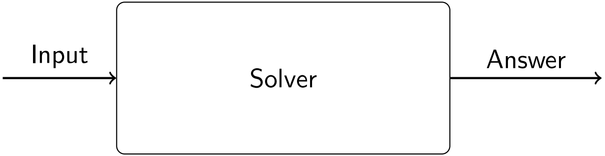
Pseudo-Boolean Proof Logging for Problems that are not Pseudo-Boolean

Ciaran McCreesh

University of Glasgow

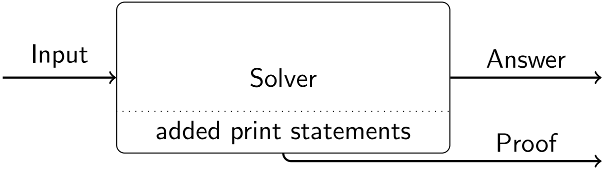
Royal Academy of Engineering

Proof Logging



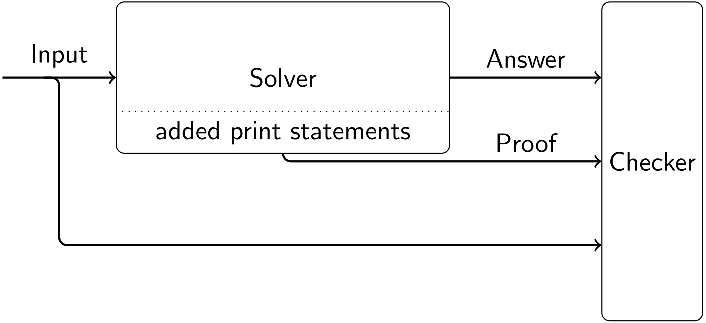
1. Run solver on problem input.

Proof Logging



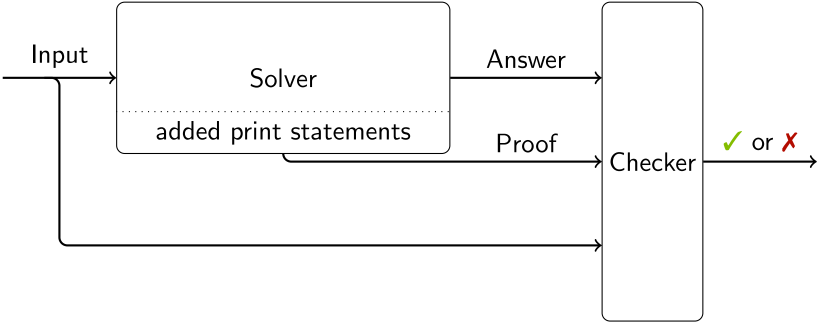
1. Run solver on problem input.
2. Solver also prints out a proof as part of its output.

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.

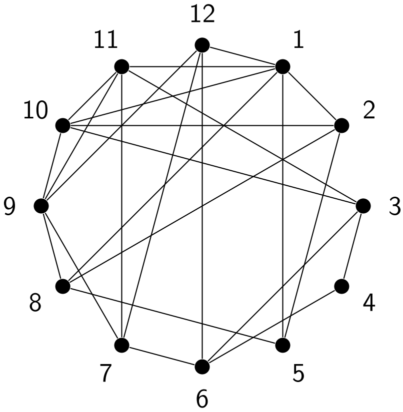
So Far...

- Pseudo-Boolean problems are a superset of SAT / CNF.
- Cutting planes is a superset of resolution.
- Decoupling solver language from proof language: easier or more efficient proofs if we can use a richer proof language, even if the solver isn't searching for proofs in that language.

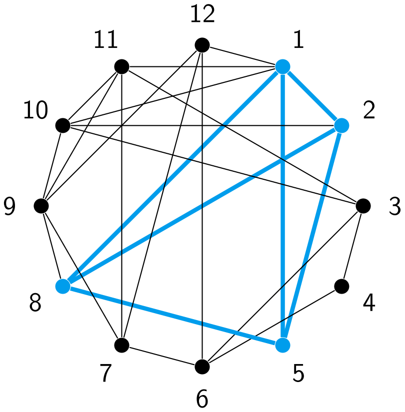
The Rest of This Talk

- Lots of more general algorithms that aren't thought of as doing "proof search".
- Extended cutting planes is still a good language for justifying their inferences.
- We can deal with non-Boolean variables.
- We can go beyond backtracking search and clause learning.
- Key point: can still take existing algorithms and techniques, and add print statements (albeit with more thinking and book-keeping required).

The Maximum Clique Problem



The Maximum Clique Problem



Maximum Clique Solvers

There are a lot of dedicated solvers for clique problems.

But there are issues:

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

Often used inside other solvers:

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

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)

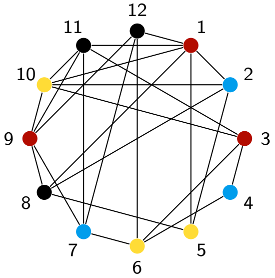
Key data structures:

- 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.

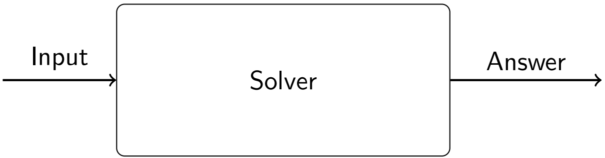
A Brief and Incomplete Guide to Clique Solving (4/4)

- 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.

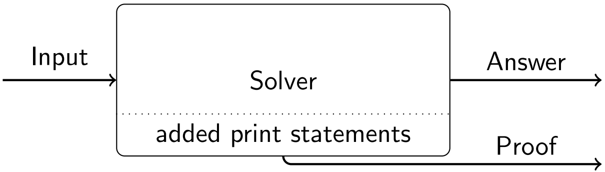
Making a Proof Logging Clique Solver

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

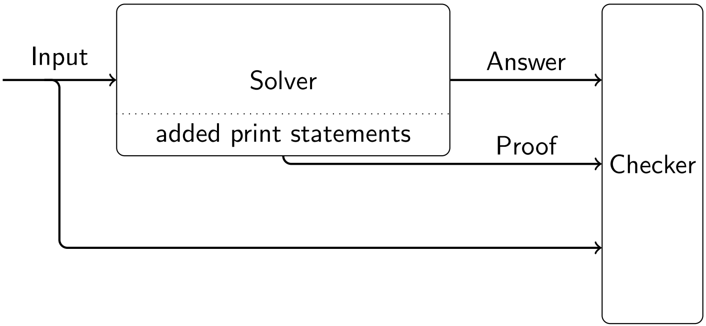
A Slightly Different Workflow



A Slightly Different Workflow

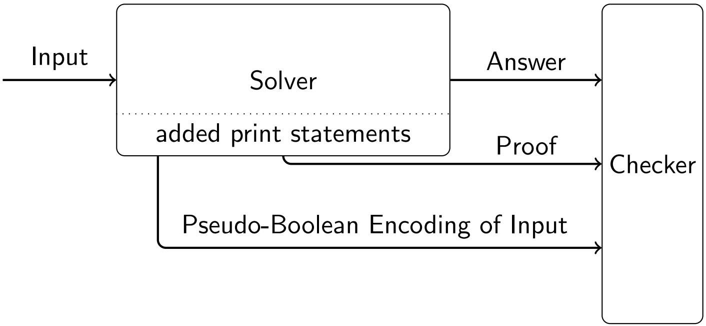


A Slightly Different Workflow

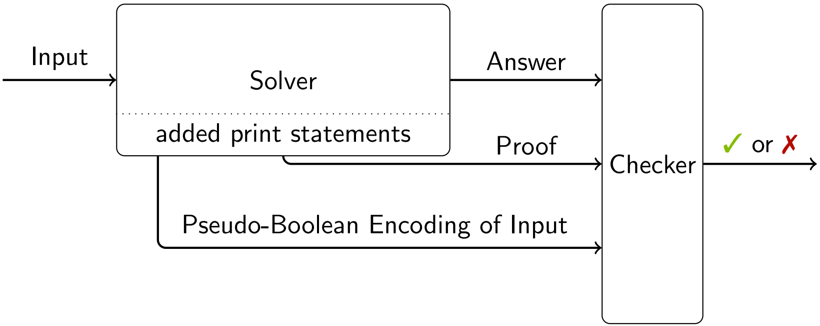


Gocht, McBride, McCreesh, Nordström, Prosser, Trimble: Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems, CP 2022

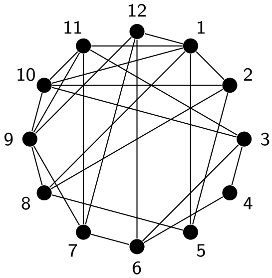
A Slightly Different Workflow



A Slightly Different Workflow



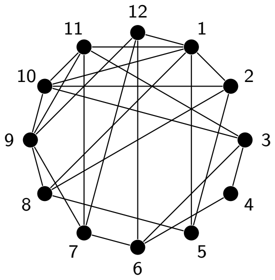
A Pseudo-Boolean Encoding for Clique (in OPB Format)




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

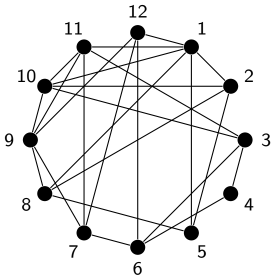
First Attempt at a Proof

```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



First Attempt at a Proof

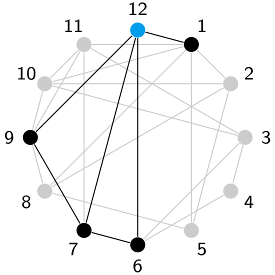
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Start with a header
Load the 41 problem axioms

First Attempt at a Proof

```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```

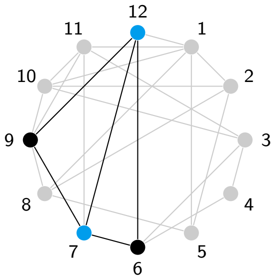


Branch accepting 12
Throw away non-adjacent vertices

Gocht, McBride, McCreesh, Nordström, Prosser, Trimble: Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems, CP 2022

