Proof Logging for Dynamic Programming Algorithms

Ciaran McCreesh

In collaboration with Matthew McIlree; Emir Demirović and Konstantin Sidorov (TU Delft);
and Jakob Nordström and Andy Oertel (Lund University / University of Copenhagen)

University of Glasgow

Royal Academy of Engineering

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

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

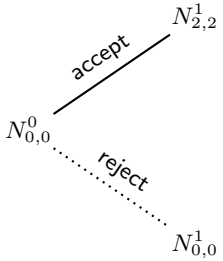
with $p \geq p'$ and $w \leq w'$ then we need only consider the former.

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

Viewing Dynamic Programming as a Decision Diagram

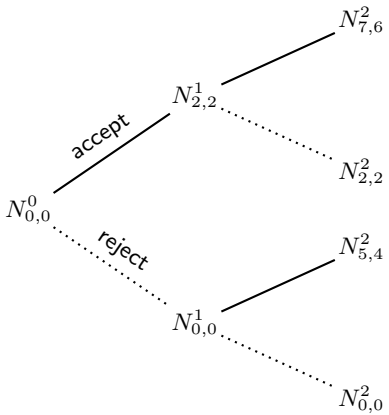
$N^0_{0,0}$

Viewing Dynamic Programming as a Decision Diagram



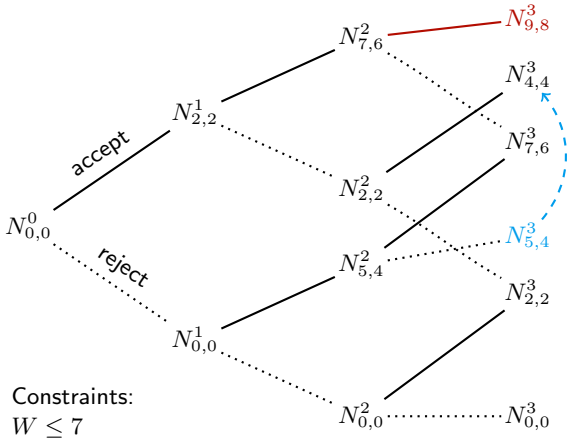
$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2$

Viewing Dynamic Programming as a Decision Diagram



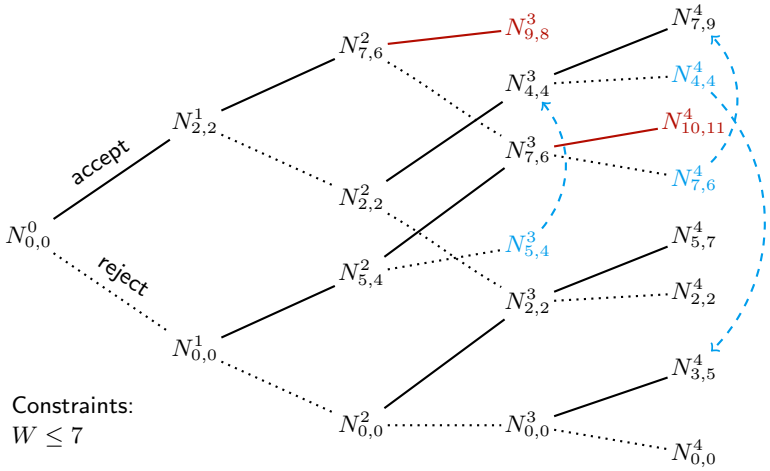
$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4$

