 *          ⇝ cc1
p nonadjb_d 5 *                                          ⇝ cc2
p nonadjc_d 2 *                                          ⇝ cc3
p obj cc1 + cc2 + cc3 +                                  ⇝ done
c done 0
```

- Colour classes have weights.
  - Just multiply a colour class by its weight.
- Vertices can split their weights between colour classes.
  - That's fine, no changes needed.
- Implementation effort: an afternoon, having seen roughly how it's done for unweighted cliques.

Maximum Common Subgraph



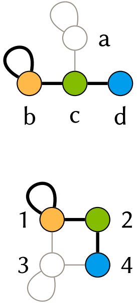
Maximum Common Connected Subgraph

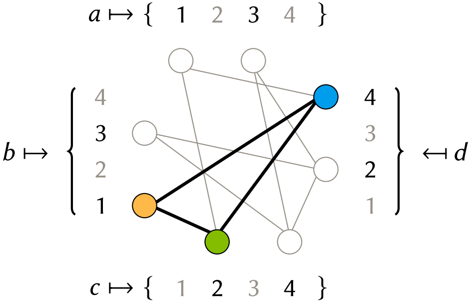


The McSplit Solver

- A CP forward checker, but with different underlying data structures.
- All-different-except-$\bot$ as a bound function.
- Connected is handled by a combination of branching rules and propagation.
  - Slightly awkward to encode in PB: requires dependent auxiliary variables.
  - Reverse unit propagation handles it without help.

Reduction to Clique



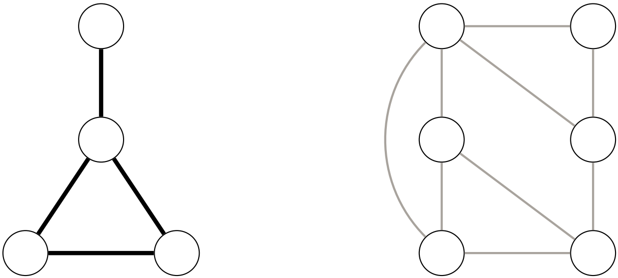


- We can encode this reduction using cutting planes rules. No need for a different OPB file.
- The clique solver does not need to be modified.
- This even works for connectivity.

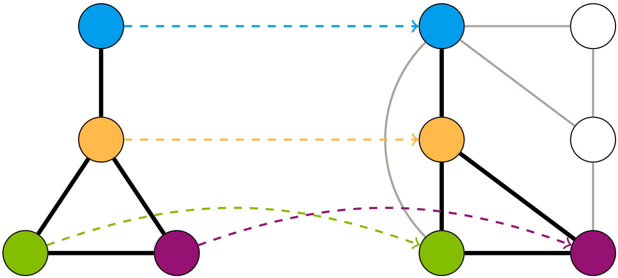
Results

- McSplit: implemented in a day by someone with no prior proof logging experience.
  - 16,300 instances, proof logging slowdowns of 67.0 and 298.9.
  - McSplit can make five million recursive calls per second.
  - Verification slowdown of 13.4 and 21.6.
- Clique: implemented alongside the algorithm in under a day.
  - 11,400 instances verified, proof logging slowdown of 28.6 and 39.7.
  - Verification slowdown of 11.3 and 73.1.
  - Caught a bug in the implementation that testing had missed.

Subgraph Isomorphism



Subgraph Isomorphism



Subgraph Isomorphism in Pseudo-Boolean Form

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

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

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

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

- 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:

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

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 $\mathsf{N}(p) = \{q, r, s\}$ and $\mathsf{N}(t) = \{u, v\}$.
- 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, at most one of $x_{q,u}$, $x_{r,u}$, and $x_{s,u}$ can be true, and similarly for v.
- Add both these injectivity constraints, getting

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

Degree Reasoning in Cutting Planes

- Continuing with the sum of sums

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

- Add the literal axioms $x_i \geq 0$ 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 Cutting Planes

```
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 Cutting Planes

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

Other Forms of Reasoning

- We can also do:
  - All-different.
  - Distance filtering.
  - Neighbourhood degree sequences.
  - Path filtering.
  - Supplemental graphs.
- Proof steps are "efficient" using cutting planes.