First Attempt at a Proof

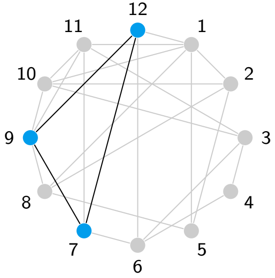
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch also accepting 7
Throw away non-adjacent vertices

First Attempt at a Proof

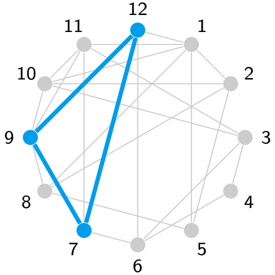
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch also accepting 9
Throw away non-adjacent vertices

First Attempt at a Proof

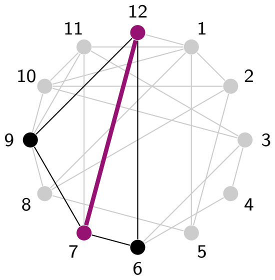
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

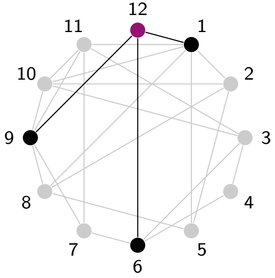
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

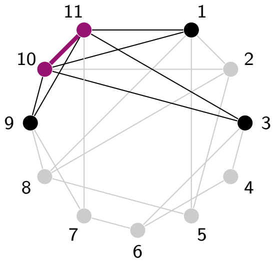
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

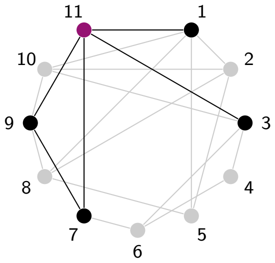
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

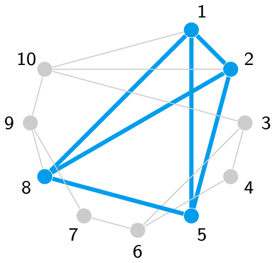
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

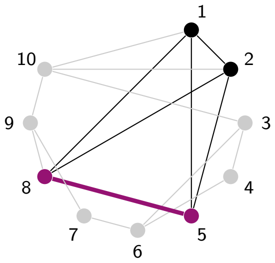
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

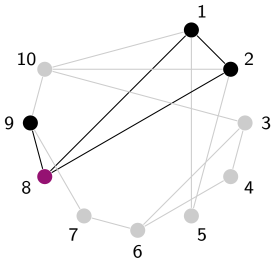
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



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

First Attempt at a Proof

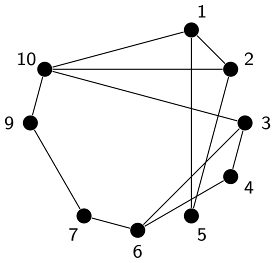
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Backtrack from 8
Still not enough vertices
Delete the vertex

First Attempt at a Proof

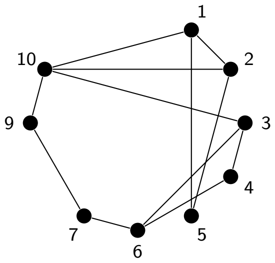
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Remaining graph is 3-colourable
Backtrack from root node

First Attempt at a Proof

```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Finish with what we've concluded

We specify a lower and an upper bound

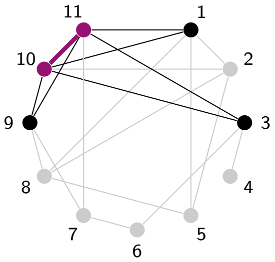
Remember we're minimising $\sum_v -1 \times v$, so a 4-clique has an objective value of -4

Verifying This Proof (Or Not. . .)

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

Verifying This Proof (Or Not. . .)

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



Verifying This Proof (Or Not. . .)

```
$ veripb --trace clique.opb clique-attempt-one.veripb
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
  ConstraintId 002: 1 ~x2 1 ~x3 >= 1
...
  ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: soli x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
  ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 >= 4
line 004: rup 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: rup 1 ~x12 >= 1 ;
  ConstraintId 044: 1 ~x12 >= 1
line 006: rup 1 ~x11 1 ~x10 >= 1 ;
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```

Dealing With Colourings

The colour bound doesn't follow by RUP...

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

Dealing With Colourings

The colour bound doesn't follow by RUP...

But 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. $x_1 + x_6 + x_9 \leq 1$

Dealing With Colourings

The colour bound doesn't follow by RUP...

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

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

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" cutting planes to write this derivation to proof log.

What This Looks Like in the Proof Log

```
pseudo-Boolean proof version 2.0
f 41
soli x12 x7 x9
rup 1 ~x12 1 ~x7 >= 1 ;
* bound, colour classes [ x1 x6 x9 ]
pol 7_{1≁6} 19_{1≁9} + 24_{6≁9} + 2 d
pol 42_obj -1 +
rup 1 ~x12 >= 1 ;
* bound, colour classes [ x1 x3 x9 ]
pol 1_{1≁3} 19_{1≁9} + 21_{3≁9} + 2 d
pol 42_obj -1 +
rup 1 ~x11 1 ~x10 >= 1 ;
* bound, colour classes [ x1 x3 x7 ]
* [ x9 ]
pol 1_{1≁3} 10_{1≁7} + 12_{3≁7} + 2 d
pol 42_obj -1 +
rup 1 ~x11 >= 1 ;
soli x8 x5 x2 x1
rup 1 ~x8 1 ~x5 >= 1 ;
* bound, colour classes [ x1 x9 ] [ x2 ]
pol 53_obj 19_{1≁9} +
rup 1 ~x8 >= 1 ;
* bound, colour classes [ x1 x3 x7 ]
* [ x2 x4 x9 ] [ x5 x6 x10 ]
pol 1_{1≁3} 10_{1≁7} + 12_{3≁7} + 2 d
pol 53_obj -1 +
pol 4_{2≁4} 20_{2≁9} + 22_{4≁9} + 2 d
pol 53_obj -3 + -1 +
pol 9_{5≁6} 26_{5≁10} + 27_{6≁10} + 2 d
pol 53_obj -5 + -3 + -1 +
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```

Verifying This Proof (For Real, This Time)