Viewing Dynamic Programming as a Decision Diagram



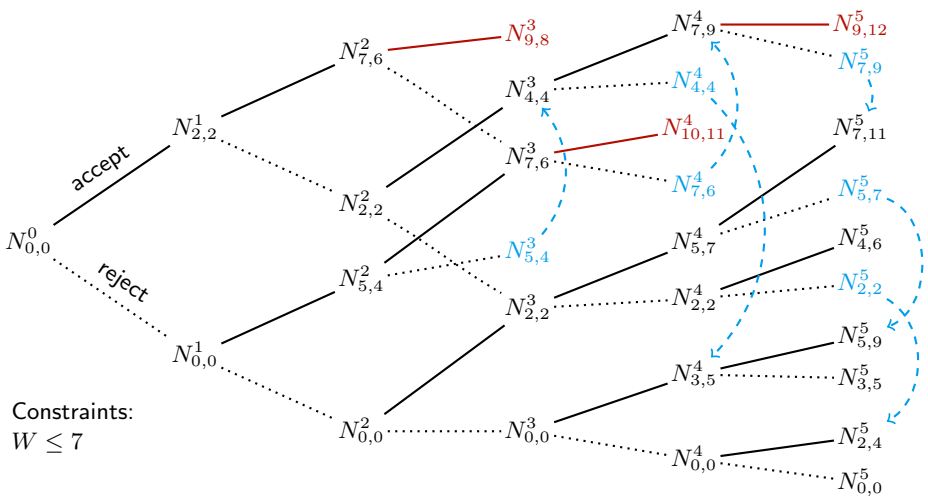
$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4 \quad \boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2$

Viewing Dynamic Programming as a Decision Diagram



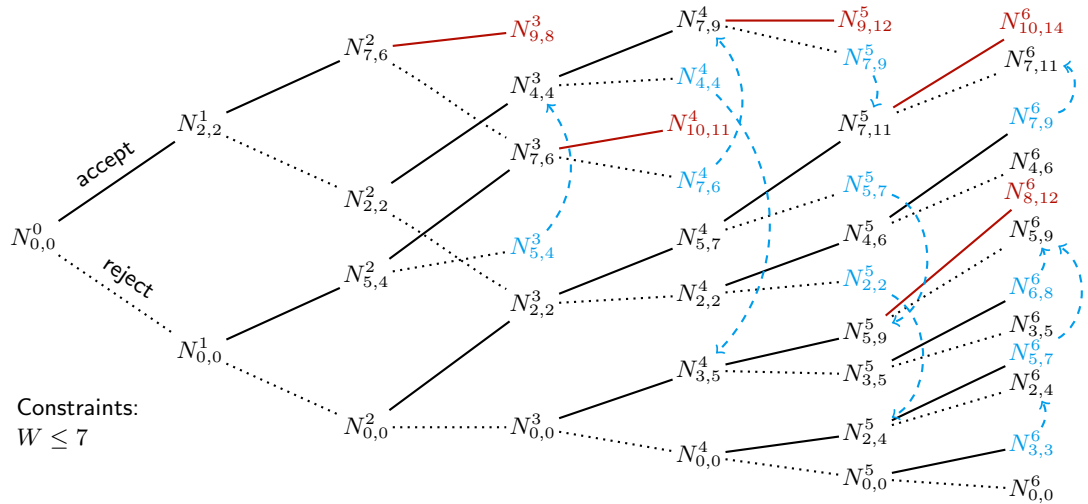
$\boldsymbol{w}_1{=}2, \boldsymbol{p}_1{=}2 \quad \boldsymbol{w}_2{=}5, \boldsymbol{p}_2{=}4 \quad \boldsymbol{w}_3{=}2, \boldsymbol{p}_3{=}2 \quad \boldsymbol{w}_4{=}3, \boldsymbol{p}_4{=}5$

Viewing Dynamic Programming as a Decision Diagram



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

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?

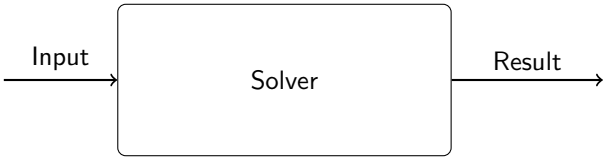
Is This Correct?

Do you trust the theory?

Do you trust your PhD student to implement it correctly?

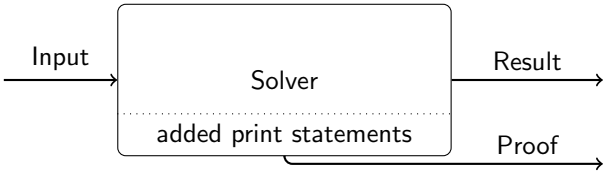
Would you trust this inside a larger solver, where side constraints could apply?

