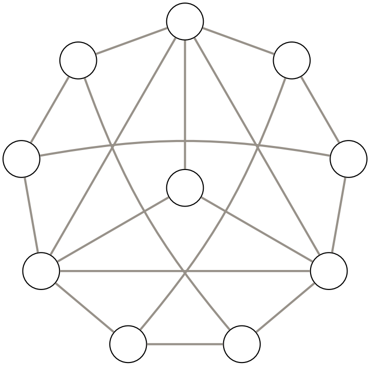
Proof Logging for Subgraph-Finding Algorithms

Ciaran McCreesh

University of Glasgow

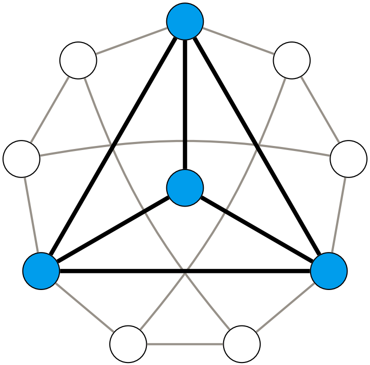
Royal Academy of Engineering

Maximum Clique



Prosser: Exact Algorithms for Maximum Clique: A Computational Study, Algorithms 5(4) (2012)

Maximum Clique



A Brief and Incomplete Guide to Clique Solving (1/4)

Recursive maximum clique algorithm:

- Pick a vertex v.
- Either v is in the clique...
  - Throw away every vertex not adjacent to v.
  - If vertices remain, recurse.
- ...or v is not in the clique, so
  - Throw v away and pick another vertex.

A Brief and Incomplete Guide to Clique Solving (2/4)

Key data structures:

- Growing clique C.
- Shrinking set of potential vertices P.
  - All the vertices we haven't thrown away yet.
  - Every $v \in P$ is adjacent to every $w \in C$.

A Brief and Incomplete Guide to Clique Solving (2/4)

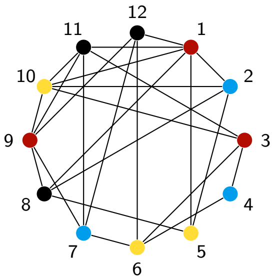
Key data structures:

- Growing clique C.
- Shrinking set of potential vertices P.
  - All the vertices we haven't thrown away yet.
  - Every $v \in P$ is adjacent to every $w \in C$.

Branch and bound:

- Remember the biggest clique $C^\star$ found so far.
- If $|C| + |P| \leq |C^\star|$, no need to keep going.

A Brief and Incomplete Guide to Clique Solving (3/4)



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

We can use $|C| + \#colours(P)$ as a bound, for any colouring.

Prosser: Exact Algorithms for Maximum Clique: A Computational Study, Algorithms 5(4) (2012)

A Brief and Incomplete Guide to Clique Solving (4/4)

- This brings us to 1997.
- Many improvements since then:
  - better bound functions,
  - clever vertex selection heuristics,
  - efficient data structures,
  - local search,
  - ...
- But key ideas for proof logging can be explained without worrying about such things.

Demotivation

My first experience of research: a summer internship reimplementing a clique enumeration algorithm from the literature.

My code produced the “wrong” answer on a few instances.

Demotivation

My first experience of research: a summer internship reimplementing a clique enumeration algorithm from the literature.

My code produced the “wrong” answer on a few instances.

I spent a month trying to find and fix it.

Demotivation

My first experience of research: a summer internship reimplementing a clique enumeration algorithm from the literature.

My code produced the “wrong” answer on a few instances.

I spent a month trying to find and fix it.

The published answers were wrong.

Demotivation

My first experience of research: a summer internship reimplementing a clique enumeration algorithm from the literature.

My code produced the "wrong" answer on a few instances.

I spent a month trying to find and fix it.

The published answers were wrong.

Five years later, my first small grant: trying to use computational experiments to understand the empirical hardness of maximum clique.

Demotivation

My first experience of research: a summer internship reimplementing a clique enumeration algorithm from the literature.

My code produced the “wrong” answer on a few instances.

I spent a month trying to find and fix it.

The published answers were wrong.

Five years later, my first small grant: trying to use computational experiments to understand the empirical hardness of maximum clique.

A billion problem instances solved using “state of the art” solver and £20,000 of supercomputer time (which used to be a lot, before ChatGPT came along).

Demotivation

My first experience of research: a summer internship reimplementing a clique enumeration algorithm from the literature.

My code produced the "wrong" answer on a few instances.

I spent a month trying to find and fix it.

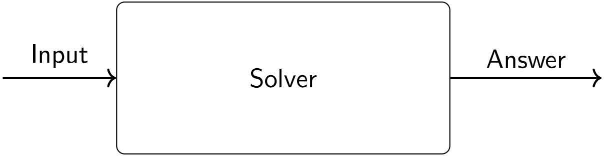
The published answers were wrong.

Five years later, my first small grant: trying to use computational experiments to understand the empirical hardness of maximum clique.

A billion problem instances solved using "state of the art" solver and £20,000 of supercomputer time (which used to be a lot, before ChatGPT came along).

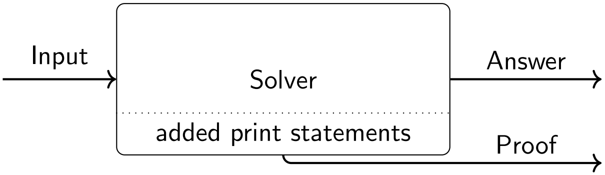
For around a thousand instances, the solver gave the wrong answer.

Proof Logging



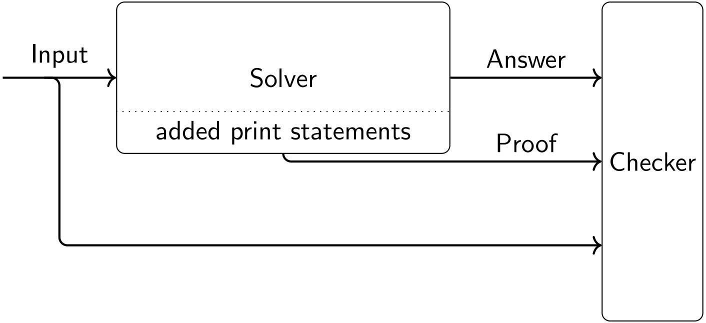
1. Run solver on problem input.

Proof Logging



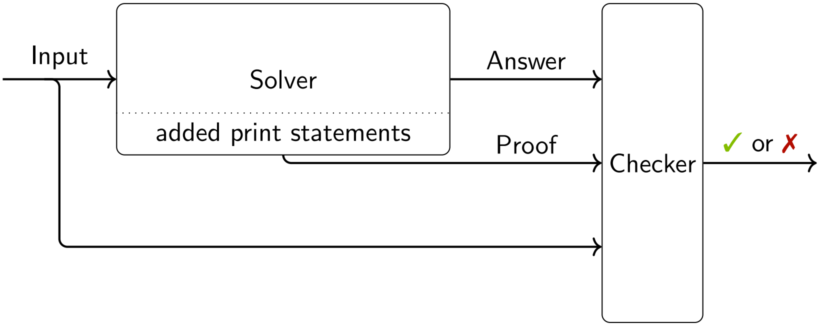
1. Run solver on problem input.
2. Solver also prints out a proof as part of its output.

Proof Logging



1. Run solver on problem input.
2. Solver also prints out a proof as part of its output.
3. Feed input + solution + proof to proof checker.

Proof Logging



1. Run solver on problem input.
2. Solver also prints out a proof as part of its output.
3. Feed input + solution + proof to proof checker.
4. Verify that proof checker says solution 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.

What is a Proof?

What's the smallest instance in which four fifth powers sum to a fifth power?

ChatGPT said: You're referring to a taxicab number-type problem, specifically for fifth powers:
What is the smallest positive integer solution to the equation $a^5 + b^5 + c^5 + d^5 = e^5$ where a, b, c, d, e are positive integers?
This is a well-known problem in number theory. The first known and smallest such solution was found in 1966 by L.J. Lander and T.R. Parkin:

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

Let's break that down:

$$\begin{aligned} 27^5 &= 14348907 \\ 84^5 &= 418211328 \\ 110^5 &= 1610510000 \\ 133^5 &= 433494437 \\ 14348907 + 418211328 + 1610510000 + 433494437 &= 6191730672 \\ 144^5 &= 6191730672 \end{aligned}$$

It checks out.

What is a Proof?

What's the smallest instance in which four fifth powers sum to a fifth power?

ChatGPT said: You're referring to a taxicab number-type problem, specifically for fifth powers:
What is the smallest positive integer solution to the equation $a^5 + b^5 + c^5 + d^5 = e^5$ where a, b, c, d, e are positive integers?
This is a well-known problem in number theory. The first known and smallest such solution was found in 1966 by L.J. Lander and T.R. Parkin:

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

Let's break that down:

$$\begin{aligned} 27^5 &= 14348907 \\ 84^5 &= 418211328 \\ 110^5 &= 1610510000 \\ 133^5 &= 433494437 \\ 14348907 + 418211328 + 1610510000 + 433494437 &= 6191730672 \\ 144^5 &= 6191730672 \end{aligned}$$

WRONG!

It checks out.

What is a Proof?

What's the smallest instance in which four fifth powers sum to a fifth power?

ChatGPT said: You're referring to a taxicab number-type problem, specifically for fifth powers:
What is the smallest positive integer solution to the equation $a^5 + b^5 + c^5 + d^5 = e^5$ where a, b, c, d, e are positive integers?
This is a well-known problem in number theory. The first known and smallest such solution was found in 1966 by L.J. Lander and T.R. Parkin:

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

Let's break that down:

$$\begin{aligned} 27^5 &= 14348907 \\ 84^5 &= 418211328 \\ 110^5 &= 1610510000 \\ 133^5 &= 433494437 && \text{WRONG!} \\ 14348907 + 418211328 + 1610510000 + 433494437 &= 6191730672 && \text{WRONG!} \\ 144^5 &= 6191730672 \end{aligned}$$

It checks out.

What is a Proof?

How do you know that there aren't any smaller solutions?