```
$ veripb --trace clique.opb clique-attempt-two.veripb
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
...
  ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: soli x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
  ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 >= 4
...
  ConstraintId 061: 1 ~x5 1 ~x6 1 ~x10 >= 2
line 028: pol 53 57 + 59 + 61 +
  ConstraintId 062: 1 x8 1 x11 1 x12 >= 2
line 029: rup >= 1 ;
  ConstraintId 063: >= 1
line 030: output NONE
line 031: conclusion BOUNDS -4 -4
line 032: end pseudo-Boolean proof
=== end trace ===
```

Different Clique Algorithms

Different search orders?

- ✓ Irrelevant for proof logging.

Using local search to initialise?

- ✓ Just log the incumbent.

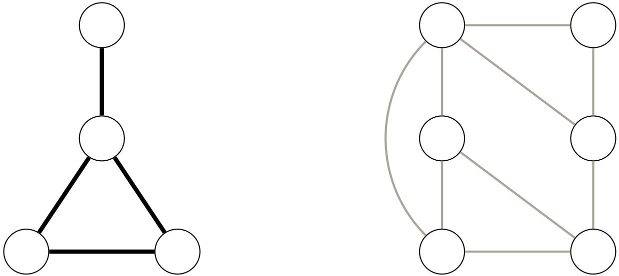
Different bound functions?

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

Weighted cliques?

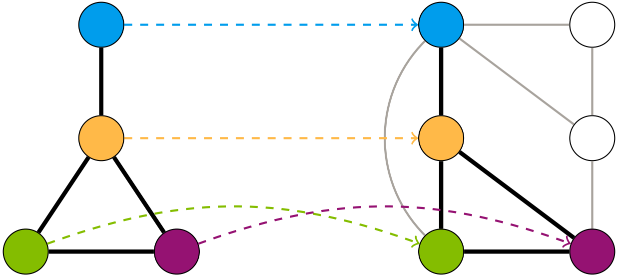
- ✓ Multiply a colour class by its largest weight.
- ✓ Also works for vertices "split between colour classes".

Subgraph Isomorphism



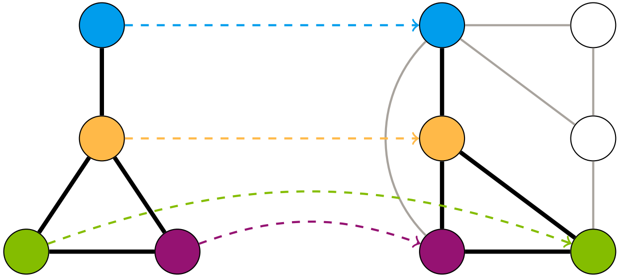
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches

Subgraph Isomorphism



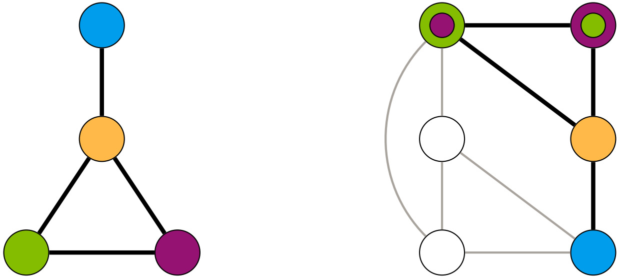
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches

Subgraph Isomorphism



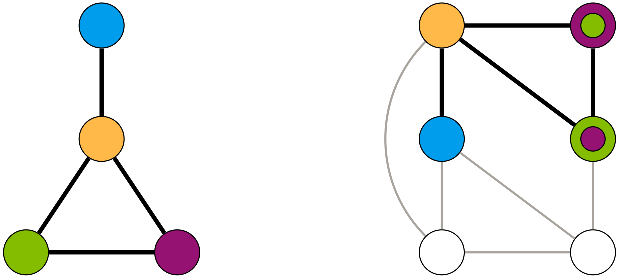
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches

Subgraph Isomorphism



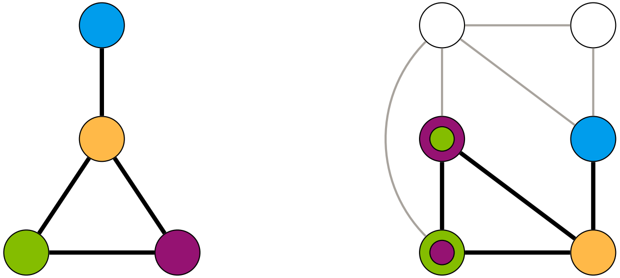
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, . . .
- Often want to find all matches

Subgraph Isomorphism



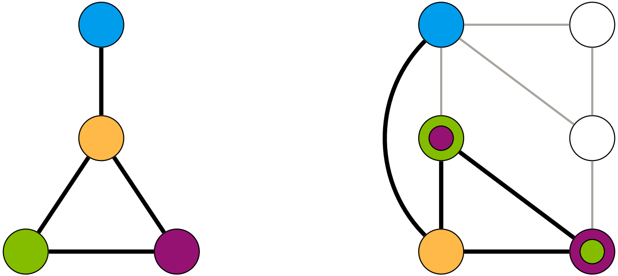
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, . . .
- Often want to find all matches

Subgraph Isomorphism



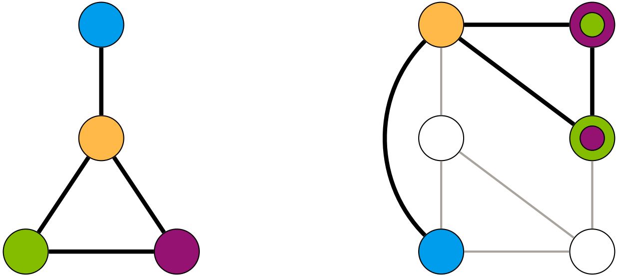
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches

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:

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

Each target vertex may be used at most once:

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

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:

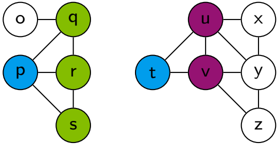
$$\sum_{p \in \mathrm{V}(P)} -x_{p,t} \geq -1 \qquad t \in \mathrm{V}(T)$$

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

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

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

Degree Reasoning in Cutting Planes

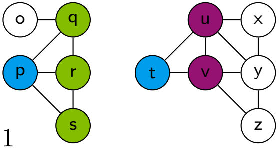


Pattern vertex p of degree $\deg(p)$ can never be mapped to target vertex t of degree $< \deg(p)$ in any subgraph isomorphism.

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:

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

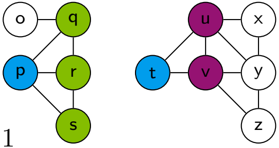
Literal axioms:

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

Add these together . . .

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

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 and divide by 3 to get

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

Degree Reasoning in VeriPB

pol $18_{p\sim t:q}$ $19_{p\sim t:r}$ + $20_{p\sim t:s}$ + * sum adjacency constraints
  $12_{inj(u)}$ + $13_{inj(v)}$ + * sum injectivity constraints
  xo_u + xo_v + * cancel stray xo_*
  xp_u + xp_v + * cancel stray xp_*
  3 d * divide, and we're done

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

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

Other Forms of Reasoning

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

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

Other Forms of Reasoning

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

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

Proof steps are "efficient" using cutting planes:

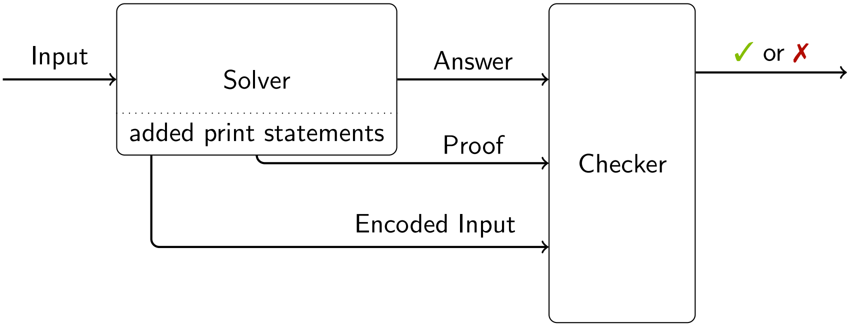
- Length of proof $\approx$ time complexity of the reasoning algorithms.
- Most proof steps require only trivial additional computations.

Code

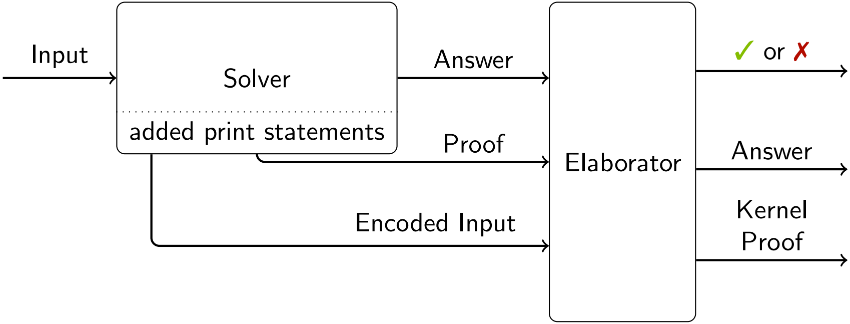
`https://github.com/ciaranm/glasgow-subgraph-solver`

Released under MIT Licence.

Reducing the Trust Base

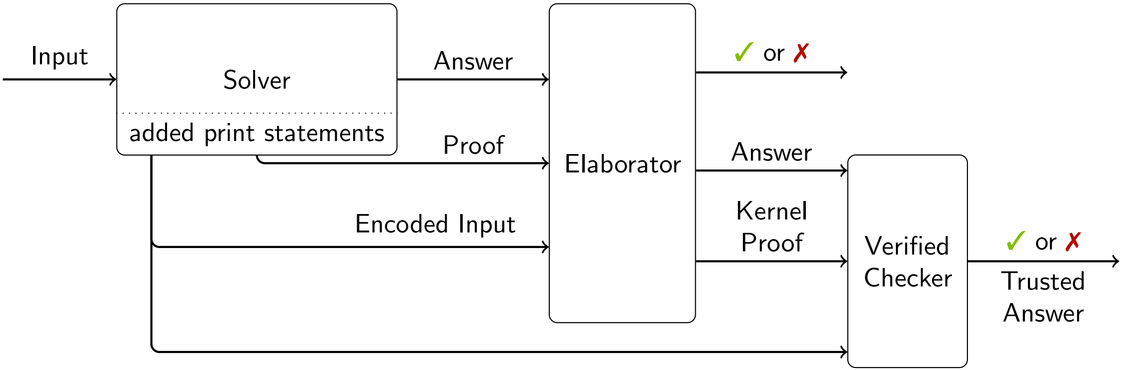


Reducing the Trust Base

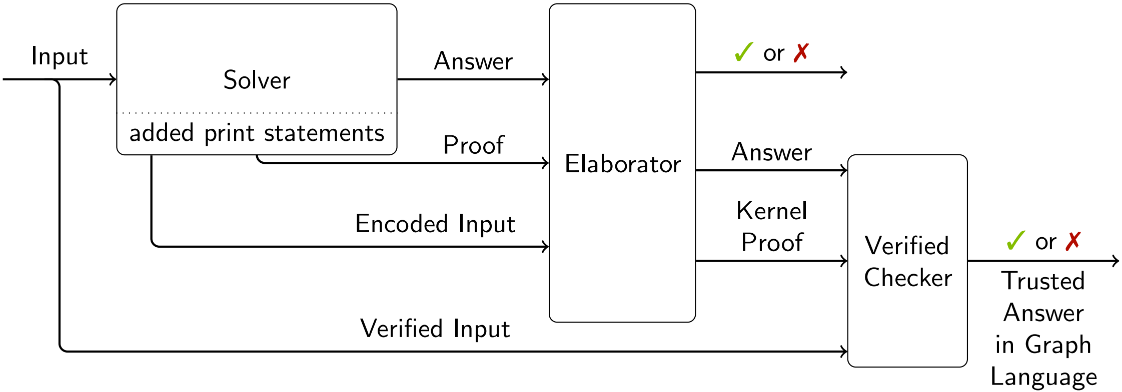


Gocht, McCreesh, Myreen, Nordström, Oertel, Tan: End-to-End Verification for Subgraph Solving, AAAI 2024

Reducing the Trust Base



Reducing the Trust Base



End-to-End Verification of Subgraph-Finding

```
$ glasgow_clique_solver brock200_4.clq --prove brock200_4 --proof-names --recover-proof-enc
omega = 17
clique = 12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

$ veripb proof.opb proof.pbp
Verification succeeded.

$ grep conclusion proof.pbp
conclusion BOUNDS 183 183

$ cake_pb_clique brock200_4.clq > brock200_4.verifiedopb

$ veripb proof.verifiedopb proof.pbp --proofOutput proof.corepb
Verification succeeded.