Proof Logging



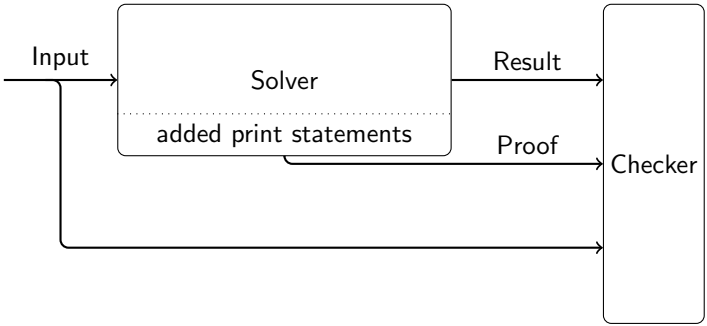
1. Run solver on problem input.

Proof Logging



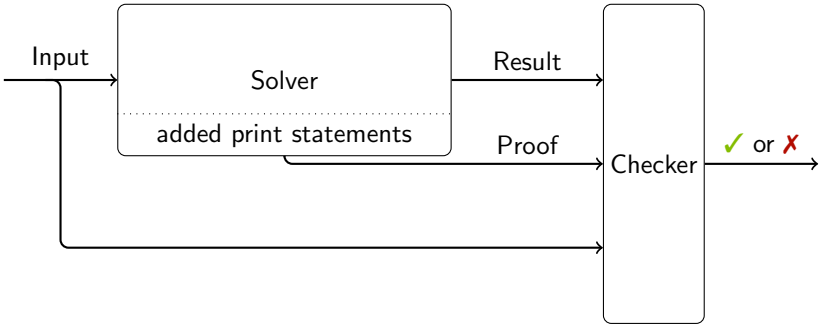
1. Run solver on problem input.
2. Solver also prints out a proof as part of its output.

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.

The VeriPB Proof System

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

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

Pseudo-Boolean Problems

Variables $x_i \in \{0, 1\}$

Literals $\{x_i, \overline{x}_i\}$

$\overline{x}_i$ defined to be $1 - x_i$, means "not x"

Constraints $\sum_i a_i x_i \bowtie A$ for $a_i, A \in \mathbb{Z}$ and $\bowtie \in \{\leq, \geq\}$

Can rewrite into normalised form with $a_i, A \in \mathbb{N}$ and $\bowtie = \geq$

Objective Maximise $\sum_i a_i x_i$

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.

Implications as Pseudo-Boolean Constraints

Can express implications

$$y \Rightarrow 3x_1 + 2x_2 + x_3 + x_4 \geq 3$$

as

$$3\overline{y} + 3x_1 + 2x_2 + x_3 + x_4 \geq 3$$

We can also do $\wedge_i y_i \Rightarrow C$ and $y \Leftarrow C$.

What is a Proof?

For satisfiable problem instances: just a witness.

What is a Proof?

For satisfiable problem instances: just a witness.

For unsatisfiable problem instances:

- Start by assuming the 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 $0 \geq 1$.

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

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

Saturation
assumes normalised form

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

CNF, Resolution, and Implications

Pseudo-Boolean constraints are a superset of CNF clauses:

$$a \vee \overline{b} \vee c \qquad \Leftrightarrow \qquad a + \overline{b} + c \geq 1$$

CNF proofs can be done using the resolution rule:

Resolution

$$\frac{a_1 \vee a_2 \vee \cdots \vee c \qquad \overline{c} \vee b_1 \vee b_2 \vee \cdots}{a_1 \vee a_2 \vee \cdots b_1 \vee b_2 \vee \cdots}$$

We can simulate this in cutting planes using addition then saturation.

CNF, Resolution, and Implications

Pseudo-Boolean constraints are a superset of CNF clauses:

$$a \vee \overline{b} \vee c \qquad \Leftrightarrow \qquad a + \overline{b} + c \geq 1$$

CNF proofs can be done using the resolution rule:

Resolution

$$\frac{a_1 \vee a_2 \vee \cdots \vee c \qquad \overline{c} \vee b_1 \vee b_2 \vee \cdots}{a_1 \vee a_2 \vee \cdots b_1 \vee b_2 \vee \cdots}$$

We can simulate this in cutting planes using addition then saturation.