ChatGPT said: Great question – and it gets right into the heart of computational number theory.
The short answer is: we don't know for sure that there are no smaller solutions. But here's why the solution

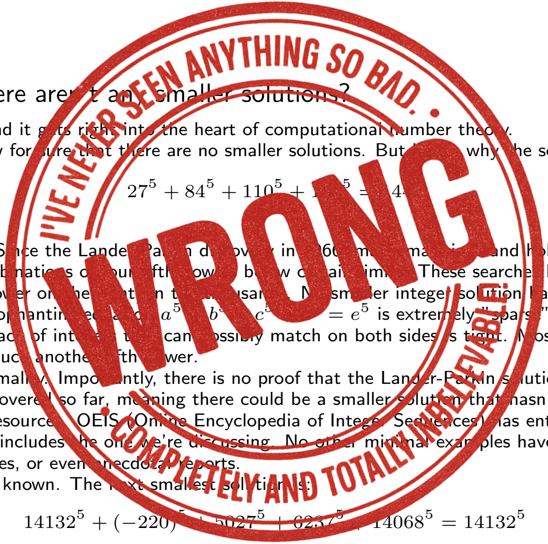
$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

is considered the smallest known:
1. Extensive Exhaustive Searches. Since the Lander-Parkin discovery in 1966, mathematicians and hobbyists have used computers to exhaustively search all possible combinations of four fifth powers below certain limits. These searches have tested billions of combinations, often up to values of e (the fifth power on the right) in the thousands. No smaller integer solution has been found despite this effort.
2. Nature of the Equation. The Diophantine equation $a^5 + b^5 + c^5 + d^5 = e^5$ is extremely "sparse" in terms of having solutions. Fifth powers grow very rapidly, so the space of integers that can possibly match on both sides is tight. Most fifth powers don't come close to summing nicely with others to produce another fifth power.
3. No Proof of Uniqueness or Minimality. Importantly, there is no proof that the Lander-Parkin solution is the smallest possible. It's just the smallest one that has been discovered so far, meaning there could be a smaller solution that hasn't yet been found.
4. Reference to OEIS and Other Resources. OEIS (Online Encyclopedia of Integer Sequences) has entries on this topic (like A023042) listing known solutions, but it only includes the one we're discussing. No other minimal examples have appeared in academic literature, high-powered computational searches, or even anecdotal reports.
As of now, no smaller solutions are known. The next smallest solution is:

$$14132^5 + (-220)^5 + 5027^5 + 6237^5 + 14068^5 = 14132^5$$

This was found by Lander, Parkin, and Selfridge in 1967.

What is a Proof?



How do you know that there aren't any smaller solutions?

ChatGPT said: Great question – and it gets right into the heart of computational number theory.
The short answer is: we don't know for sure that there are no smaller solutions. But here's why the solution

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

is considered the smallest known:
1. Extensive Exhaustive Searches. Since the Lander-Parkin discovery in 1966, mathematicians and hobbyists have used computers to exhaustively search all possible combinations of four fifth powers below certain limits. These searches have tested billions of combinations, often up to values of e (the fifth power of the integer) in the thousands. No smaller integer solution has been found despite this effort.
2. Nature of the Equation. The Diophantine equation $a^5 + b^5 + c^5 + d^5 = e^5$ is extremely "sparse" in terms of having solutions. Fifth powers grow very rapidly, so the space of integers that can possibly match on both sides is tight. Most fifth powers don't come close to summing nicely with others to produce another fifth power.
3. No Proof of Uniqueness or Minimality. Importantly, there is no proof that the Lander-Parkin solution is the smallest possible. It's just the smallest one that has been discovered so far, meaning there could be a smaller solution that hasn't yet been found.
4. Reference to OEIS and Other Resources. OEIS (Online Encyclopedia of Integer Sequences) has entries on this topic (like A023042) listing known solutions, but it only includes the one we're discussing. No other minimal examples have appeared in academic literature, high-powered computational searches, or even anecdotal reports.
As of now, no smaller solutions are known. The next smallest solution is:

$$14132^5 + (-220)^5 + 5027^5 + 6237^5 + 14068^5 = 14132^5$$

This was found by Lander, Parkin, and Selfridge in 1967.

What is a Proof?

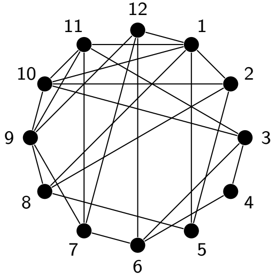
- Start with some facts about the problem, which we assume are true.
- At each step, derive a new fact from existing facts, in a way that is easily checkable if we agree with each previous fact.
- Finish by deriving whatever we want to show.

What is a Proof?

- Start with some facts about the problem, which we assume are true.
- At each step, derive a new fact from existing facts, in a way that is easily checkable if we agree with each previous fact.
- Finish by deriving whatever we want to show.
- For maximum clique:
  - Start with the definition of a maximum clique, and with the properties of our graph.
  - Need to show two things:
    - There is a solution with ω vertices.
    - There is no solution with more than ω vertices.

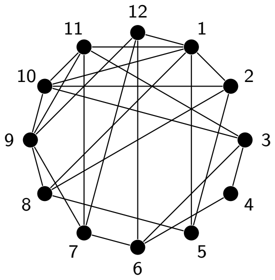
What Might an Ad-Hoc Proof Look Like?

```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



What Might an Ad-Hoc Proof Look Like?

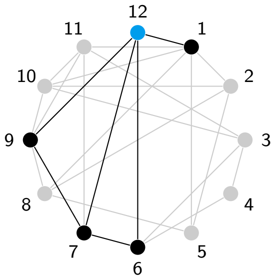
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Start with a header

What Might an Ad-Hoc Proof Look Like?

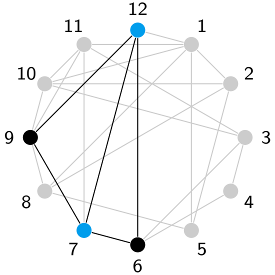
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Branch accepting 12
Throw away non-adjacent vertices

What Might an Ad-Hoc Proof Look Like?

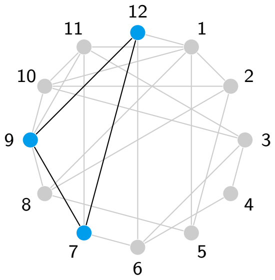
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Branch also accepting 7
Throw away non-adjacent vertices

What Might an Ad-Hoc Proof Look Like?

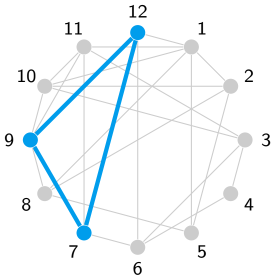
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Branch also accepting 9
Throw away non-adjacent vertices

What Might an Ad-Hoc Proof Look Like?

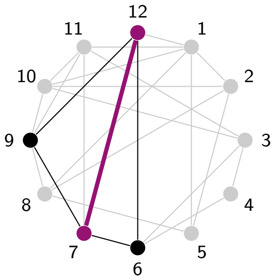
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



We branched on $12, 7, 9$
Found a new incumbent
Now looking for a ≥ 4 vertex clique

What Might an Ad-Hoc Proof Look Like?

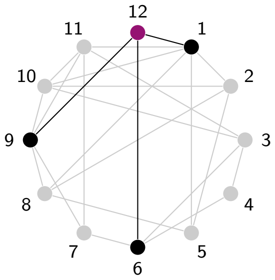
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Backtrack from 12, 7
9 explored already, only 6 feasible
No ≥ 4 vertex clique possible
Effectively this deletes the 7–12 edge

What Might an Ad-Hoc Proof Look Like?

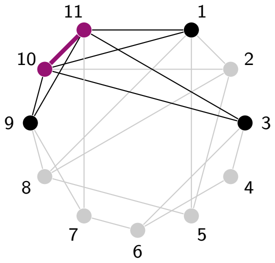
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Backtrack from 12
Only 1, 6 and 9 feasible (1-colourable)
No ≥ 4 vertex clique possible
Effectively this deletes vertex 12

What Might an Ad-Hoc Proof Look Like?

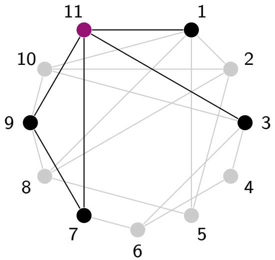
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



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

What Might an Ad-Hoc Proof Look Like?

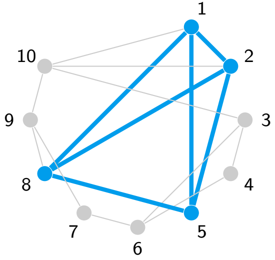
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Backtrack from 11
2-colourable, so no ≥ 4 clique
Delete the vertex

What Might an Ad-Hoc Proof Look Like?

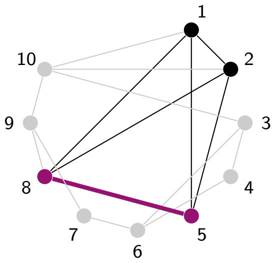
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



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

What Might an Ad-Hoc Proof Look Like?

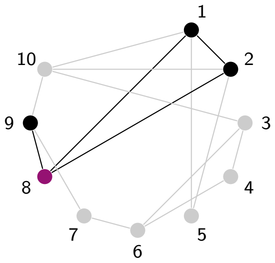
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



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

What Might an Ad-Hoc Proof Look Like?

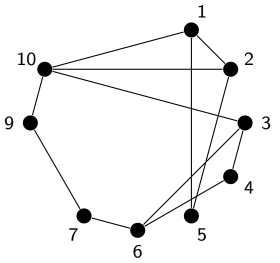
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Backtrack from 8
Still not enough vertices
Delete the vertex

What Might an Ad-Hoc Proof Look Like?

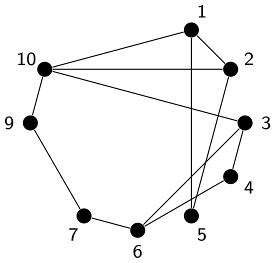
```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Remaining graph is 3-colourable
Backtrack from root node

What Might an Ad-Hoc Proof Look Like?

```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack
conclusion bounds 4 4
end maximum clique proof
```



Finish with what we've concluded
We specify a lower and an upper bound
Here they're the same, because we solved to optimality

Pseudo-Boolean Problems

I don't want to write a checker for an ad-hoc proof format, though, so let's try using VeriPB.

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

Except VeriPB works with pseudo-Boolean problems, not cliques...