$ cake_pb_clique brock200_4.clq proof.corepb
s VERIFIED MAX CLIQUE SIZE |CLIQUE| = 17
```

What Exactly are we Verifying?

$$\text{is\_clique}\ vs\ (v,e) \stackrel{\text{def}}{=}$$
$$vs \subseteq \{\ 0,1,\ldots,v-1\ \} \ \wedge$$
$$\forall\, x\ y.\ x \in vs \wedge y \in vs \wedge x \neq y \Rightarrow \text{is\_edge}\ e\ x\ y$$
$$\text{max\_clique\_size}\ g \stackrel{\text{def}}{=} \max_{\text{set}} \{\ \text{card}\ vs \mid \text{is\_clique}\ vs\ g\ \}$$

What Exactly are we Verifying?

$$\text{clique\_eq\_str}\ n \overset{\text{def}}{=} \texttt{"s VERIFIED MAX CLIQUE SIZE |CLIQUE| = "}\ \hat{}\ \text{toString}\ n\ \hat{}\ \texttt{"\textbackslash n"}$$

$$\text{clique\_bound\_str}\ l\ u \overset{\text{def}}{=}$$
$$\texttt{"s VERIFIED MAX CLIQUE SIZE BOUND "}\ \hat{}\ \text{toString}\ l\ \hat{}\ \texttt{" <= |CLIQUE| <= "}\ \hat{}\ \text{toString}\ u\ \hat{}\ \texttt{"\textbackslash n"}$$

$$\begin{aligned}
&\vdash \text{cake_pb_clique_run}\ cl\ fs\ mc\ ms \Rightarrow \\
&\quad \text{machine_sem}\ mc\ (\text{basis_ffi}\ cl\ fs)\ ms \subseteq \\
&\quad\ \ \text{extend_with_resource_limit}\ \{\ \text{Terminate Success}\ (\text{cake_pb_clique_io_events}\ cl\ fs)\ \} \wedge \\
&\quad \exists\, out\ err. \\
&\quad\ \ \text{extract_fs}\ fs\ (\text{cake_pb_clique_io_events}\ cl\ fs) = \text{Some}\ (\text{add_stdout}\ (\text{add_stderr}\ fs\ err)\ out) \wedge \\
&\quad\ \ (out \neq \texttt{""} \Rightarrow \\
&\quad\quad \exists\, g.\ \text{get_graph_dimacs}\ fs\ (\text{el}\ 1\ cl) = \text{Some}\ g \wedge \\
&\quad\quad\quad (\text{length}\ cl = 2 \wedge out = \text{concat}\ (\text{print_pbf}\ (\text{full_encode}\ g)) \vee \\
&\quad\quad\quad\ \text{length}\ cl = 3 \wedge \\
&\quad\quad\quad\quad (out = \text{clique_eq_str}\ (\text{max_clique_size}\ g) \vee \\
&\quad\quad\quad\quad \exists\, l\ u. out = \text{clique_bound_str}\ l\ u \wedge (\forall\, vs.\ \text{is_clique}\ vs\ g \Rightarrow \text{card}\ vs \leq u) \wedge \\
&\quad\quad\quad\quad\quad\quad \exists\, vs.\ \text{is_clique}\ vs\ g \wedge l \leq \text{card}\ vs)))
\end{aligned}$$

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 or a subgraph isomorphism.
- Input parsing and output formatting.

No need to trust, or even know about:

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

Code

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

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

https://gitlab.com/MIAOresearch/software/CakePB

What About Constraint Programming?

Non-Boolean variables?

Constraints?

- Encoding constraints in pseudo-Boolean form?
- Justifying inferences?

Reformulations?

Compiling CP Variables (1/2)

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1$$

Compiling CP Variables (1/2)

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} \\ + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1$$

This doesn't work for large domains...

Compiling CP Variables (1/2)

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$
\begin{aligned}
a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3}& \\
+ a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} &= 1
\end{aligned}
$$

This doesn't work for large domains...

We could use a binary encoding:

$$
\begin{aligned}
-16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} &\geq -3 \qquad \text{and} \\
16a_{\text{neg}} + -1a_{\text{b0}} + -2a_{\text{b1}} + -4a_{\text{b2}} + -8a_{\text{b3}} &\geq -9
\end{aligned}
$$

This doesn't propagate much, but that isn't a problem for proof logging.

Compiling CP Variables (1/2)

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1$$

This doesn't work for large domains. . .

We could use a binary encoding:

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

This doesn't propagate much, but that isn't a problem for proof logging.

Convention in what follows:

- Upper-case A, B, C are CP variables;
- Lower-case a, b, c are corresponding Boolean variables in PB encoding.

Compiling CP Variables (2/2)

We can mix binary and an order encoding! Where needed, define:

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

Compiling CP Variables (2/2)

We can mix binary and an order encoding! Where needed, define:

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

When creating $a_{\geq i}$, also introduce pseudo-Boolean constraints encoding

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist.

Compiling CP Variables (2/2)

We can mix binary and an order encoding! Where needed, define:

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

When creating $a_{\geq i}$, also introduce pseudo-Boolean constraints encoding

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist.

We can do this:

- Inside the pseudo-Boolean model, where needed;
- Otherwise lazily during proof logging.

Compiling Constraints

- Also need to compile every constraint to pseudo-Boolean form.
- Doesn't need to be a propagating encoding.
- Can use additional variables.

Compiling Linear Inequalities

Given inequality

$$2A + 3B + 4C \geq 42$$

where $A, B, C \in \{-3 \ldots 9\}$,

Compiling Linear Inequalities

Given inequality

$$2A + 3B + 4C \geq 42$$

where $A, B, C \in \{-3 \ldots 9\}$,

Encode in pseudo-Boolean form as

$$\begin{aligned} &-32a_{\text{neg}} + 2a_{\text{b0}} + 4a_{\text{b1}} + 8a_{\text{b2}} + 16a_{\text{b3}} \\ &+ -48b_{\text{neg}} + 3b_{\text{b0}} + 6b_{\text{b1}} + 12b_{\text{b2}} + 24b_{\text{b3}} \\ &+ -64c_{\text{neg}} + 4c_{\text{b0}} + 8c_{\text{b1}} + 16c_{\text{b2}} + 32c_{\text{b3}} \geq 42 \end{aligned}$$