Another way of viewing this is:

Resolution

$$\frac{(\overline{a}_1 \wedge \overline{a}_2 \wedge \cdots) \Rightarrow c \qquad c \Rightarrow (b_1 \vee b_2 \vee \cdots)}{(\overline{a}_1 \wedge \overline{a}_2 \wedge \cdots) \Rightarrow (b_1 \vee b_2 \vee \cdots)}$$

Extended Cutting Planes Proofs

We'll also need a way of introducing a new variable that are true if and only if a certain constraint holds (*reification*).

Extension

y a fresh variable
C any constraint

$$\frac{}{y \Rightarrow C \qquad y \Leftarrow C}$$

In reality: we have a more general extension rule, but we don't need it today.

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.

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 Constraints

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

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

Unit Propagation for 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 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 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*.

Reverse Unit Propagation

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

RUP

C any constraint

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

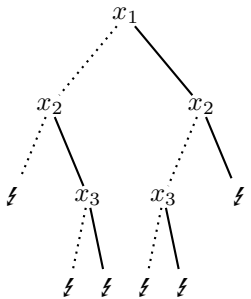
Can rewrite any RUP step as a sequence of cutting planes steps, but often writing RUP proofs is much more convenient.

RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$

RUP as a Proof Framework for Backtracking Search

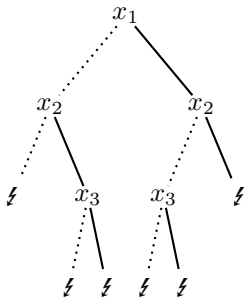
$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$



RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$

rup $x_1 + x_2 \geq 1$

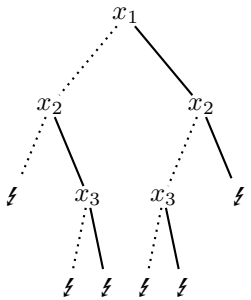


RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$

rup $x_1 + x_2 \geq 1$

rup $x_1 + \overline{x}_2 + x_3 \geq 1$



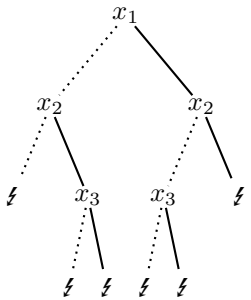
RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$

$\text{rup}\ \ x_1 + x_2 \geq 1$

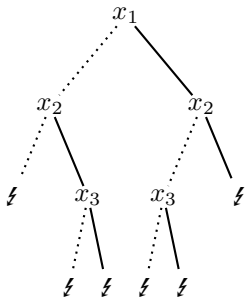
$\text{rup}\ \ x_1 + \overline{x}_2 + x_3 \geq 1$

$\text{rup}\ \ x_1 + \overline{x}_2 + \overline{x}_3 \geq 1$



RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$



rup $x_1 + x_2 \geq 1$

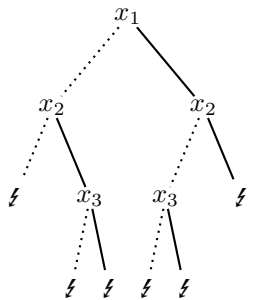
rup $x_1 + \overline{x}_2 + x_3 \geq 1$

rup $x_1 + \overline{x}_2 + \overline{x}_3 \geq 1$

rup $x_1 + \overline{x}_2 \geq 1$

RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$



rup $x_1 + x_2 \geq 1$

rup $x_1 + \overline{x}_2 + x_3 \geq 1$

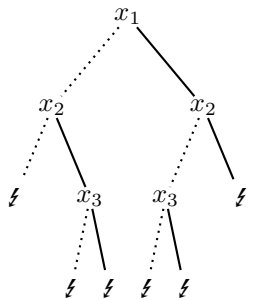
rup $x_1 + \overline{x}_2 + \overline{x}_3 \geq 1$