Pseudo-Boolean Problems

- We have a set of variables x_i that must be given the value 0 (often means "false") or 1 ("true").
- A literal ℓ_i is a variable x_i or its negation $1 - x_i$, written as either $\tilde{}x_i$ or $\overline{x}_i$.
- Constraints are integer linear inequalities

$$\sum_i c_i \cdot \ell_i \geq A$$

  where c_i and A are integers.
- These are a superset of CNF, because

$$x \vee \overline{y} \vee z \qquad \leftrightarrow \qquad x + \overline{y} + z \geq 1$$

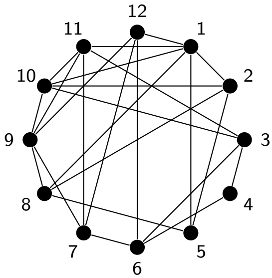
- We might have an objective to minimise,

$$\min \sum_i c_i \cdot \ell_i$$

Pseudo-Boolean Problems

Variables x_1 to x_{12}, $x_i = 1$ means "vertex i is in the clique".

```
min: 1 ~x1 1 ~x2 1 ~x3 1 ~x4 1 ~x5 1 ~x6 1 ~x7 1 ~x8 1 ~x9 1 ~x10 1 ~x11 1 ~x12 ;
@noedge1_3 -1 x3 -1 x1 >= -1 ;
@noedge1_4 -1 x4 -1 x1 >= -1 ;
@noedge1_6 -1 x6 -1 x1 >= -1 ;
@noedge1_7 -1 x7 -1 x1 >= -1 ;
@noedge1_9 -1 x9 -1 x1 >= -1 ;
@noedge2_3 -1 x3 -1 x2 >= -1 ;
@noedge2_4 -1 x4 -1 x2 >= -1 ;
@noedge2_6 -1 x6 -1 x2 >= -1 ;
@noedge2_7 -1 x7 -1 x2 >= -1 ;
@noedge2_9 -1 x9 -1 x2 >= -1 ;
@noedge2_11 -1 x11 -1 x2 >= -1 ;
@noedge2_12 -1 x12 -1 x2 >= -1 ;
* ...and a further 29 similar lines for the remaining non-edges
```



Pseudo-Boolean Problems

Variables x_1 to x_{12}, $x_i = 1$ means "vertex i is in the clique".

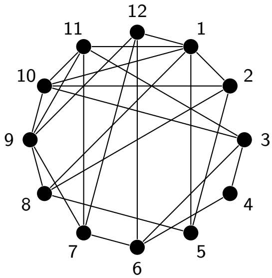
```
min: 1 ~x1 1 ~x2 1 ~x3 1 ~x4 1 ~x5 1 ~x6 1 ~x7 1 ~x8 1 ~x9 1 ~x10 1 ~x11 1 ~x12 ;
@noedge1_3 -1 x3 -1 x1 >= -1 ;
@noedge1_4 -1 x4 -1 x1 >= -1 ;
@noedge1_6 -1 x6 -1 x1 >= -1 ;
@noedge1_7 -1 x7 -1 x1 >= -1 ;
@noedge1_9 -1 x9 -1 x1 >= -1 ;
@noedge2_3 -1 x3 -1 x2 >= -1 ;
@noedge2_4 -1 x4 -1 x2 >= -1 ;
@noedge2_6 -1 x6 -1 x2 >= -1 ;
@noedge2_7 -1 x7 -1 x2 >= -1 ;
@noedge2_9 -1 x9 -1 x2 >= -1 ;
@noedge2_11 -1 x11 -1 x2 >= -1 ;
@noedge2_12 -1 x12 -1 x2 >= -1 ;
* ...and a further 29 similar lines for the remaining non-edges
```

Has to be a minimisation problem.

Multiplication and addition are implicit, so read this as

$$\min \sum_{i=1}^{12} 1 \cdot \overline{x}_i$$

i.e. minimise the number of vertices not selected.



Pseudo-Boolean Problems

Variables x_1 to x_{12}, $x_i = 1$ means "vertex i is in the clique".

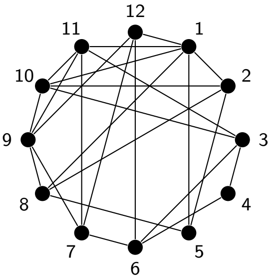
```
min: 1 ~x1 1 ~x2 1 ~x3 1 ~x4 1 ~x5 1 ~x6 1 ~x7 1 ~x8 1 ~x9 1 ~x10 1 ~x11 1 ~x12 ;
@noedge1_3 -1 x3 -1 x1 >= -1 ;
@noedge1_4 -1 x4 -1 x1 >= -1 ;
@noedge1_6 -1 x6 -1 x1 >= -1 ;
@noedge1_7 -1 x7 -1 x1 >= -1 ;
@noedge1_9 -1 x9 -1 x1 >= -1 ;
@noedge2_3 -1 x3 -1 x2 >= -1 ;
@noedge2_4 -1 x4 -1 x2 >= -1 ;
@noedge2_6 -1 x6 -1 x2 >= -1 ;
@noedge2_7 -1 x7 -1 x2 >= -1 ;
@noedge2_9 -1 x9 -1 x2 >= -1 ;
@noedge2_11 -1 x11 -1 x2 >= -1 ;
@noedge2_12 -1 x12 -1 x2 >= -1 ;
* ...and a further 29 similar lines for the remaining non-edges
```

For each non-edge, can't take both vertices. Note

$$-1 \cdot x_3 + -1 \cdot x1 \geq -1$$

is the same as

$$1 \cdot x_3 + 1 \cdot x_1 \leq 1$$

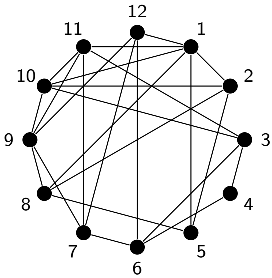


Pseudo-Boolean Problems

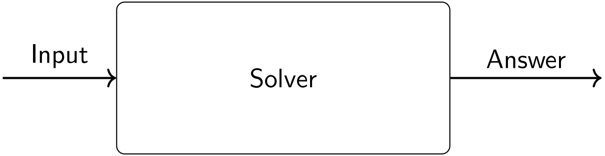
Variables x_1 to x_{12}, $x_i = 1$ means "vertex i is in the clique".

```
min: 1 ~x1 1 ~x2 1 ~x3 1 ~x4 1 ~x5 1 ~x6 1 ~x7 1 ~x8 1 ~x9 1 ~x10 1 ~x11 1 ~x12 ;
@noedge1_3 -1 x3 -1 x1 >= -1 ;
@noedge1_4 -1 x4 -1 x1 >= -1 ;
@noedge1_6 -1 x6 -1 x1 >= -1 ;
@noedge1_7 -1 x7 -1 x1 >= -1 ;
@noedge1_9 -1 x9 -1 x1 >= -1 ;
@noedge2_3 -1 x3 -1 x2 >= -1 ;
@noedge2_4 -1 x4 -1 x2 >= -1 ;
@noedge2_6 -1 x6 -1 x2 >= -1 ;
@noedge2_7 -1 x7 -1 x2 >= -1 ;
@noedge2_9 -1 x9 -1 x2 >= -1 ;
@noedge2_11 -1 x11 -1 x2 >= -1 ;
@noedge2_12 -1 x12 -1 x2 >= -1 ;
* ...and a further 29 similar lines for the remaining non-edges
```

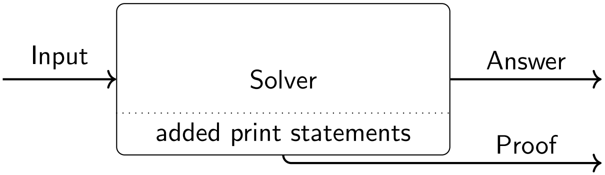
The @label is optional. It gives the constraint a name, which we'll use later on.



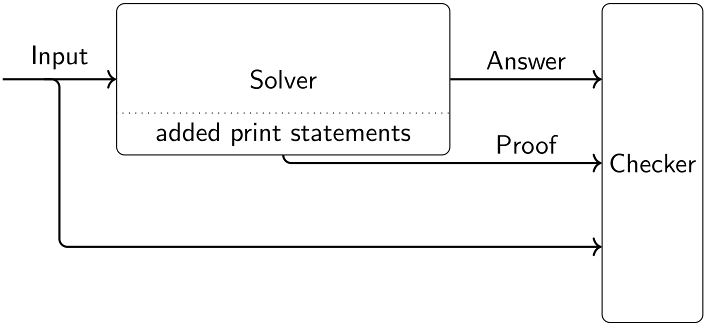
A Slightly Different Workflow



A Slightly Different Workflow

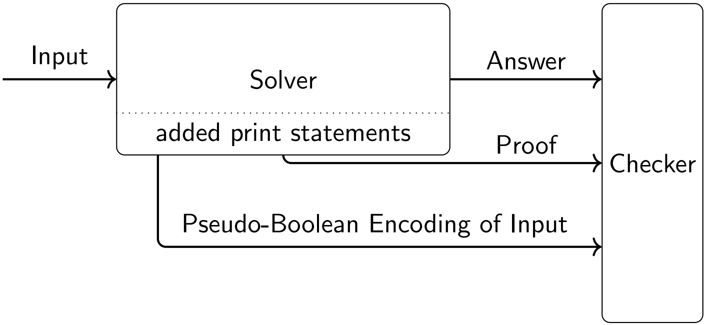


A Slightly Different Workflow

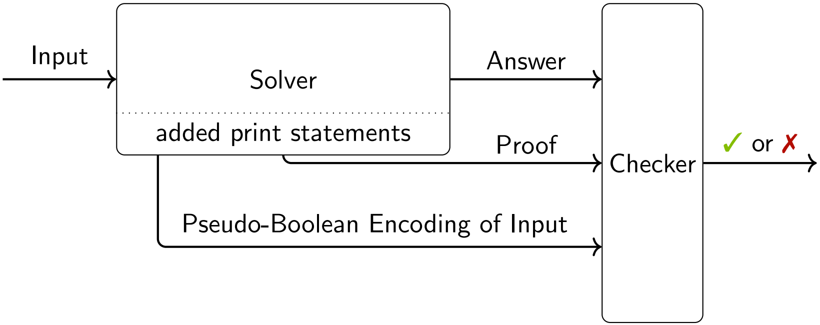


Gocht, McBride, McCreesh, Nordström, Prosser, Trimble: Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems, CP 2020

A Slightly Different Workflow



A Slightly Different Workflow



A VeriPB Proof, Attempt One

```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack

conclusion bounds 4 4
end maximum clique proof
```

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

Let's try directly translating our ad-hoc proof into VeriPB syntax.