Compiling Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table.
Variable assignments must match some row in table.

Compiling Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table.
Variable assignments must match some row in table.

Given table constraint

$$(A, B, C) \in [(1,2,3),(1,3,4),(2,2,5)]$$

define

$$3\bar{t}_1 + a_{=1} + b_{=2} + c_{=3} \geq 3 \qquad \text{i.e. } t_1 \Rightarrow (a_{=1} \wedge b_{=2} \wedge c_{=3})$$

$$3\bar{t}_2 + a_{=1} + b_{=4} + c_{=4} \geq 3 \qquad \text{i.e. } t_2 \Rightarrow (a_{=1} \wedge b_{=4} \wedge c_{=4})$$

$$3\bar{t}_3 + a_{=2} + b_{=2} + c_{=5} \geq 3 \qquad \text{i.e. } t_3 \Rightarrow (a_{=2} \wedge b_{=2} \wedge c_{=5})$$

using tuple selector variables

$$t_1 + t_2 + t_3 = 1$$

Encoding Constraint Definitions

Already know how to do it for any constraint with a sane encoding using some combination of

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

Simplicity is important, propagation strength isn't.

Justifying Search

Mostly this works as in earlier examples.

Restarts are easy.

No need to justify guesses or decisions, only backtracking.

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log.

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log.

If it follows from unit propagation on the encoding, nothing needed

Some propagators and encodings need RUP steps for inferences

- A lot of propagators are effectively "doing a little bit of lookahead" but in an efficient way.

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log.

If it follows from unit propagation on the encoding, nothing needed

Some propagators and encodings need RUP steps for inferences

- A lot of propagators are effectively “doing a little bit of lookahead” but in an efficient way.

A few need explicit cutting planes justifications written to the proof log:

- Linear inequalities just need to multiply and add.
- All-different needs a bit more.
- Might need the help of a good PhD student for some propagators.

Justifying All-Different Failures

$$
\begin{aligned}
V &\in \{1 \quad\quad\quad 4 \quad 5\} \\
W &\in \{1 \quad 2 \quad 3 \quad\quad\quad\} \\
X &\in \{\quad\; 2 \quad 3 \quad\quad\quad\} \\
Y &\in \{1 \quad\quad 3 \quad\quad\quad\} \\
Z &\in \{1 \quad\quad 3 \quad\quad\quad\}
\end{aligned}
$$

Justifying All-Different Failures

$$
\begin{aligned}
V &\in \{1 \quad \quad \quad 4 \quad 5\} \\
W &\in \{1 \quad 2 \quad 3 \quad \quad \quad\} \\
X &\in \{\quad 2 \quad 3 \quad \quad \quad\} \\
Y &\in \{1 \quad \quad 3 \quad \quad \quad\} \\
Z &\in \{1 \quad \quad 3 \quad \quad \quad\}
\end{aligned}
$$

Justifying All-Different Failures

$$V \in \{1 \quad 4 \quad 5\}$$
$$W \in \{1 \quad 2 \quad 3\} \qquad w_{=1} + w_{=2} + w_{=3} \geq 1 \quad [\,W \text{ takes some value}\,]$$
$$X \in \{2 \quad 3\}$$
$$Y \in \{1 \quad 3\}$$
$$Z \in \{1 \quad 3\}$$

Justifying All-Different Failures

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

$W \in \{1\ 2\ 3\}$	$w_{=1} + w_{=2} + w_{=3} \geq 1$	[W takes some value]
$X \in \{2\ 3\}$	$x_{=2} + x_{=3} \geq 1$	[X takes some value]
$Y \in \{1\ 3\}$	$y_{=1} + y_{=3} \geq 1$	[Y takes some value]
$Z \in \{1\ 3\}$	$z_{=1} + z_{=3} \geq 1$	[Z takes some value]

Justifying All-Different Failures

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

$$W \in \{1 \quad 2 \quad 3\} \qquad w_{=1} + w_{=2} + w_{=3} \geq 1 \quad [\, W \text{ takes some value} \,]$$

$$X \in \{2 \quad 3\} \qquad x_{=2} + x_{=3} \geq 1 \quad [\, X \text{ takes some value} \,]$$

$$Y \in \{1 \quad 3\} \qquad y_{=1} + y_{=3} \geq 1 \quad [\, Y \text{ takes some value} \,]$$

$$Z \in \{1 \quad 3\} \qquad z_{=1} + z_{=3} \geq 1 \quad [\, Z \text{ takes some value} \,]$$

$$\rightarrow \quad -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 \quad [\, \text{At most one variable} = 1 \,]$$

$$\rightarrow \quad -w_{=2} + -x_{=2} \geq -1 \quad [\, \text{At most one variable} = 2 \,]$$

$$\rightarrow \quad -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1 \quad [\, \text{At most one variable} = 3 \,]$$

Justifying All-Different Failures

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

$W \in \{1\ 2\ 3\}$	$w_{=1} + w_{=2} + w_{=3}$	≥ 1	[W takes some value]
$X \in \{2\ 3\}$	$x_{=2} + x_{=3}$	≥ 1	[X takes some value]
$Y \in \{1\ 3\}$	$y_{=1} + y_{=3}$	≥ 1	[Y takes some value]
$Z \in \{1\ 3\}$	$z_{=1} + z_{=3}$	≥ 1	[Z takes some value]
$\rightarrow$	$-v_{=1} + -w_{=1} + -y_{=1} + -z_{=1}$	≥ -1	[At most one variable $= 1$]
$\rightarrow$	$-w_{=2} + -x_{=2}$	≥ -1	[At most one variable $= 2$]
$\rightarrow$	$-w_{=3} + -x_{=3} + -y_{=3} + -z_{=3}$	≥ -1	[At most one variable $= 3$]
	$-v_{=1}$	≥ 1	[Sum all constraints so far]

Justifying All-Different Failures

$$V \in \{1 \quad 4 \quad 5\}$$
$$W \in \{1 \quad 2 \quad 3\} \qquad w_{=1} + w_{=2} + w_{=3} \geq 1 \quad [\text{ } W \text{ takes some value}]$$
$$X \in \{2 \quad 3\} \qquad x_{=2} + x_{=3} \geq 1 \quad [\text{ } X \text{ takes some value}]$$
$$Y \in \{1 \quad 3\} \qquad y_{=1} + y_{=3} \geq 1 \quad [\text{ } Y \text{ takes some value}]$$
$$Z \in \{1 \quad 3\} \qquad z_{=1} + z_{=3} \geq 1 \quad [\text{ } Z \text{ takes some value}]$$

$$\rightarrow \quad -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 \quad [\text{ At most one variable} = 1 \text{ }]$$