rup $x_1 + \overline{x}_2 \geq 1$

rup $x_1 \geq 1$

RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$



rup $x_1 + x_2 \geq 1$

rup $x_1 + \overline{x}_2 + x_3 \geq 1$

rup $x_1 + \overline{x}_2 + \overline{x}_3 \geq 1$

rup $x_1 + \overline{x}_2 \geq 1$

rup $x_1 \geq 1$

rup $\overline{x}_1 + x_2 + x_3 \geq 1$

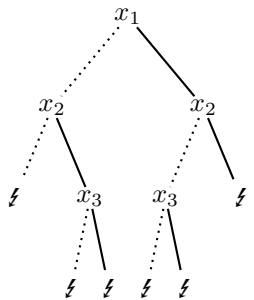
rup $\overline{x}_1 + x_2 + \overline{x}_3 \geq 1$

rup $\overline{x}_1 + x_2 \geq 1$

rup $\overline{x}_1 \geq 1$

RUP as a Proof Framework for Backtracking Search

$$2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 2x_6 \leq 5$$
$$3x_1 + 3x_2 + 4x_3 + x_4 + 3x_5 + x_6 \geq 8$$



rup $x_1 + x_2 \geq 1$

rup $x_1 + \overline{x}_2 + x_3 \geq 1$

rup $x_1 + \overline{x}_2 + \overline{x}_3 \geq 1$

rup $x_1 + \overline{x}_2 \geq 1$

rup $x_1 \geq 1$

rup $\overline{x}_1 + x_2 + x_3 \geq 1$

rup $\overline{x}_1 + x_2 + \overline{x}_3 \geq 1$

rup $\overline{x}_1 + x_2 \geq 1$

rup $\overline{x}_1 \geq 1$

rup $0 \geq 1$

Interleaving Propagations

- Many facts discovered by propagation algorithms do *not* follow by RUP.
- Interleave additional extra RUP steps for propagated facts:
  - Table constraints.
  - Circuit constraints (simple propagators).
- Interleave explicit extended cutting planes proofs:
  - All-different Hall sets.
  - Linear inequalities with non-0/1 variables.
  - Circuit constraints (cleverer propagators).
- Enough to do subgraph isomorphism, constraint programming, ...

Proofs Under Implication

Theorem

Let's say we have a collection of constraints $\mathcal{C}$ and know how to get a proof deriving D.

Suppose now we replace one or more constraints C_i from $\mathcal{C}$ with implications $\wedge G_i \Rightarrow C$.

Then we know how to get a proof deriving $\wedge \cup_i G_i \Rightarrow D$.

Handwavy proof sketch.

For RUP, add in the implications and it "just works".

For cutting planes, run the same proof, but saturate between each step. Use literal axioms at the end to re-introduce any assumptions that disappear from $\wedge \cup_i G_i$, and to fix up coefficients.

For extension rules, keep them as-is. □

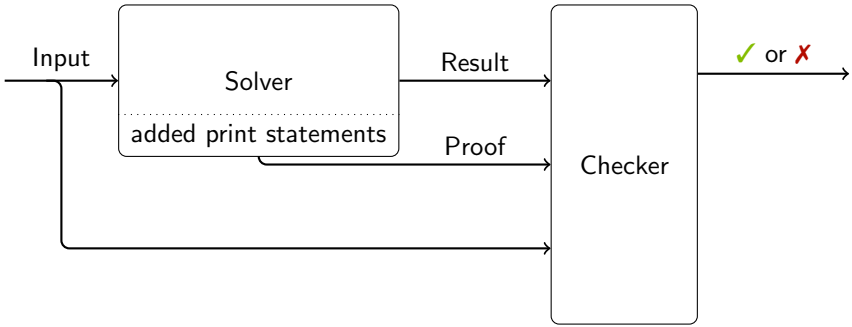
Branch and Bound?

- Log solutions as we find them, get a solution-improving constraint.
- So the semantics are "this instance is unsatisfiable if the objective has to beat whatever the best solution I found is".

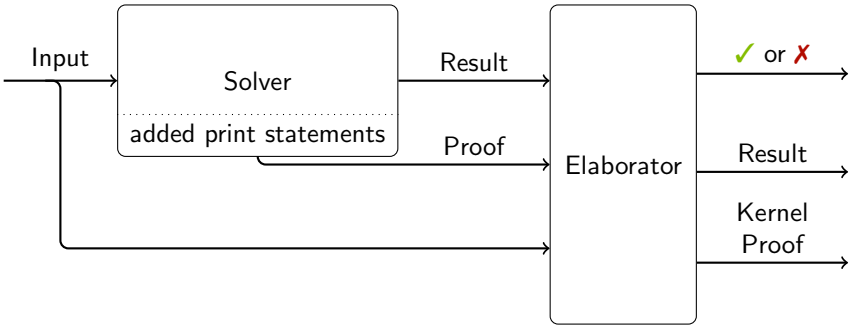
Branch and Bound?