A VeriPB Proof, Attempt One

maximum clique proof	pseudo-Boolean proof version 3.0
solution 7 9 12	@obj soli x7 x9 x12 ;
backtrack 12 7	rup 1 ~x12 1 ~x7 >= 1 ;
backtrack 12	rup 1 ~x12 >= 1 ;
backtrack 11 10	rup 1 ~x11 1 ~x10 >= 1 ;
backtrack 11	rup 1 ~x11 >= 1 ;
solution 1 2 5 8	@obj soli x1 x2 x5 x8 ;
backtrack 8 5	rup 1 ~x8 1 ~x5 >= 1 ;
backtrack 8	rup 1 ~x8 >= 1 ;
backtrack	rup >= 1 ;
	output NONE ;
conclusion bounds 4 4	conclusion BOUNDS 8 8 ;
end maximum clique proof	end pseudo-Boolean proof ;

We still start with a header.

A VeriPB Proof, Attempt One

`maximum clique proof`	`pseudo-Boolean proof version 3.0`
`solution 7 9 12`	`@obj soli x7 x9 x12 ;`
`backtrack 12 7`	`rup 1 ~x12 1 ~x7 >= 1 ;`
`backtrack 12`	`rup 1 ~x12 >= 1 ;`
`backtrack 11 10`	`rup 1 ~x11 1 ~x10 >= 1 ;`
`backtrack 11`	`rup 1 ~x11 >= 1 ;`
`solution 1 2 5 8`	`@obj soli x1 x2 x5 x8 ;`
`backtrack 8 5`	`rup 1 ~x8 1 ~x5 >= 1 ;`
`backtrack 8`	`rup 1 ~x8 >= 1 ;`
`backtrack`	`rup >= 1 ;`
	`output NONE ;`
`conclusion bounds 4 4`	`conclusion BOUNDS 8 8 ;`
`end maximum clique proof`	`end pseudo-Boolean proof ;`

The solution command is “soli”.

We put an “x” in front of vertex numbers, which are now Boolean variables.

The “@obj” is a label, which we'll use later.

A VeriPB Proof, Attempt One

`maximum clique proof`	`pseudo-Boolean proof version 3.0`
`solution 7 9 12`	`@obj soli x7 x9 x12 ;`
`backtrack 12 7`	`rup 1 ~x12 1 ~x7 >= 1 ;`
`backtrack 12`	`rup 1 ~x12 >= 1 ;`
`backtrack 11 10`	`rup 1 ~x11 1 ~x10 >= 1 ;`
`backtrack 11`	`rup 1 ~x11 >= 1 ;`
`solution 1 2 5 8`	`@obj soli x1 x2 x5 x8 ;`
`backtrack 8 5`	`rup 1 ~x8 1 ~x5 >= 1 ;`
`backtrack 8`	`rup 1 ~x8 >= 1 ;`
`backtrack`	`rup >= 1 ;`
	`output NONE ;`
`conclusion bounds 4 4`	`conclusion BOUNDS 8 8 ;`
`end maximum clique proof`	`end pseudo-Boolean proof ;`

To backtrack, we use "rup".

We're saying "at least one of these variables must be false", i.e. at least one of these vertices must not be selected if we're to find a larger clique.

A VeriPB Proof, Attempt One

`maximum clique proof`	`pseudo-Boolean proof version 3.0`
`solution 7 9 12`	`@obj soli x7 x9 x12 ;`
`backtrack 12 7`	`rup 1 ~x12 1 ~x7 >= 1 ;`
`backtrack 12`	`rup 1 ~x12 >= 1 ;`
`backtrack 11 10`	`rup 1 ~x11 1 ~x10 >= 1 ;`
`backtrack 11`	`rup 1 ~x11 >= 1 ;`
`solution 1 2 5 8`	`@obj soli x1 x2 x5 x8 ;`
`backtrack 8 5`	`rup 1 ~x8 1 ~x5 >= 1 ;`
`backtrack 8`	`rup 1 ~x8 >= 1 ;`
`backtrack`	`rup >= 1 ;`
	`output NONE ;`
`conclusion bounds 4 4`	`conclusion BOUNDS 8 8 ;`
`end maximum clique proof`	`end pseudo-Boolean proof ;`

Same idea.

A VeriPB Proof, Attempt One

```
maximum clique proof
solution 7 9 12
backtrack 12 7
backtrack 12
backtrack 11 10
backtrack 11
solution 1 2 5 8
backtrack 8 5
backtrack 8
backtrack

conclusion bounds 4 4
end maximum clique proof
```

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

Backtracking from the root note is saying “at least one of the variables from this empty sum must be true”, i.e. asserting contradiction.

A VeriPB Proof, Attempt One

`maximum clique proof`	`pseudo-Boolean proof version 3.0`
`solution 7 9 12`	`@obj soli x7 x9 x12 ;`
`backtrack 12 7`	`rup 1 ~x12 1 ~x7 >= 1 ;`
`backtrack 12`	`rup 1 ~x12 >= 1 ;`
`backtrack 11 10`	`rup 1 ~x11 1 ~x10 >= 1 ;`
`backtrack 11`	`rup 1 ~x11 >= 1 ;`
`solution 1 2 5 8`	`@obj soli x1 x2 x5 x8 ;`
`backtrack 8 5`	`rup 1 ~x8 1 ~x5 >= 1 ;`
`backtrack 8`	`rup 1 ~x8 >= 1 ;`
`backtrack`	`rup >= 1 ;`
	`output NONE ;`
`conclusion bounds 4 4`	`conclusion BOUNDS 8 8 ;`
`end maximum clique proof`	`end pseudo-Boolean proof ;`

The “output” rule is for advanced features which we're not using, so we have no output.

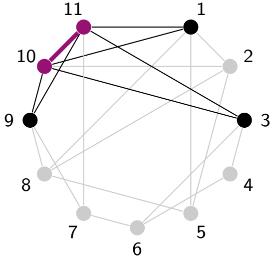
Recall we're minimising the number of unselected vertices, so the bound is $12 - 4 = 8$.

Verifying This Proof (Or Not...)

```
$ pboxide_veripb example.opb example-1.pbp
Running PBOxide VeriPB version 0.2.0-357263b
Error: Verification error at example-1.pbp:6!

Caused by:
        The constraint is not implied by reverse unit propagation (RUP).
```

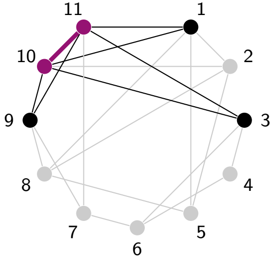
Verifying This Proof (Or Not. . .)

```
$ pboxide_veripb example.opb example-1.pbp
Running PBOxide VeriPB version 0.2.0-357263b
Error: Verification error at example-1.pbp:6!

Caused by:
        The constraint is not implied by reverse unit propagation (RUP).
```



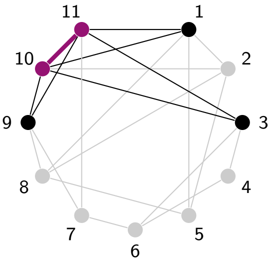
Verifying This Proof (Or Not. . .)

```
$ pboxide_veripb --trace example.opb example-1.pbp
Running PBOxide VeriPB version 0.2.0-357263b
  Objective: min 1 ~x1 1 ~x2 1 ~x3 1 ~x4 1 ~x5 1 ~x6 1 ~x7 1 ~x8 1 ~x9 1 ~x10 1 ~x11 1 ~x12 + 0 ;
  ConstraintId 1: 1 ~x1 1 ~x3 >= 1
  ConstraintId 2: 1 ~x2 1 ~x3 >= 1
  ...
line    2: @obj soli x12 x7 x9 ;
  ConstraintID 42: 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    3: rup 1 ~x12 1 ~x7 1 ~x9 >= 1 ;
  ConstraintID 43: 1 ~x7 1 ~x9 1 ~x12 >= 1
  ...
line    6: rup 1 ~x11 1 ~x10 >= 1 ;
Error: Verification error at example-1.pbp:6!

Caused by:
        The constraint is not implied by reverse unit propagation (RUP).
```



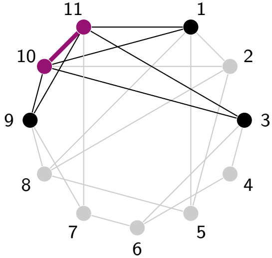
Verifying This Proof (Or Not. . .)

```
$ pboxide_veripb --trace-failed example.opb example-1.pbp
  ...
line    6: rup 1 ~x11 1 ~x10 >= 1 ;
Propagation check failed! The propagation had the following trail:
  propagatons in format: <assignment> (<reason constraint>)
    ~x12 (1 ~x12 >= 1)
    x10 (1 x10 1 x11 >= 2)
    x11 (1 x10 1 x11 >= 2)
    ~x6 (1 ~x6 1 ~x10 >= 1)
    ~x7 (1 ~x7 1 ~x10 >= 1)
    ~x8 (1 ~x8 1 ~x10 >= 1)
    ~x4 (1 ~x4 1 ~x10 >= 1)
    ~x5 (1 ~x5 1 ~x10 >= 1)
    ~x2 (1 ~x2 1 ~x11 >= 1)
Error: Verification error at example-1.pbp:6!

Caused by:
        The constraint is not implied by reverse unit propagation (RUP).
```



Verifying This Proof (Or Not. . .)

- The proof checker isn't smart enough to figure out that:
  - The vertices 1, 3, and 9 can be coloured using one colour. . .
  - And each colour class contributes at most one to the objective variable. . .
  - So the "find a clique with more than three vertices" solution-improving constraint can't be satisfied if we have accepted both 10 and 11.



Reasoning With Colour Classes

If we could take the "objective improving" constraint

```
line    2: @obj soli x12 x7 x9 ;
  ConstraintID 42: 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
```

i.e.

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} \geq 4$$

Reasoning With Colour Classes

If we could take the "objective improving" constraint

```
line    2: @obj soli x12 x7 x9 ;
  ConstraintID 42: 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
```

i.e.