$$\rightarrow \quad -w_{=2} + -x_{=2} \geq -1 \quad [\text{ At most one variable} = 2 \text{ }]$$
$$\rightarrow \quad -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1 \quad [\text{ At most one variable} = 3 \text{ }]$$

$$-v_{=1} \geq 1 \quad [\text{ Sum all constraints so far }]$$
$$v_{=1} \geq 0 \quad [\text{ Variable } v_{=1} \text{ non-negative }]$$

Justifying All-Different Failures

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

$W \in \{1\ 2\ 3\}$	$w_{=1} + w_{=2} + w_{=3}$	≥ 1	[W takes some value]
$X \in \{2\ 3\}$	$x_{=2} + x_{=3}$	≥ 1	[X takes some value]
$Y \in \{1\ 3\}$	$y_{=1} + y_{=3}$	≥ 1	[Y takes some value]
$Z \in \{1\ 3\}$	$z_{=1} + z_{=3}$	≥ 1	[Z takes some value]
$\rightarrow$	$-v_{=1} + -w_{=1} + -y_{=1} + -z_{=1}$	≥ -1	[At most one variable $= 1$]
$\rightarrow$	$-w_{=2} + -x_{=2}$	≥ -1	[At most one variable $= 2$]
$\rightarrow$	$-w_{=3} + -x_{=3} + -y_{=3} + -z_{=3}$	≥ -1	[At most one variable $= 3$]
	$-v_{=1}$	≥ 1	[Sum all constraints so far]
	$v_{=1}$	≥ 0	[Variable $v_{=1}$ non-negative]
	0	≥ 1	[Sum above two constraints]

Code

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

Released under MIT Licence.

Partial MiniZinc support, more soon. A growing collection of global constraints:

- Absolute value.
- All-different.
- Circuit (check, prevent, SCC).
- Count.
- Element.
- Inverse.
- Knapsack.
- Minumum and Maximum.
- n Value.
- Parity.
- (Reified) integer linear (in)equalities (with large domains, and GAC reformulation).
- Regular (and hence Stretch, Geost, DiffN).
- Smart Table (and hence Lex, At Most One, Not All Equal).

Knapsack Problems

$x_i \in \{0, 1\}$	whether or not we take item i
$\sum_i \boldsymbol{w}_i x_i \leq W$	total weight of items taken not too heavy
$\text{maximise} \sum_i \boldsymbol{p}_i x_i$	yay capitalism

For our running example,

$$\boldsymbol{w} = [2, 5, 2, 3, 2, 3] \text{ and}$$
$$\boldsymbol{p} = [2, 4, 2, 5, 4, 3] \text{ with}$$
$$W \leq 7$$

Dynamic Programming for Knapsack

To decide whether we're taking the ith item, with w weight available to spend,

$$
\begin{aligned}
&P(i, w) = \max(\\
&\qquad P(i-1, w), \\
&\qquad P(i-1, w-\boldsymbol{w}_i) + \boldsymbol{p}_i \text{ if } \boldsymbol{w}_i \leq w \\
&) \\
&P(0, w) = 0
\end{aligned}
$$

Sparse Dynamic Programming

Key ideas:

- "Maximum" selects between partial sums on the same items with the same combined weights but different profits.
- Don't calculate the same state more than once.
- Only calculate partial sums of weights and profits that can actually be achieved.

Algorithmic details matter a lot for performance, but end up being more or less the same for proof logging.

Merging More States

The “maximum” means, if we could reach states

$$\sum_{i=1}^{\ell} \boldsymbol{w}_i = w \text{ and } \sum_{i=1}^{\ell} \boldsymbol{p}_i = p \qquad \text{or} \qquad \sum_{i=1}^{\ell} \boldsymbol{w}_i = w \text{ and } \sum_{i=1}^{\ell} \boldsymbol{p}_i = p'$$

with $p > p'$ then we only need to consider the state with profit p.

Merging More States

The "maximum" means, if we could reach states

$$\sum_{i=1}^{\ell} \boldsymbol{w}_i = w \text{ and } \sum_{i=1}^{\ell} \boldsymbol{p}_i = p \qquad \text{or} \qquad \sum_{i=1}^{\ell} \boldsymbol{w}_i = w \text{ and } \sum_{i=1}^{\ell} \boldsymbol{p}_i = p'$$

with $p > p'$ then we only need to consider the state with profit p.

More generally, if we have two states

$$\sum_{i=1}^{\ell} \boldsymbol{w}_i = w \text{ and } \sum_{i=1}^{\ell} \boldsymbol{p}_i = p \qquad \text{or} \qquad \sum_{i=1}^{\ell} \boldsymbol{w}_i = w' \text{ and } \sum_{i=1}^{\ell} \boldsymbol{p}_i = p'$$

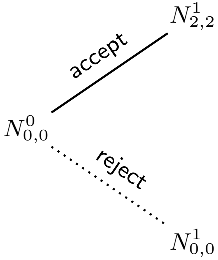
with $p \geq p'$ and $w \leq w'$ then we need only consider the former.

Whether or not this can be detected efficiently depends upon how the algorithm is implemented.

Viewing Dynamic Programming as a Decision Diagram

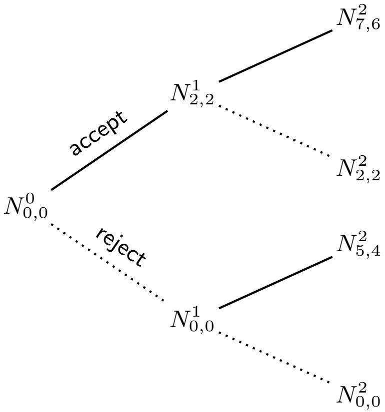
$N^0_{0,0}$

Viewing Dynamic Programming as a Decision Diagram



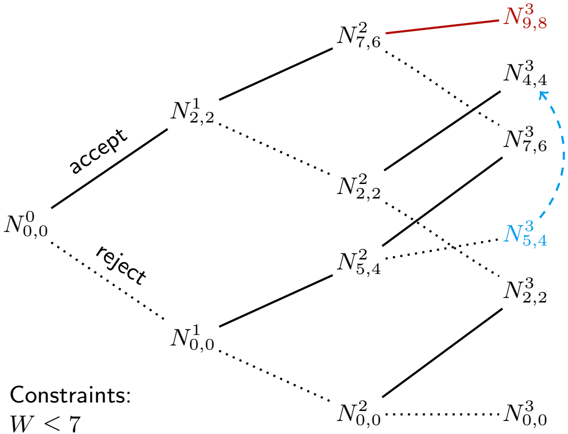
$\boldsymbol{w}_1 = 2, \boldsymbol{p}_1 = 2$

Viewing Dynamic Programming as a Decision Diagram



$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$

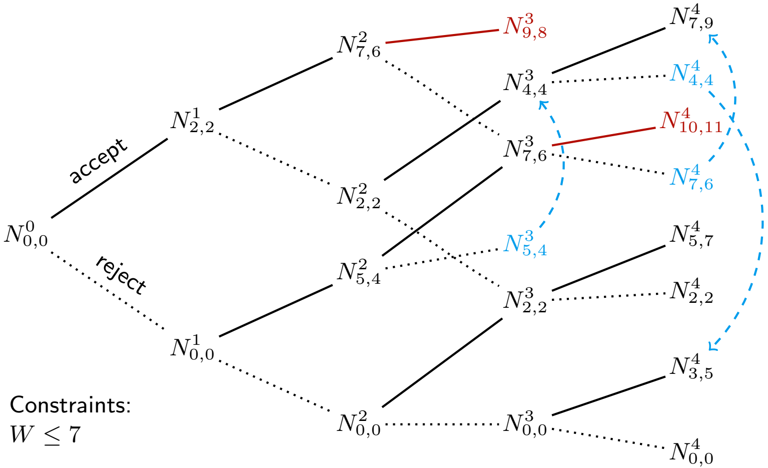
Viewing Dynamic Programming as a Decision Diagram



Constraints:
$W \leq 7$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4 \quad \boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$

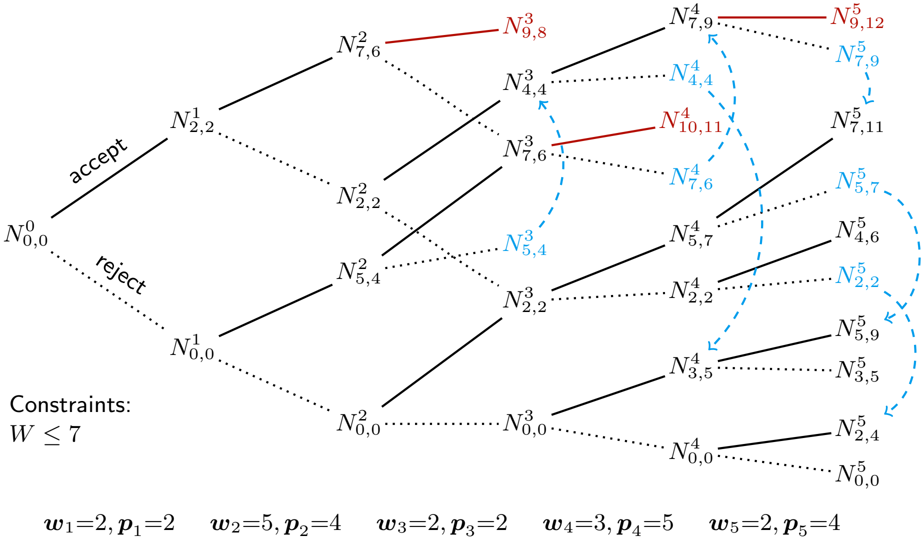
Viewing Dynamic Programming as a Decision Diagram



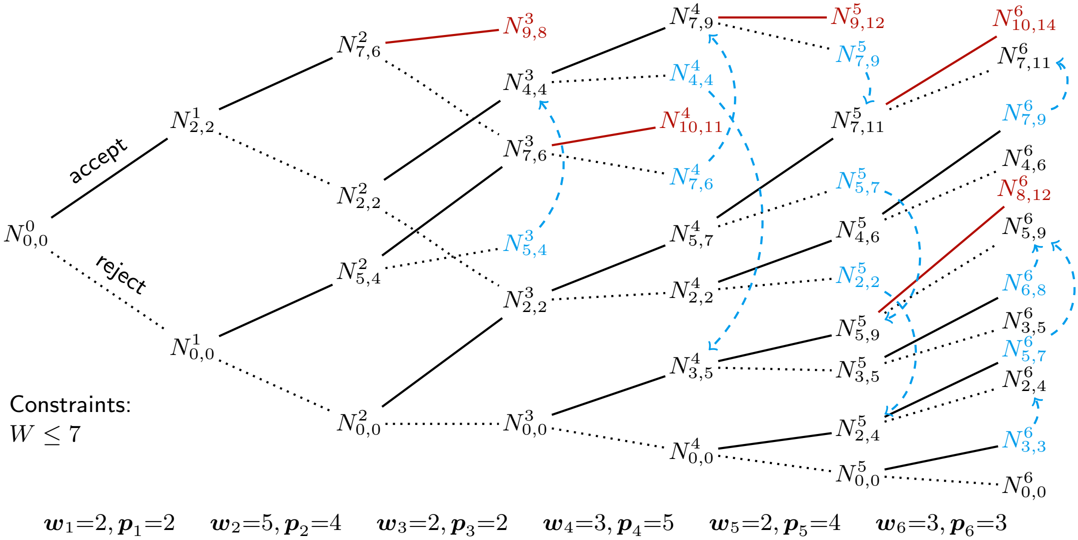
Constraints:
$W \leq 7$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2$ $\boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$ $\boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$ $\boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5$

Viewing Dynamic Programming as a Decision Diagram



Viewing Dynamic Programming as a Decision Diagram



Is This Correct?

Do you trust the theory?

Is This Correct?

Do you trust the theory?

Do you trust your PhD student to implement it correctly?

Is This Correct?

Do you trust the theory?

Do you trust your PhD student to implement it correctly?

Would you trust this inside a larger solver, where side constraints could apply?

Is This Correct?

Do you trust the theory?

Do you trust your PhD student to implement it correctly?

Would you trust this inside a larger solver, where side constraints could apply?

Knapsack as a Pseudo-Boolean Problem