- Log solutions as we find them, get a solution-improving constraint.
- So the semantics are "this instance is unsatisfiable if the objective has to beat whatever the best solution I found is".
- Have to replace "equisatisfiable" with "equioptimal"!

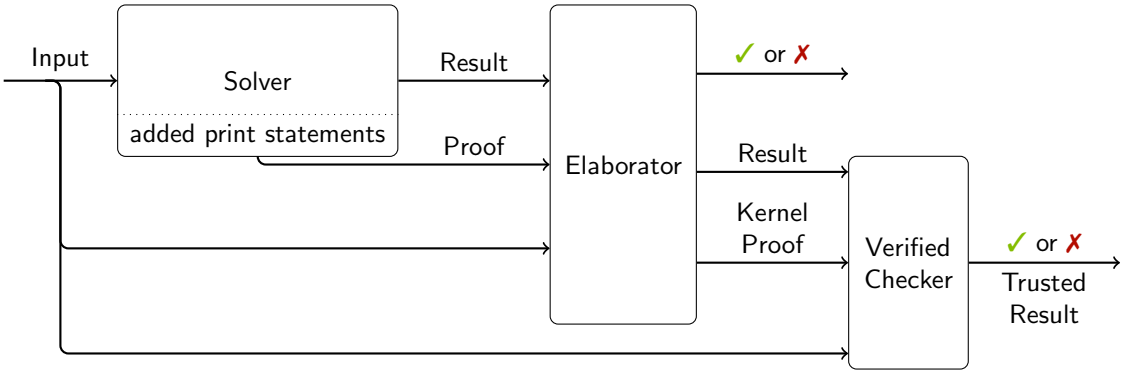
Should You Trust the Checker?



Should You Trust the Checker?



Should You Trust the Checker?



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

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

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

"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.

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.

A Change of States

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

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

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

$$N_{w,p}^\ell \Leftrightarrow W{\uparrow}_w^\ell + W{\downarrow}_w^\ell + P{\uparrow}_p^\ell + P{\downarrow}_p^\ell \geq 4$$

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

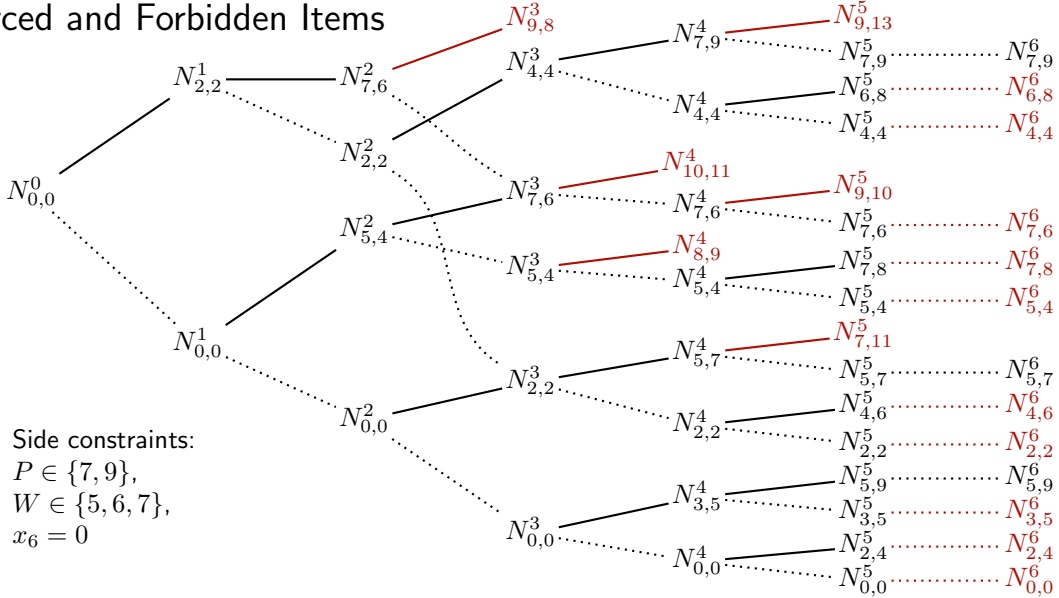
We can read off all possible values for P and W from the final layer.