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} \geq 4$$

and add an at-most-one constraint over the colour class $\{x_1, x_3, x_9\}$,

$$\overline{x}_1 + \overline{x}_3 + \overline{x}_9 \geq 2$$

Reasoning With Colour Classes

If we could take the "objective improving" constraint

```
line    2: @obj soli x12 x7 x9 ;
  ConstraintID 42: 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
```

i.e.

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} \geq 4$$

and add an at-most-one constraint over the colour class $\{x_1, x_3, x_9\}$,

$$\overline{x}_1 + \overline{x}_3 + \overline{x}_9 \geq 2$$

we would get

$$(x_1 + \overline{x}_1) + x_2 + (x_3 + \overline{x}_3) + x_4 + x_5 + x_6 + x_7 + x_8 + (x_9 + \overline{x}_9) + x_{10} + x_{11} + x_{12} \geq 6$$

Reasoning With Colour Classes

If we could take the "objective improving" constraint

```
line    2: @obj soli x12 x7 x9 ;
  ConstraintID 42: 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
```

i.e.

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} \geq 4$$

and add an at-most-one constraint over the colour class $\{x_1, x_3, x_9\}$,

$$\overline{x}_1 + \overline{x}_3 + \overline{x}_9 \geq 2$$

we would get

$$(x_1 + \overline{x}_1) + x_2 + (x_3 + \overline{x}_3) + x_4 + x_5 + x_6 + x_7 + x_8 + (x_9 + \overline{x}_9) + x_{10} + x_{11} + x_{12} \geq 6$$

and simplifying using $x_i + \overline{x}_i = 1$ we get

$$x_2 + x_4 + x_5 + x_6 + x_7 + x_8 + x_{10} + x_{11} + x_{12} \geq 3$$

from which it is much easier to see that taking both vertices 10 and 11 isn't going to work.

Reasoning With Colour Classes

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
a 1 x2 1 x4 1 x5 1 x6 1 x7 1 x8 1 x10 1 x11 1 x12 >= 3 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
a 1 x8 1 x11 1 x12 >= 2 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

The "a" means "I'm asserting this without a justification."

It turns out we'll need to help here too. This time we're dealing with three colour classes, each of three vertices. Note that this isn't obvious from the constraint we're asserting...

Reasoning With Colour Classes

```
$ pboxide_veripb example.opb example-2.pbp
Running PBOxide VeriPB version 0.2.0-357263b
s VERIFIED BOUNDS 8 <= obj <= 8
Warning: The proof used unchecked assumptions.
```

This passes, but the checker complains about our use of the “a” rule.

This is fair: it’s really not obvious that the constraints we specify are valid. In general, we might have worked very hard to produce a good colour bound.

Reasoning With Colour Classes

```
pseudo-Boolean proof version 3.0 ;
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
a 1 ~x1 1 ~x3 1 ~x9 >= 2 ;
pol @obj -1 + ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
a 1 ~x1 1 ~x3 1 ~x7 >= 2 ;
a 1 ~x2 1 ~x4 1 ~x9 >= 2 ;
a 1 ~x5 1 ~x6 1 ~x10 >= 2 ;
pol @obj -1 + -2 + -3 + ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

"pol" means "reverse Polish notation". So, take the solution-improving constraint we labelled "@obj", and add the previous constraint (negative numbers are relative to the current constraint).

Here we're asserting at-most-one constraints for three colour classes, and then adding all of them to the solution-improving constraint.

Reasoning With Colour Classes

```
$ pboxide_veripb example.opb example-3.pbp
Running PBOxide VeriPB version 0.2.0-357263b
s VERIFIED BOUNDS 8 <= obj <= 8
Warning: The proof used unchecked assumptions.
```

Still relying upon assertions, but “these vertices form a colour class” is easily verifiable.

Gocht, McBride, McCreesh, Nordström, Prosser, Trimble: Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems, CP 2020

Recovering At-Most-One Constraints

We can lazily recover at-most-one constraints for each colour class!

Recovering At-Most-One Constraints

We can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3
\end{aligned}
$$

Recovering At-Most-One Constraints

We can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
&/\,2 && = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq \frac{3}{2}
\end{aligned}
$$

Recovering At-Most-One Constraints

We can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
&/\,2 && = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2
\end{aligned}
$$

Recovering At-Most-One Constraints

We can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
& (\overline{x}_1 + \overline{x}_6 \geq 1) & & \\
+ & (\overline{x}_1 + \overline{x}_9 \geq 1) & & = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+ & (\overline{x}_6 + \overline{x}_9 \geq 1) & & = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
& / \, 2 & & = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2
\end{aligned}
$$

i.e. $x_1 + x_6 + x_9 \leq 1$

Recovering At-Most-One Constraints

We can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
&/\,2 && = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2
\end{aligned}
$$

i.e. $x_1 + x_6 + x_9 \leq 1$

This generalises to colour classes of any size v.

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

Solvers don't need to "understand" why this works (or when exactly it is necessary) to write this derivation to proof log.

Recovering At-Most-One Constraints

```
pseudo-Boolean proof version 3.0
@obj soli x12 x7 x9 ;
rup 1 ~x12 1 ~x7 1 ~x9 >= 1 ;
rup 1 ~x12 1 ~x7 >= 1 ;
pol @noedge1_6 @noedge1_9 + @noedge6_9 + 2 d @obj + ;
rup 1 ~x12 >= 1 ;
pol @noedge1_3 @noedge1_9 + @noedge3_9 + 2 d @obj + ;
rup 1 ~x11 1 ~x10 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @obj + ;
rup 1 ~x11 >= 1 ;
@obj soli x8 x5 x2 x1 ;
rup 1 ~x8 1 ~x5 >= 1 ;
pol @obj @noedge1_9 + ;
rup 1 ~x8 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @noedge2_4 @noedge2_9 + @noedge4_9 + 2 d +
    @noedge5_6 @noedge5_10 + @noedge6_10 + 2 d + @obj + ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

It turns out we do colour-class reasoning in each of these three places, although the verifier doesn't actually need help for two of them.

In each case, we add together three non-adjacency constraints, divide by two, and then add the result to the solution-improving constraint.

Here we're creating three at-most-one constraints, and then adding all of them to the solution-improving constraint.

Recovering At-Most-One Constraints

```
$ pboxide_veripb example.opb example-4.pbp
Running PBOxide VeriPB version 0.2.0-357263b
s VERIFIED BOUNDS 8 <= obj <= 8
```

Why a General-Purpose Proof System?

Why this, rather than having a “colouring bound” rule directly in the proof format?

- Dozens of different colouring algorithms, and extensions using MaxSAT-like reasoning.
- Even more when we look at problems like maximum weight clique. Weighted colour class rules can be extremely clever (e.g. vertices can split their weights between multiple colours).
- Don’t really want to have a proof checker and format per solver if we can help it. . .

Gocht: Certifying Correctness for Combinatorial Algorithms: by Using Pseudo-Boolean Reasoning. PhD Thesis, Lund University (2022)

What is a VeriPB Proof?

For unsatisfiable problem instances:

- Start by assuming the pseudo-Boolean constraints in the input.
- At each step, derive a new additional constraint that must hold, based upon what we know so far.
  - "Must hold" means "equisatisfiable": can't turn a satisfiable instance into an unsatisfiable instance, or vice-versa.
  - Steps have to be efficiently computable, possibly with hints.
- Finish by deriving $0 \geq 1$.

Gocht: Certifying Correctness for Combinatorial Algorithms: by Using Pseudo-Boolean Reasoning. PhD Thesis, Lund University (2022)

What is a VeriPB Proof?

For unsatisfiable problem instances:

- Start by assuming the pseudo-Boolean constraints in the input.
- At each step, derive a new additional constraint that must hold, based upon what we know so far.
  - "Must hold" means "equisatisfiable": can't turn a satisfiable instance into an unsatisfiable instance, or vice-versa.
  - Steps have to be efficiently computable, possibly with hints.
- Finish by deriving $0 \geq 1$.

For optimisation:

- Can give witnesses of solutions, which become a solution-improving constraint.
- So we prove "unsatisfiable, if you want something better than a solution I told you was best".
- Now "equisatisfiable" becomes "equioptimal".

Gocht: Certifying Correctness for Combinatorial Algorithms: by Using Pseudo-Boolean Reasoning. PhD Thesis, Lund University (2022)

Cutting Planes Proofs 1: Linear Inequalities

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 $c \in \mathbb{N}^+$

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

Cutting Planes Proofs 2: 0-1 Variables

Literal axioms

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

Division

for any $c \in \mathbb{N}^+$
assumes normalised form
with $a_i, A \in \mathbb{N}$

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

Saturation

assumes normalised form

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \min(a_i, A)\, \ell_i \geq A}$$

Unit Propagation for Clauses

Given the following,

$$a \vee \overline{b} \vee c$$
$$\overline{c} \vee d$$

Suppose I tell you that a must be false and b must be true.

Gocht: Certifying Correctness for Combinatorial Algorithms: by Using Pseudo-Boolean Reasoning. PhD Thesis, Lund University (2022)

Unit Propagation for Clauses

Given the following,

$$a \vee \overline{b} \vee c$$
$$\overline{c} \vee d$$

Suppose I tell you that a must be false and b must be true.

Must set c to true to satisfy first clause.

Unit Propagation for Clauses

Given the following,

$$a \vee \overline{b} \vee c$$
$$\overline{c} \vee d$$

Suppose I tell you that a must be false and b must be true.

Must set c to true to satisfy first clause.

Now must set d to false to satisfy second clause.

Unit Propagation for Pseudo-Boolean Constraints

$$5a + 2b + 3c + d + e + f \geq 5$$

Suppose I tell you that $a = 0$. Can't say anything yet.

Unit Propagation for Pseudo-Boolean Constraints

$$5a + 2b + 3c + d + e + f \geq 5$$

Suppose I tell you that $a = 0$. Can't say anything yet.

Suppose I tell you that $b = 0$ as well. Then $c = 1$ must hold.

Unit Propagation for Pseudo-Boolean Constraints

$$5a + 2b + 3c + d + e + f \geq 5$$

Suppose I tell you that $a = 0$. Can't say anything yet.

Suppose I tell you that $b = 0$ as well. Then $c = 1$ must hold.

Suppose I tell you that $d = 0$ as well. Then $e = 1$ and $f = 1$ must hold.

Unit Propagation for Pseudo-Boolean Constraints

$$5a + 2b + 3c + d + e + f \geq 5$$

Suppose I tell you that $a = 0$. Can't say anything yet.