$$2x_1 + 5x_2 + 2x_3 + 3x_4 + 2x_5 + 3x_6 \leq 7$$
$$\text{maximise } 2x_1 + 4x_2 + 2x_3 + 5x_4 + 4x_5 + 3x_6$$

We must *describe* knapsack in pseudo-Boolean terms, but our solver can do whatever it likes.

Proofs for Dynamic Programming Algorithms for Knapsack

- For backtracking search, we constructed a proof tree out of RUP steps.
- For dynamic programming:
  - Use extension variables to describe states.
  - Prove implications between states to create a decision diagram.

Extension Variables for States

For each state (or entry in the matrix) on layer ℓ, create extension variables

$$W_w^\ell \Leftrightarrow \sum_{i=1}^{\ell} \boldsymbol{w}_i x_i \geq w$$

$$P_p^\ell \Leftrightarrow \sum_{i=1}^{\ell} \boldsymbol{p}_i x_i \leq p$$

$$N_{w,p}^\ell \Leftrightarrow W_w^\ell + P_p^\ell \geq 2$$

Transitioning Between States

We don't have to take an item on layer ℓ, so need to prove:

$$W_w^{\ell-1} \wedge \overline{x}_\ell \Rightarrow W_w^\ell$$
$$P_p^{\ell-1} \wedge \overline{x}_\ell \Rightarrow P_p^\ell$$
$$N_{w,p}^{\ell-1} \wedge \overline{x}_\ell \Rightarrow N_{w,p}^\ell$$

Transitioning Between States

We don't have to take an item on layer ℓ, so need to prove:

$$W_w^{\ell-1} \wedge \overline{x}_\ell \Rightarrow W_w^\ell$$
$$P_p^{\ell-1} \wedge \overline{x}_\ell \Rightarrow P_p^\ell$$
$$N_{w,p}^{\ell-1} \wedge \overline{x}_\ell \Rightarrow N_{w,p}^\ell$$

If we can't take item on layer ℓ, need to prove:

$$W_w^{\ell-1} \Rightarrow \overline{x}_\ell$$
$$N_{w,p}^{\ell-1} \Rightarrow \overline{x}_\ell$$
$$N_{w,p}^{\ell-1} \Rightarrow N_{w,p}^\ell$$

Transitioning Between States

We don't have to take an item on layer ℓ, so need to prove:

$$W_w^{\ell-1} \wedge \overline{x}_\ell \Rightarrow W_w^\ell$$
$$P_p^{\ell-1} \wedge \overline{x}_\ell \Rightarrow P_p^\ell$$
$$N_{w,p}^{\ell-1} \wedge \overline{x}_\ell \Rightarrow N_{w,p}^\ell$$

If we can't take item on layer ℓ, need to prove:

$$W_w^{\ell-1} \Rightarrow \overline{x}_\ell$$
$$N_{w,p}^{\ell-1} \Rightarrow \overline{x}_\ell$$
$$N_{w,p}^{\ell-1} \Rightarrow N_{w,p}^\ell$$

If we can take item on layer ℓ, we need to prove:

$$W_w^{\ell-1} \wedge x_\ell \Rightarrow W_{w'}^\ell$$
$$P_p^{\ell-1} \wedge x_\ell \Rightarrow P_{p'}^\ell$$
$$N_{w,p}^{\ell-1} \wedge x_\ell \Rightarrow N_{w',p'}^\ell$$
$$N_{w,p}^{\ell-1} \Rightarrow N_{w,p}^\ell + N_{w',p'}^\ell \geq 1$$

where

$$(w', p') = (w + \boldsymbol{w}_\ell, p + \boldsymbol{p}_\ell)$$

Merged States

For each $N^\ell_{w,p}$ that is dominated by some other $N^\ell_{w',p'}$, we prove $N^\ell_{w,p} \Rightarrow N^\ell_{w',p'}$.

We can do this by unwrapping the conjunction, proving

$$W^\ell_w \Rightarrow W^\ell_{w'} \quad \text{i.e.} \quad (\sum_{i=1}^{\ell} \boldsymbol{w}_i x_i \geq w) \Rightarrow (\sum_{i=1}^{\ell} \boldsymbol{w}_i x_i \geq w') \text{ for some } w' \leq w$$

$$P^\ell_p \Rightarrow P^\ell_{p'} \quad \text{i.e.} \quad (\sum_{i=1}^{\ell} \boldsymbol{p}_i x_i \geq p) \Rightarrow (\sum_{i=1}^{\ell} \boldsymbol{p}_i x_i \geq p') \text{ for some } p' \geq p$$

"If there is an assignment to the first ℓ x_i variables where the weight sums to at least 7 and the profit to no more than 4, then there is an assignment where the weight sums to at least 6 and the profit to no more than 5".

Establishing Completeness

Must show that we have to be in one of the states on this layer,

$$\sum_{(w,p)\text{ on layer }\ell} N^{\ell}_{w,p} \geq 1$$

We can use the at-least-one constraint

$$\sum_{(w,p)\text{ on layer }\ell-1} N^{\ell-1}_{w,p} \geq 1$$

from the previous layer, and resolve on each

$$N^{\ell-1}_{w,p} \Rightarrow N^{\ell}_{w,p} + N^{\ell}_{w',p'} \geq 1 \qquad \text{or} \qquad N^{\ell-1}_{w,p} \Rightarrow N^{\ell}_{w,p}$$

Reading Off a Conclusion

We can log an optimal solution, and get a solution-improving constraint.

We have an at-least-one constraint over feasible states on the final layer, which we can unwrap to only talk about profits.

The solution-improving constraint contradicts each entry in the at-least-one constraint.

Autotabulation

- Sometimes advantageous to replace several constraints over the same variables with a single table constraint.
- Can be done by a skilled modeller, or by a solver automatically.
  - But what if the modeller or solver makes a mistake?
- We can do this inside the proof, rather than inside the model.

Gocht, McCreesh, Nordström: An Auditable Constraint Programming Solver, CP 2022

Autotabulation Proofs

- Run a search over a restricted subset of variables.
- Whenever we find a solution, create an extension variable

$$t_i \Leftrightarrow (x_{=1} \wedge y_{=2} \wedge z_{=4})$$

- For the remainder of the proof, add $\bar{t}_i$ as a guess.
- End up deriving $\wedge_i \bar{t}_i \Rightarrow \bot$, which is $\vee_i t_i$ or $\sum_i t_i \geq 1$.

Knapsack as a Constraint

$$x_i \in \{0, 1, \text{maybe other non-negative values}\}$$
$$W, P \in \{\text{some domain of non-negative values}\}$$
$$W = \sum_i \boldsymbol{w}_i x_i$$
$$P = \sum_i \boldsymbol{p}_i x_i$$

Now we can have lower and upper bounds on both W and P, and maybe we can reason that some items must or must not be taken.

Effectively we're solving two (or one, or more?) non-negative integer linear equations simultaneously.

Decision Diagrams are Table Constraints but Better

- Can often represent the solution set compactly as a decision diagram.
- Decision diagrams are like tables, but with bits of the table merged together.