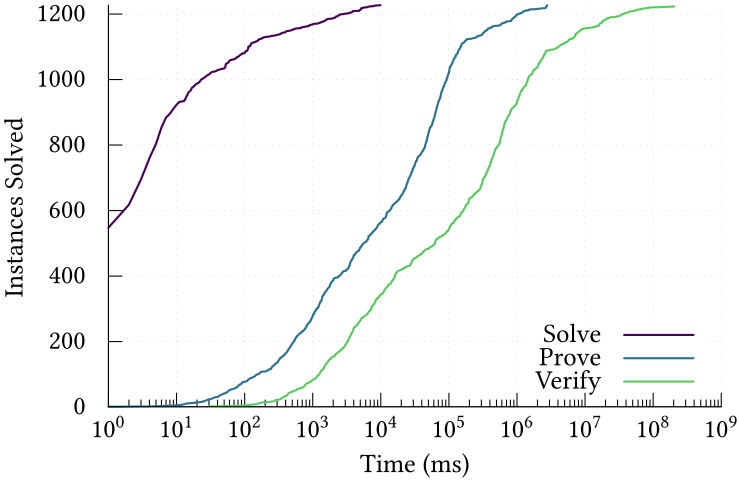
It Works!

- Able to produce and verify Glasgow Subgraph Solver proofs for medium-sized instances for the first time.
- Can't guarantee the solver is free of bugs, but if it ever outputs an incorrect answer, we will detect it.
- No changes to the reasoning carried out by the solver.

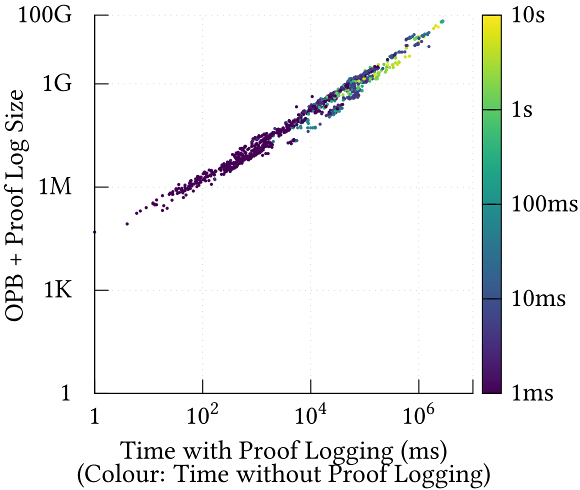
Problem Instances

- The Pseudo-Boolean models can be large: had to restrict to instances with no more than 260 vertices in the target graph.
- Took enumeration instances which could be solved without proof logging in under ten seconds.
- 1,227 instances from Solnon's benchmark collection:
  - 789 unsatisfiable, up to 50,635,140 solutions in the rest.
  - 498 instances solved without guessing.
  - Hardest solved satisfiable and unsatisfiable instances required 53,605,482 and 2,074,386 recursive calls.

Hard Disks Make This Quite Slow



Hard Disks Make This Quite Slow



Constraint Programming

- Integer domains.
- Rich constraints with different propagation algorithms.
- Need to reformulate constraints and models.

Extension Variables

- Given a pseudo-Boolean constraint C and a fresh variable y, introduce

$$y \leftrightarrow C$$

- Straightforward use of redundance-based strengthening.

Expressing CP Variables in Pseudo-Boolean Form

- Given $X \in \{1, 2, 3\}$, create $x_{=1}$, $x_{=2}$ and $x_{=3}$?
- Would also want $x_{\geq 1}$ and $x_{\geq 2}$ for convenience.
- Doesn't work for large domains whose bounds are trimmed during search.

Binary Encodings?

- Given A with domain $\{-3 \ldots 9\}$, how about

$$-32a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} + 16a_{\text{b4}} \geq -3 \text{ and}$$
$$32a_{\text{neg}} + -1a_{\text{b0}} + -2a_{\text{b1}} + -4a_{\text{b2}} + -8a_{\text{b3}} + -16a_{\text{b4}} \geq -9.$$

- Weakly propagating, but that doesn't matter.
- Really annoying for proofs.

Lazily Introducing Direct Variables

- Go with the binary encoding.
- 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.

Propagators

- All different, linear inequalities: cutting planes.
- Table, absolute value, minimum / maximum: reverse unit propagation.
- Element, GAC linear equalities: reformulation then reverse unit propagation.
- Not equals: lazy reformulation.

Reformulation

- Gratuitous use of extension variables.
- Sufficient for, e.g. tabulation of constraints.
- Also allows for more compact not-equals on large domains.

Symmetries

- We could do proof logging for symmetry constraints, without including them in the OPB file.

Open Problems and Ongoing Work

- Verification:
  - A formally verified verifier.
  - Verifying pseudo-Boolean encodings.
  - Performance.
- Proof-related:
  - "Lemmas", or substitution proofs?
  - Counting that isn't just enumeration.
  - Approximate counting, uniform sampling, etc? Pareto fronts?
  - Proof trimming or minimisation?
- Things to proof log:
  - Symmetric explanation learning.
  - The 400 remaining global constraints I've not done yet.
  - Every single dedicated solving algorithm ever.
- Beyond proofs:
  - Proof mining for experimental algorithmics?

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk

University of Glasgow

Royal Academy of Engineering