Suppose I tell you that $b = 0$ as well. Then $c = 1$ must hold.

Suppose I tell you that $d = 0$ as well. Then $e = 1$ and $f = 1$ must hold.

In general: integer bounds consistency. We can do this *efficiently*, but it's not quite as simple as for clauses.

Reverse Unit Propagation

Let C be any constraint. Suppose $\overline{C}$ unit propagates to contradiction. Then without loss of satisfaction or optimality, we can add C as a new constraint.

RUP
C any constraint

$$\frac{\overline{C} \text{ unit propagates to contradiction}}{C}$$

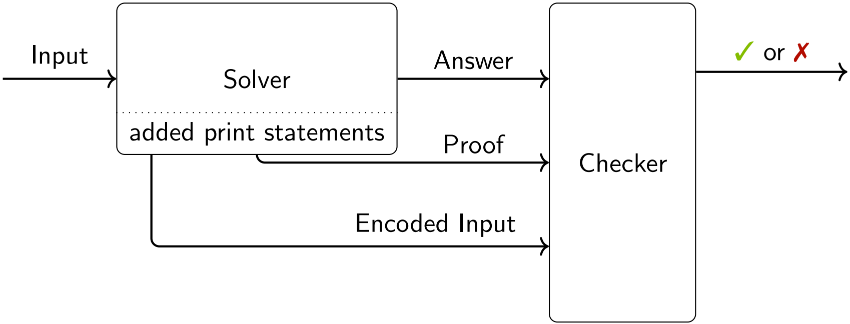
Interleaving RUP and Other Inferences

- RUP forms a good skeleton for DPLL (or CDCL) style searches.
- However, many facts discovered by smarter algorithms do *not* follow by RUP.
- Key idea: can interleave RUP with additional cutting planes steps to help out.

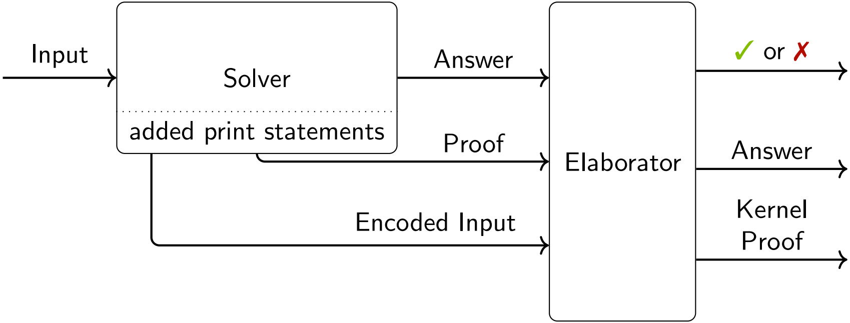
Not Appearing in this Talk

- Extension variables (redundance, dominance, and symmetries).
- Deletions.

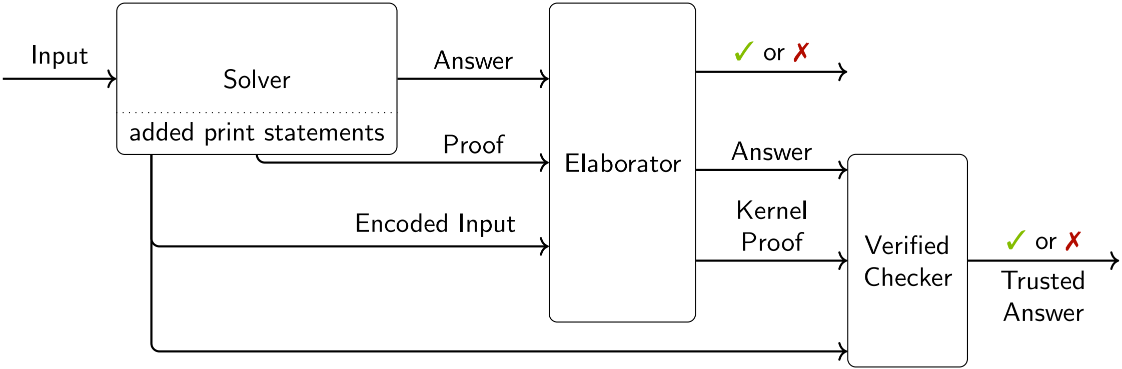
Reducing the Trust Base



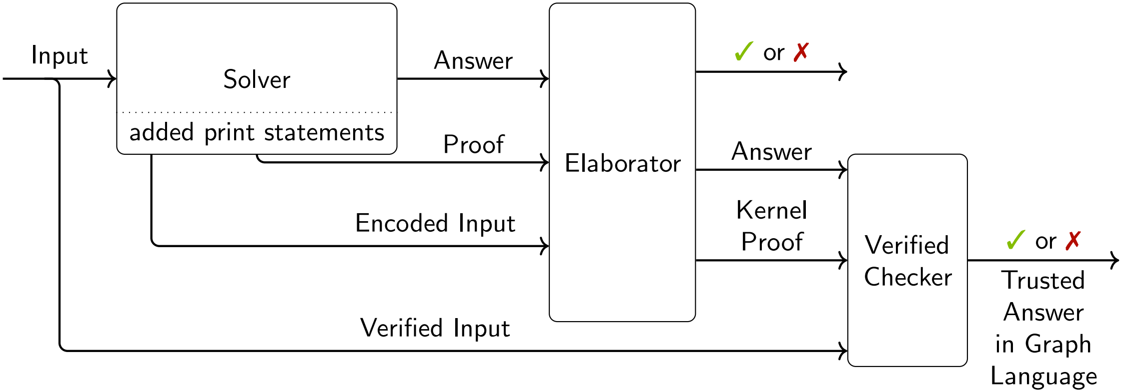
Reducing the Trust Base



Reducing the Trust Base



Reducing the Trust Base



End-to-End Verification for Maximum Clique

```
$ glasgow_clique_solver brock200_4.clq
recursions = 56613
clique = 17 (12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192)
runtime = 0.02153s

$ glasgow_clique_solver brock200_4.clq --prove brock200_4
runtime = 0.266475s

$ cake_pb_clique brock200_4.clq > brock200_4.verifiedopb

$ time pboxide_veripb --elaborate brock200_4.corepb brock200_4.verifiedopb brock200_4.pbp
s VERIFIED BOUNDS 183 <= obj <= 183
real    0m2.139s

$ time cake_pb_clique brock200_4.clq brock200_4.corepb
s VERIFIED MAX CLIQUE SIZE |CLIQUE| = 17
real    0m2.260s
```

What Exactly are we Verifying?

$$\begin{aligned}
&\textsf{is_clique}\ vs\ (v,e) \stackrel{\text{def}}{=} \\
&\quad vs \subseteq \{\ 0,1,\ldots,v-1\ \} \ \wedge \\
&\quad \forall\, x\ y.\ x \in vs \wedge y \in vs \wedge x \neq y \Rightarrow \textsf{is_edge}\ e\ x\ y \\
&\textsf{max_clique_size}\ g \stackrel{\text{def}}{=} \textsf{max}_{\textsf{set}}\ \{\ \textsf{card}\ vs \mid \textsf{is_clique}\ vs\ g\ \}
\end{aligned}$$

What Exactly are we Verifying?

$$
\begin{aligned}
&\text{clique\_eq\_str}\ n \stackrel{\text{def}}{=} \texttt{"s VERIFIED MAX CLIQUE SIZE |CLIQUE| = "}\ \hat{}\ \text{toString}\ n\ \hat{}\ \texttt{"\textbackslash n"} \\
&\text{clique\_bound\_str}\ l\ u \stackrel{\text{def}}{=} \\
&\quad \texttt{"s VERIFIED MAX CLIQUE SIZE BOUND "}\ \hat{}\ \text{toString}\ l\ \hat{}\ \texttt{" <= |CLIQUE| <= "}\ \hat{}\ \text{toString}\ u\ \hat{}\ \texttt{"\textbackslash n"} \\
&\vdash \text{cake\_pb\_clique\_run}\ cl\ fs\ mc\ ms \Rightarrow \\
&\qquad \text{machine\_sem}\ mc\ (\text{basis\_ffi}\ cl\ fs)\ ms \subseteq \\
&\qquad\quad \text{extend\_with\_resource\_limit}\ \{\ \text{Terminate Success}\ (\text{cake\_pb\_clique\_io\_events}\ cl\ fs)\ \}\ \wedge \\
&\qquad \exists\, out\ err. \\
&\qquad\quad \text{extract\_fs}\ fs\ (\text{cake\_pb\_clique\_io\_events}\ cl\ fs) = \text{Some}\ (\text{add\_stdout}\ (\text{add\_stderr}\ fs\ err)\ out)\ \wedge \\
&\qquad\quad (out \neq \texttt{""} \Rightarrow \\
&\qquad\qquad \exists\, g.\ \text{get\_graph\_dimacs}\ fs\ (\text{el}\ 1\ cl) = \text{Some}\ g\ \wedge \\
&\qquad\qquad\quad (\text{length}\ cl = 2 \wedge out = \text{concat}\ (\text{print\_pbf}\ (\text{full\_encode}\ g))\ \vee \\
&\qquad\qquad\quad\ \text{length}\ cl = 3\ \wedge \\
&\qquad\qquad\qquad (out = \text{clique\_eq\_str}\ (\text{max\_clique\_size}\ g)\ \vee \\
&\qquad\qquad\qquad \exists\, l\ u. out = \text{clique\_bound\_str}\ l\ u \wedge (\forall\, vs.\ \text{is\_clique}\ vs\ g \Rightarrow \text{card}\ vs \leq u)\ \wedge \\
&\qquad\qquad\qquad\qquad \exists\, vs.\ \text{is\_clique}\ vs\ g \wedge l \leq \text{card}\ vs)))
\end{aligned}
$$

What's Left to Trust?

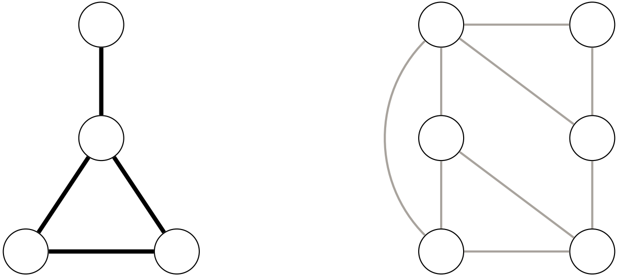
Still have to trust:

- The HOL4 theorem prover.
- That the formal HOL model of the CakeML environment corresponds to the hardware on which it is run.
- HOL definition of what it means to be a maximum clique.
- Input parsing and output formatting.