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?

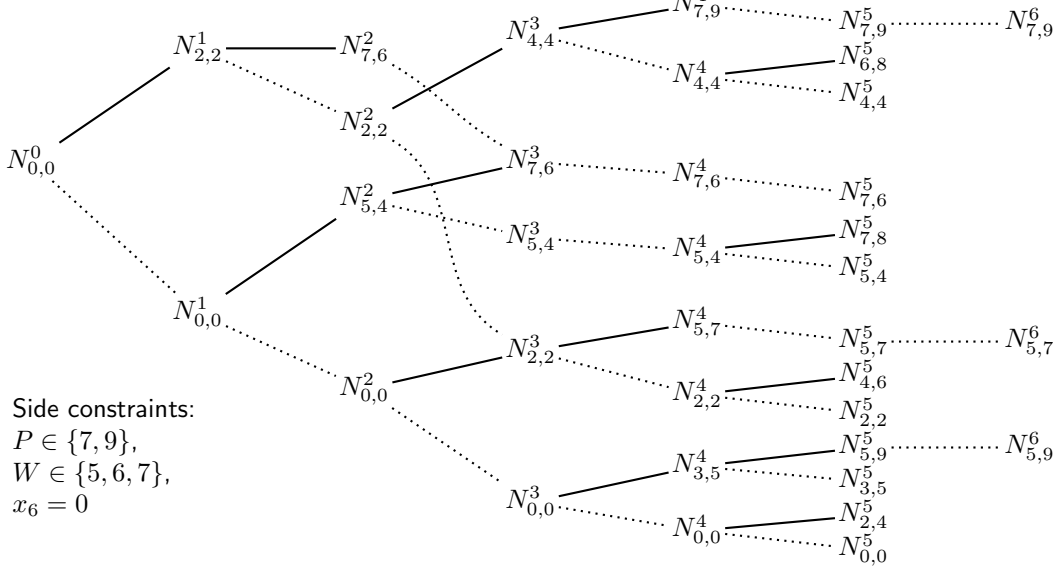
Forced and Forbidden Items



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

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

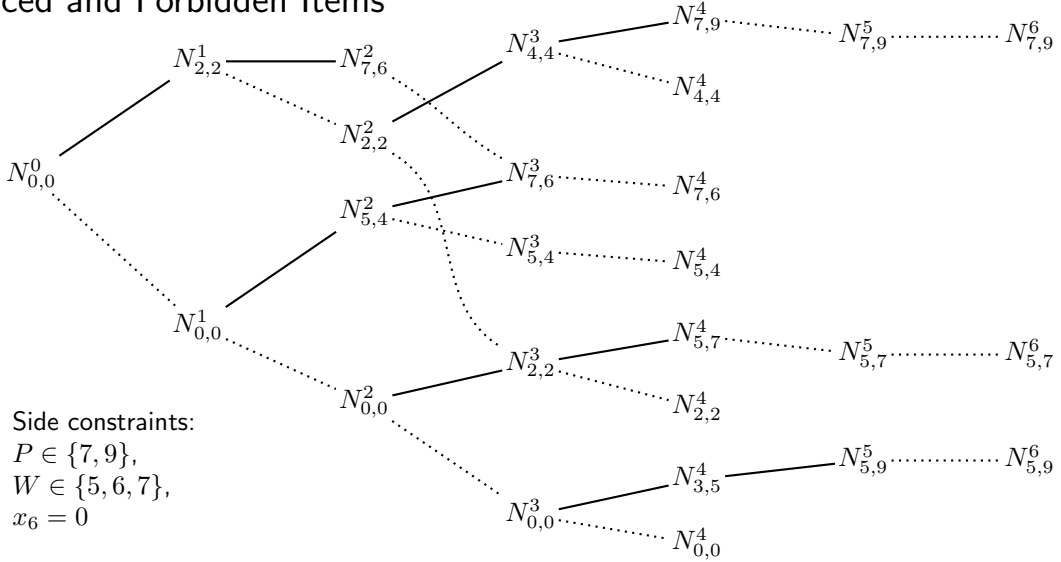
Forced and Forbidden Items