A Change of States

For each state (or entry in the matrix) on layer ℓ, define

$$W{\uparrow}^{\ell}_{w} \Leftrightarrow \sum_{i=1}^{\ell} \boldsymbol{w}_i x_i \geq w \quad \text{and} \quad W{\downarrow}^{\ell}_{w} \Leftrightarrow \sum_{i=1}^{\ell} \boldsymbol{w}_i x_i \leq w$$

$$P{\uparrow}^{\ell}_{p} \Leftrightarrow \sum_{i=1}^{\ell} \boldsymbol{p}_i x_i \geq p \quad \text{and} \quad P{\downarrow}^{\ell}_{p} \Leftrightarrow \sum_{i=1}^{\ell} \boldsymbol{p}_i x_i \leq p$$

$$N^{\ell}_{w,p} \Leftrightarrow W{\uparrow}^{\ell}_{w} + W{\downarrow}^{\ell}_{w} + P{\uparrow}^{\ell}_{p} + P{\downarrow}^{\ell}_{p} \geq 4$$

So now our states represent *exact* weights and profits.

A Change of Merge Rules

We can no longer merge non-identical states!

Reassuringly, the proofs won't work if you try this. . .

End up trying to prove "if there is an assignment to the first ℓ x_i variables where the weight sums to *exactly* 7 and the profit to *exactly* 4, then there is an assignment where the weight sums to *exactly* 6 and the profit to *exactly* 5.

Establishing Arc Consistency

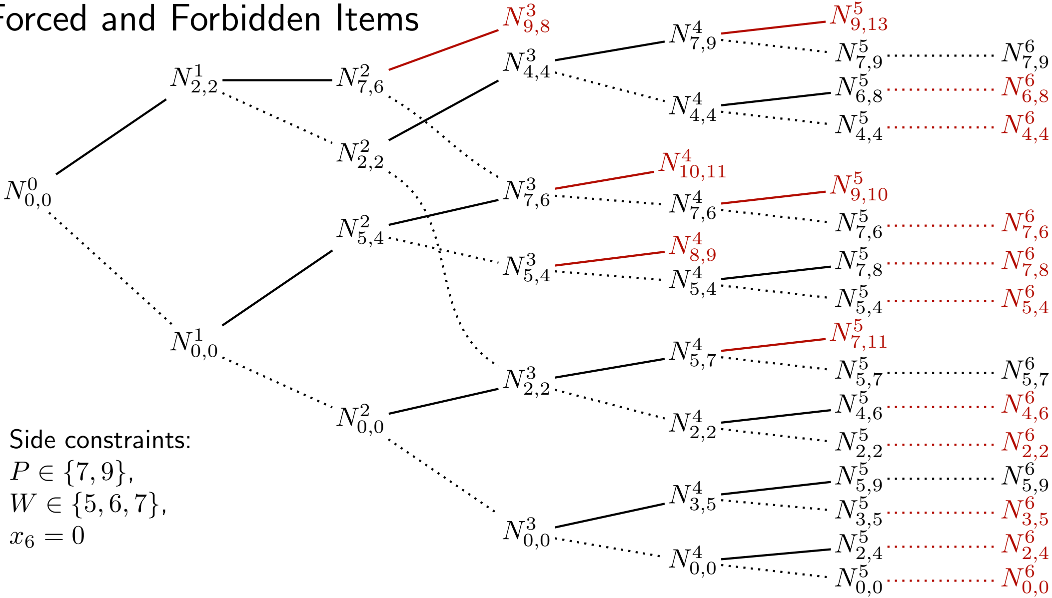
We can read off all possible values for P and W from the final layer.

Easy to use this and resolution with the at-least-one constraint to eliminate all other values.

If we used weaker state reification variables, we could merge more states but would get weaker consistency on the variables.

But what about the x_i variables?

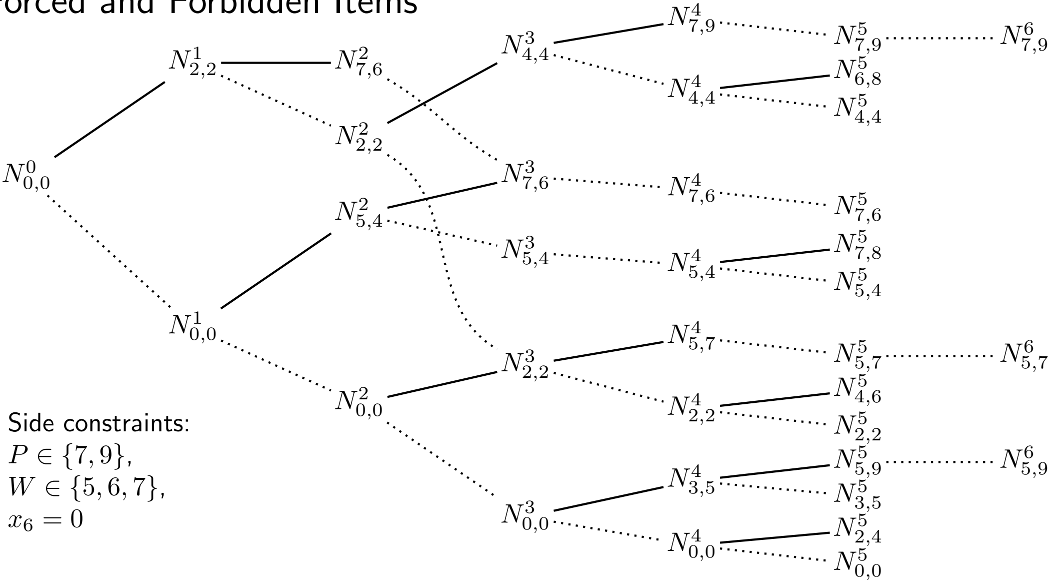
Forced and Forbidden Items



Side constraints:
$P \in \{7, 9\}$,
$W \in \{5, 6, 7\}$,
$x_6 = 0$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4 \quad \boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2 \quad \boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5 \quad \boldsymbol{w}_5{=}2, \boldsymbol{p}_5{=}4 \quad \boldsymbol{w}_6{=}3, \boldsymbol{p}_6{=}3$

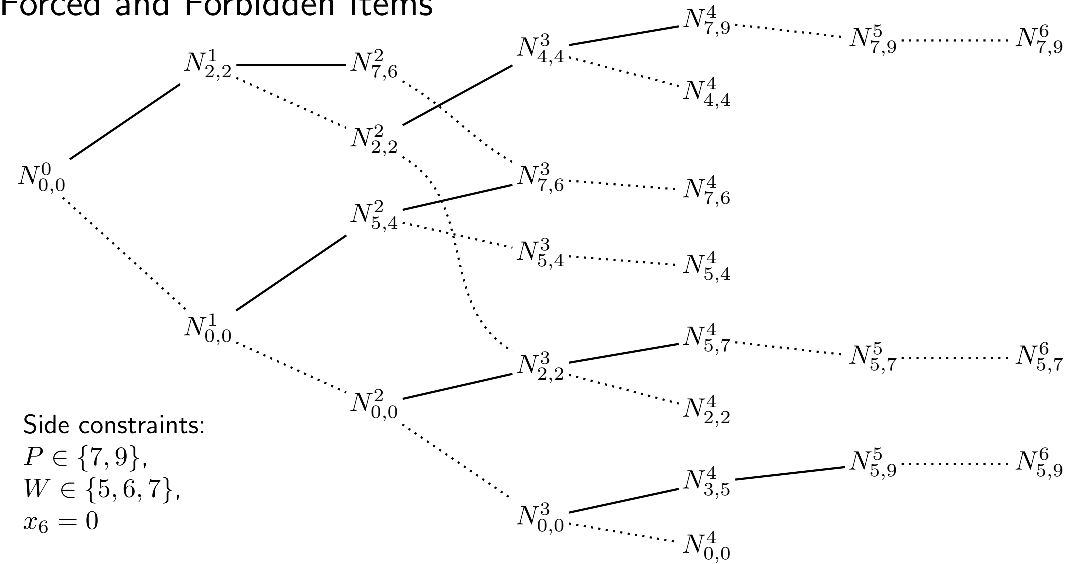
Forced and Forbidden Items



Side constraints:
$P \in \{7, 9\}$,
$W \in \{5, 6, 7\}$,
$x_6 = 0$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2$ $\boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$ $\boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$ $\boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5$ $\boldsymbol{w}_5{=}2, \boldsymbol{p}_5{=}4$ $\boldsymbol{w}_6{=}3, \boldsymbol{p}_6{=}3$

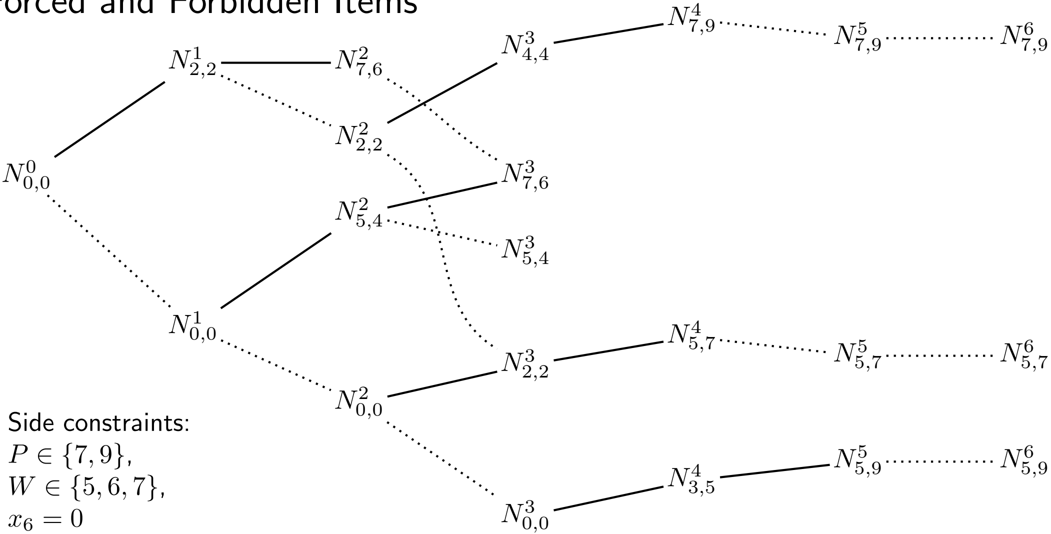
Forced and Forbidden Items



Side constraints:
$P \in \{7, 9\}$,
$W \in \{5, 6, 7\}$,
$x_6 = 0$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2$ $\boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$ $\boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$ $\boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5$ $\boldsymbol{w}_5{=}2, \boldsymbol{p}_5{=}4$ $\boldsymbol{w}_6{=}3, \boldsymbol{p}_6{=}3$

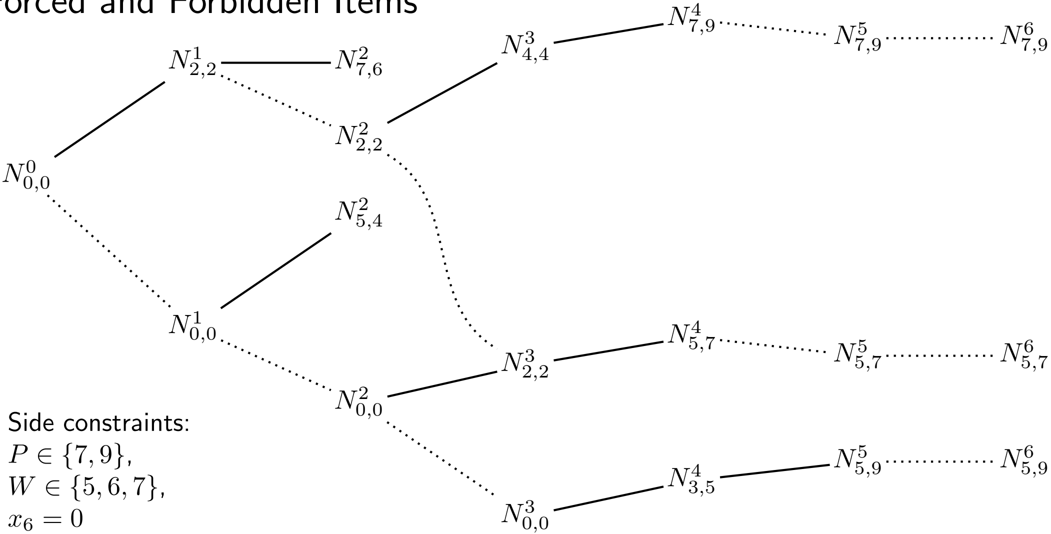
Forced and Forbidden Items



Side constraints:
$P \in \{7, 9\}$,
$W \in \{5, 6, 7\}$,
$x_6 = 0$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2$ $\boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$ $\boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$ $\boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5$ $\boldsymbol{w}_5{=}2, \boldsymbol{p}_5{=}4$ $\boldsymbol{w}_6{=}3, \boldsymbol{p}_6{=}3$

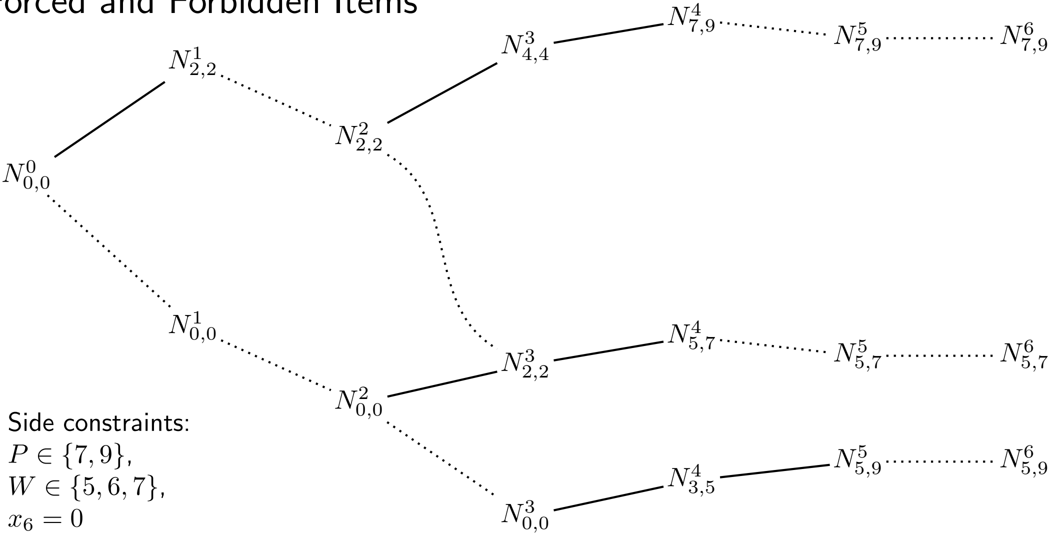
Forced and Forbidden Items



Side constraints:
$P \in \{7, 9\}$,
$W \in \{5, 6, 7\}$,
$x_6 = 0$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4 \quad \boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2 \quad \boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5 \quad \boldsymbol{w}_5{=}2, \boldsymbol{p}_5{=}4 \quad \boldsymbol{w}_6{=}3, \boldsymbol{p}_6{=}3$

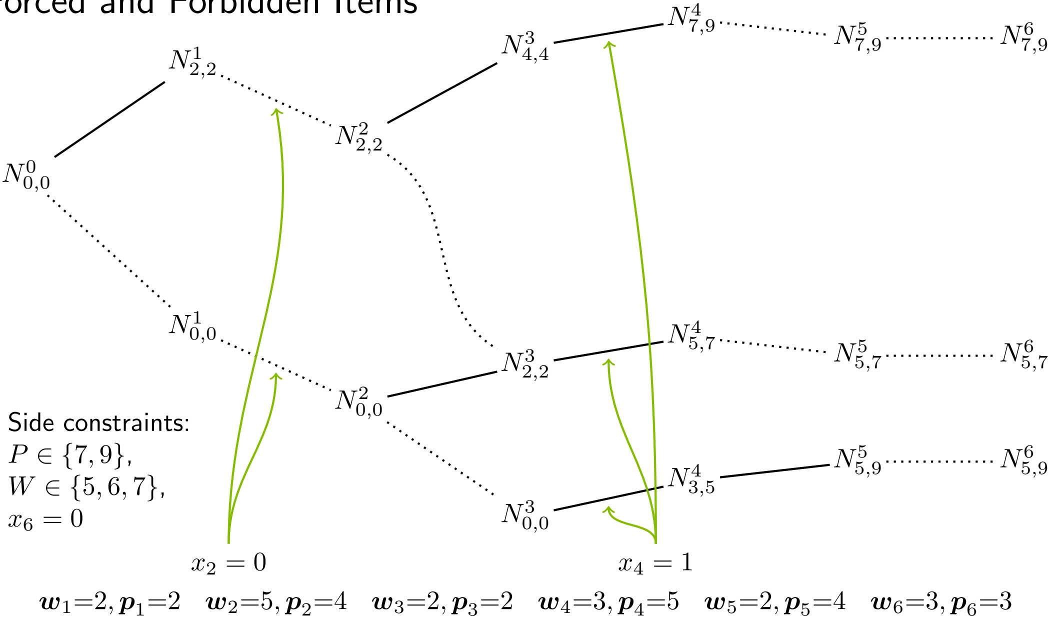
Forced and Forbidden Items



Side constraints:
$P \in \{7, 9\}$,
$W \in \{5, 6, 7\}$,
$x_6 = 0$

$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2$ $\boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$ $\boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$ $\boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5$ $\boldsymbol{w}_5{=}2, \boldsymbol{p}_5{=}4$ $\boldsymbol{w}_6{=}3, \boldsymbol{p}_6{=}3$

Forced and Forbidden Items



Solving and Justification Languages are Different

- Traditional SAT view: solvers are searching for proofs, and there is a proof system that is "natural" for the description of the problem.

Solving and Justification Languages are Different

- Traditional SAT view: solvers are searching for proofs, and there is a proof system that is "natural" for the description of the problem.
  - A huge coincidence due to CDCL and the proof that SAT solvers can't count.
  - Not really what's happening: there are resolution proofs that SAT solvers can't find.

Solving and Justification Languages are Different

- Traditional SAT view: solvers are searching for proofs, and there is a proof system that is "natural" for the description of the problem.
  - A huge coincidence due to CDCL and the proof that SAT solvers can't count.
  - Not really what's happening: there are resolution proofs that SAT solvers can't find.
- We need a stronger input language and proof system to justify modern SAT solving techniques.
- A simpler input language and proof system is fine for justifying modern CP and graph solving techniques.

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk

University of Glasgow

Royal Academy of Engineering