No need to trust, or even know about:

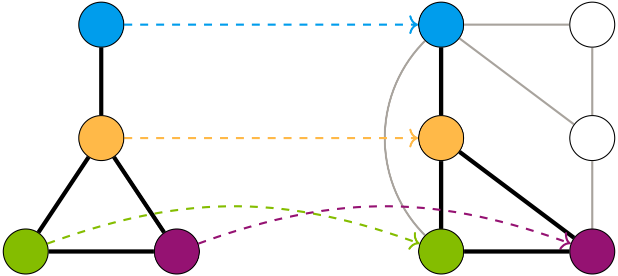
- How the solver works.
- What pseudo-Boolean means.

Subgraph Isomorphism



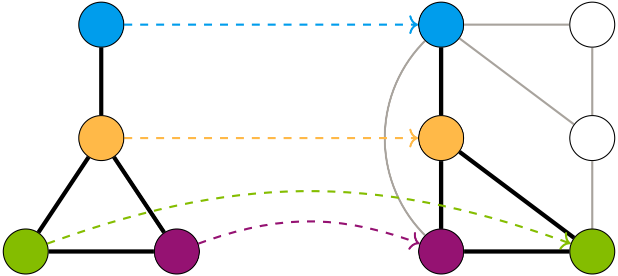
- Find the pattern inside the target.
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches.

Subgraph Isomorphism



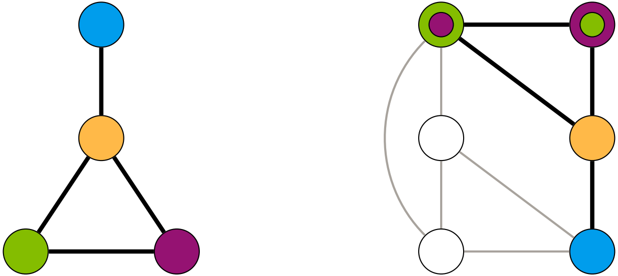
- Find the pattern inside the target.
- Applications in compilers, biochemistry, model checking, pattern recognition, . . .
- Often want to find all matches.

Subgraph Isomorphism



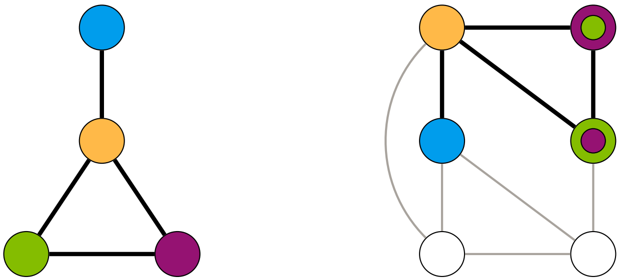
- Find the pattern inside the target.
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches.

Subgraph Isomorphism



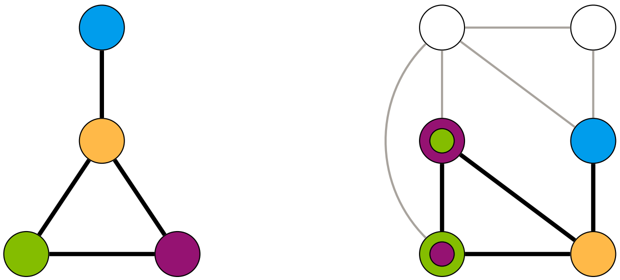
- Find the pattern inside the target.
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches.

Subgraph Isomorphism



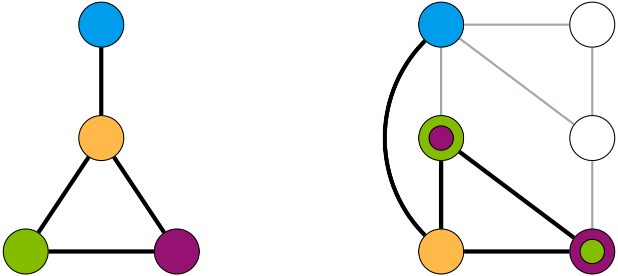
- Find the pattern inside the target.
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches.

Subgraph Isomorphism



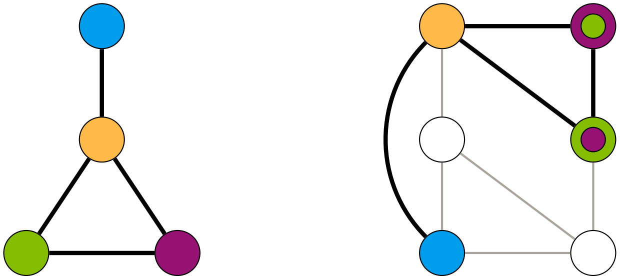
- Find the pattern inside the target.
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches.

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.

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a 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 gets a target vertex:

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

Each target vertex may be used at most once:

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

Subgraph Isomorphism in Pseudo-Boolean Form

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:

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

Injectivity (All-Different) Reasoning

$$
\begin{array}{rl}
p \in & \{\, 1 \quad \quad \quad 4 \quad 5 \,\} \\
q \in & \{\, 1 \quad 2 \quad 3 \quad \quad \,\} \\
r \in & \{\, \quad 2 \quad 3 \quad \quad \,\} \\
s \in & \{\, 1 \quad \quad 3 \quad \quad \,\} \\
t \in & \{\, 1 \quad \quad 3 \quad \quad \,\}
\end{array}
$$

Injectivity (All-Different) Reasoning

$$
\begin{array}{llllllll}
p \in \{ & 1 & & & 4 & 5 & \} \\
q \in \{ & 1 & 2 & 3 & & & \} \\
r \in \{ & & 2 & 3 & & & \} \\
s \in \{ & 1 & & 3 & & & \} \\
t \in \{ & 1 & & 3 & & & \}
\end{array}
$$

ChatGPT, can you draw me a picture of a pigeon wearing a disguise?

Injectivity (All-Different) Reasoning

$$\begin{array}{l} p \in \{1 \quad \quad 4 \quad 5\} \\ q \in \{1 \quad 2 \quad 3 \quad \quad \} \\ r \in \{\quad 2 \quad 3 \quad \quad \} \\ s \in \{1 \quad \quad 3 \quad \quad \} \\ t \in \{1 \quad \quad 3 \quad \quad \} \end{array}$$

$$x_{q,1} + x_{q,2} + x_{q,3} \geq 1$$ vertex q must be mapped

Injectivity (All-Different) Reasoning

$$
\begin{array}{llllll}
p \in \{1 \quad\quad 4 \quad 5\} & & & & & \\
q \in \{1 \; 2 \; 3 \quad\quad\} & x_{q,1} + & x_{q,2} + & x_{q,3} & \geq 1 & \text{vertex } q \text{ must be mapped} \\
r \in \{\quad 2 \; 3 \quad\quad\} & & x_{r,2} + & x_{r,3} & \geq 1 & \text{vertex } r \text{ must be mapped} \\
s \in \{1 \quad 3 \quad\quad\} & x_{s,1} & + & x_{s,3} & \geq 1 & \text{vertex } s \text{ must be mapped} \\
t \in \{1 \quad 3 \quad\quad\} & x_{t,1} & + & x_{t,3} & \geq 1 & \text{vertex } t \text{ must be mapped}
\end{array}
$$

Injectivity (All-Different) Reasoning

$$
\begin{array}{llllllll}
p \in \{1 \quad\quad 4 \quad 5\} & & & & & & & \\
q \in \{1 \quad 2 \quad 3 \quad\quad\} & x_{q,1} + & x_{q,2} + & x_{q,3} & & & \geq 1 & \text{vertex } q \text{ must be mapped} \\
r \in \{\quad 2 \quad 3 \quad\quad\} & & x_{r,2} + & x_{r,3} & & & \geq 1 & \text{vertex } r \text{ must be mapped} \\
s \in \{1 \quad\quad 3 \quad\quad\} & x_{s,1} & + & x_{s,3} & & & \geq 1 & \text{vertex } s \text{ must be mapped} \\
t \in \{1 \quad\quad 3 \quad\quad\} & x_{t,1} & + & x_{t,3} & & & \geq 1 & \text{vertex } t \text{ must be mapped}
\end{array}
$$

$$
\begin{array}{lll}
\rightarrow \quad -x_{p,1} + -x_{q,1} + -x_{s,1} + -x_{t,1} & \geq -1 & \text{use vertex 1 at most once} \\
\rightarrow \quad -x_{q,2} + -x_{r,2} & \geq -1 & \text{use vertex 2 at most once} \\
\rightarrow \quad -x_{q,3} + -x_{r,3} + -x_{s,3} + -x_{t,3} & \geq -1 & \text{use vertex 3 at most once}
\end{array}
$$

Injectivity (All-Different) Reasoning

$$
\begin{array}{llllllllll}
p \in \{1 \quad\quad 4 \quad 5\} & & & & & & & & & \\
q \in \{1 \; 2 \; 3 \quad\quad\} & x_{q,1} & + & x_{q,2} & + & x_{q,3} & & & \geq & 1 & \text{vertex } q \text{ must be mapped} \\
r \in \{\quad 2 \; 3 \quad\quad\} & & & x_{r,2} & + & x_{r,3} & & & \geq & 1 & \text{vertex } r \text{ must be mapped} \\
s \in \{1 \quad 3 \quad\quad\} & x_{s,1} & & & + & x_{s,3} & & & \geq & 1 & \text{vertex } s \text{ must be mapped} \\
t \in \{1 \quad 3 \quad\quad\} & x_{t,1} & & & + & x_{t,3} & & & \geq & 1 & \text{vertex } t \text{ must be mapped}
\end{array}
$$

$$
\begin{array}{lllllllll}
\rightarrow & -x_{p,1} & + & -x_{q,1} & + & -x_{s,1} & + & -x_{t,1} & \geq -1 & \text{use vertex 1 at most once} \\
\rightarrow & & & -x_{q,2} & + & -x_{r,2} & & & \geq -1 & \text{use vertex 2 at most once} \\
\rightarrow & & & -x_{q,3} & + & -x_{r,3} + -x_{s,3} & + & -x_{t,3} & \geq -1 & \text{use vertex 3 at most once}
\end{array}
$$

$$
-x_{p,1} \geq 1 \qquad \text{sum all of the above}
$$

Elffers, Gocht, McCreesh, Nordström: Justifying All Differences Using Pseudo-Boolean Reasoning, AAAI 2020

Injectivity (All-Different) Reasoning

$$
\begin{array}{l}
p \in \{\, 1 \quad\quad 4 \quad 5 \,\} \\
q \in \{\, 1 \quad 2 \quad 3 \quad\quad \,\} \\
r \in \{\, \quad 2 \quad 3 \quad\quad \,\} \\
s \in \{\, 1 \quad\quad 3 \quad\quad \,\} \\
t \in \{\, 1 \quad\quad 3 \quad\quad \,\}
\end{array}
$$

$x_{q,1} + x_{q,2} + x_{q,3} \geq 1$	vertex q must be mapped
$x_{r,2} + x_{r,3} \geq 1$	vertex r must be mapped
$x_{s,1} + x_{s,3} \geq 1$	vertex s must be mapped
$x_{t,1} + x_{t,3} \geq 1$	vertex t must be mapped
$\rightarrow \quad -x_{p,1} + -x_{q,1} + -x_{s,1} + -x_{t,1} \geq -1$	use vertex 1 at most once
$\rightarrow \quad -x_{q,2} + -x_{r,2} \geq -1$	use vertex 2 at most once
$\rightarrow \quad -x_{q,3} + -x_{r,3} + -x_{s,3} + -x_{t,3} \geq -1$	use vertex 3 at most once
$-x_{p,1} \geq 1$	sum all of the above
$x_{p,1} \geq 0$	variable $x_{p,1}$ non-negative

Injectivity (All-Different) Reasoning

$$p \in \{1 \quad 4 \quad 5\}$$

$$q \in \{1 \; 2 \; 3\} \qquad x_{q,1} + x_{q,2} + x_{q,3} \geq 1 \quad \text{vertex } q \text{ must be mapped}$$

$$r \in \{2 \; 3\} \qquad x_{r,2} + x_{r,3} \geq 1 \quad \text{vertex } r \text{ must be mapped}$$

$$s \in \{1 \; 3\} \qquad x_{s,1} + x_{s,3} \geq 1 \quad \text{vertex } s \text{ must be mapped}$$

$$t \in \{1 \; 3\} \qquad x_{t,1} + x_{t,3} \geq 1 \quad \text{vertex } t \text{ must be mapped}$$

$$\rightarrow \quad -x_{p,1} + -x_{q,1} + -x_{s,1} + -x_{t,1} \geq -1 \quad \text{use vertex 1 at most once}$$

$$\rightarrow \quad -x_{q,2} + -x_{r,2} \geq -1 \quad \text{use vertex 2 at most once}$$

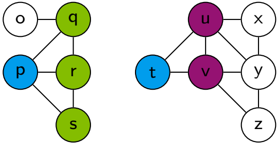
$$\rightarrow \quad -x_{q,3} + -x_{r,3} + -x_{s,3} + -x_{t,3} \geq -1 \quad \text{use vertex 3 at most once}$$

$$-x_{p,1} \geq 1 \quad \text{sum all of the above}$$

$$x_{p,1} \geq 0 \quad \text{variable } x_{p,1} \text{ non-negative}$$

$$0 \geq 1 \quad \text{sum above two constraints}$$

Degree Reasoning in Cutting Planes

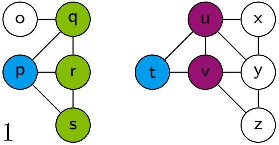


Pattern vertex p of degree $\deg(p)$ can never be mapped to target vertex t of degree $< \deg(p)$ in any subgraph isomorphism.

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



Adjacency:

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

Injectivity:

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

Literal axioms:

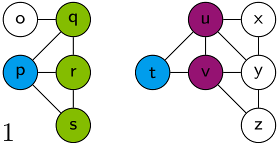
$$x_{o,u} \geq 0$$
$$x_{o,v} \geq 0$$
$$x_{p,u} \geq 0$$
$$x_{p,v} \geq 0$$

Add these together . . .

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

Gocht, McCreesh, Nordström: Subgraph Isomorphism Meets Cutting Planes: Solving With Certified Solutions, IJCAI 2020

Degree Reasoning in Cutting Planes



Adjacency:

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

Injectivity:

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

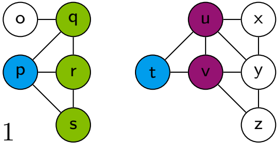
Literal axioms:

$$\begin{aligned}
x_{o,u} &\geq 0 \\
x_{o,v} &\geq 0 \\
x_{p,u} &\geq 0 \\
x_{p,v} &\geq 0
\end{aligned}$$

Add these together and divide by 3 to get

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

Degree Reasoning in Cutting Planes



Adjacency:

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

Injectivity:

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

Literal axioms:

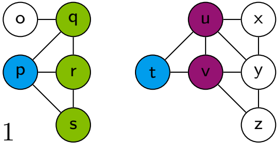
$$\begin{aligned}
x_{o,u} &\geq 0 \\
x_{o,v} &\geq 0 \\
x_{p,u} &\geq 0 \\
x_{p,v} &\geq 0
\end{aligned}$$

Add these together and divide by 3 to get

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

ChatGPT, can you draw me a picture of a sophisticated pigeon?

Degree Reasoning in Cutting Planes



Adjacency:

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

Injectivity:

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

Literal axioms:

$$\begin{aligned}
x_{o,u} &\geq 0\\
x_{o,v} &\geq 0\\
x_{p,u} &\geq 0\\
x_{p,v} &\geq 0
\end{aligned}$$

Add these together and divide by 3 to get

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

I think a sophisticated pigeon should have a monocle and a twirly mustache, rather than glasses and a beard.

Degree Reasoning in VeriPB

```
pol @adj_p_t_q @adj_p_t_r + @adj_p_t_s +     % sum adjacency constraints
    @inj_u + @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:

```
pol @adj_p_t_q @adj_p_t_r + @adj_p_t_s +     % sum adjacency constraints
    @inj_u + @inj_v + ;                      % sum injectivity constraints
ia 1 ~xp_t >= 1 : -1 ;                       % desired conclusion is syntactically implied
```

Other Forms of Reasoning

We can also log (almost) all of the other things state of the art subgraph solvers do, many of which do not involve pigeons:

- Conditional injectivity reasoning and filtering,
- Distance filtering,
- Neighbourhood degree sequences,
- Path filtering,
- Supplemental graphs.

Other Forms of Reasoning

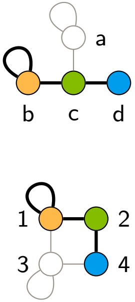
We can also log (almost) all of the other things state of the art subgraph solvers do, many of which do not involve pigeons:

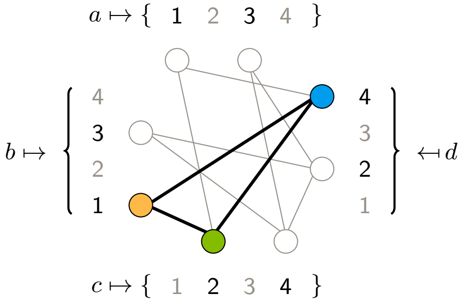
- Conditional injectivity reasoning and filtering,
- Distance filtering,
- Neighbourhood degree sequences,
- Path filtering,
- Supplemental graphs.

Proof steps are "efficient" using cutting planes:

- Length of proof $\approx$ time complexity of the reasoning algorithms.
- Most proof steps require only trivial additional computations.

Reformulation

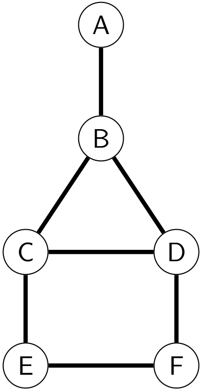




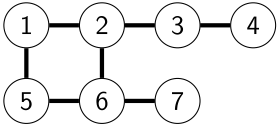
- We can encode this reduction using cutting planes rules.
- No need to modify the clique solver, either.

Pattern Graph Symmetries

- If a solution exists, a solution where $C < D$ exists.
- Might want to decide the constraint $C < D$ dynamically during search, or even to change constraint to $F < E$ for different subproblems.
- Enumeration proofs: "unsatisfiable, except for all the solutions I listed". But what does it mean to count or enumerate solutions under symmetries?

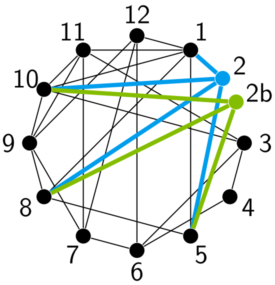


Target Graph (Conditional) Symmetries



- If 4 removed from all domains dynamically during search (all pattern vertices of degree 1 already assigned elsewhere?), a symmetry appears.
- If 3 and 7 removed, even more symmetries appear.
- We can still count when exploiting target symmetries, but we now seem to need non-trivial amounts of group theory to explain correctness efficiently.

Dominance



Can ignore vertex 2b.

- Every neighbour of 2b is also a neighbour of 2.
- Too expensive to detect upfront, so we catch it on backtrack instead.

Lemmas, Maybe?

- If pattern vertex P and its neighbourhood form a 5-clique, can't map to any vertex T whose neighbourhood does not form at least a 5-clique.
- Can combine all of the techniques discussed to justify this efficiently in terms of work done, but only for a specific choice of P and T.
- Solvers can reuse clique computations, though…

Conclusion

- Need to be able to express a wide range of reasoning rules, efficiently.
  - Much more variety of reasoning between solvers in other areas, compared to SAT solving.
  - Cutting planes can do this (which is somewhat mysterious).
- Mixing RUP and explicit derivations makes proofs much easier to write.
- No need for a close coupling between how the solver works and how the proof system works.

Conclusion

- Need to be able to express a wide range of reasoning rules, efficiently.
  - Much more variety of reasoning between solvers in other areas, compared to SAT solving.
  - Cutting planes can do this (which is somewhat mysterious).
- Mixing RUP and explicit derivations makes proofs much easier to write.
- No need for a close coupling between how the solver works and how the proof system works.
- ChatGPT is somewhat better at drawing pigeons than it is at maths.

ChatGPT, can you draw me a picture of a wise pigeon?

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk

University of Glasgow

Royal Academy of Engineering