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

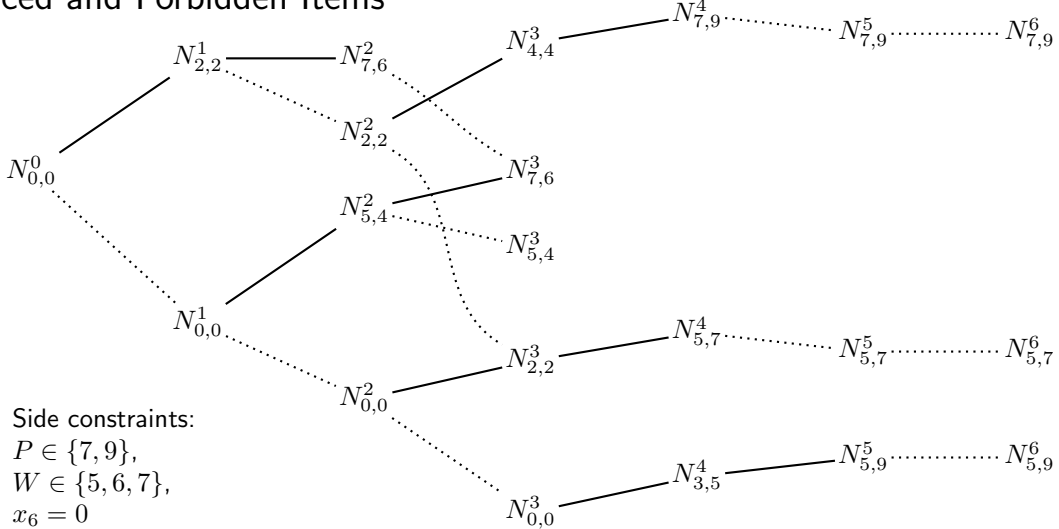
Forced and Forbidden Items



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

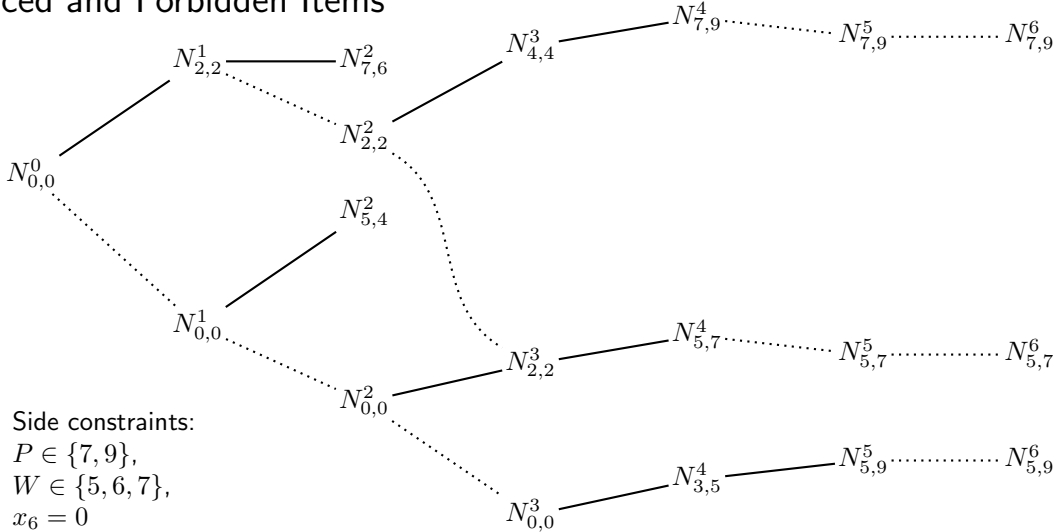
Forced and Forbidden Items



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

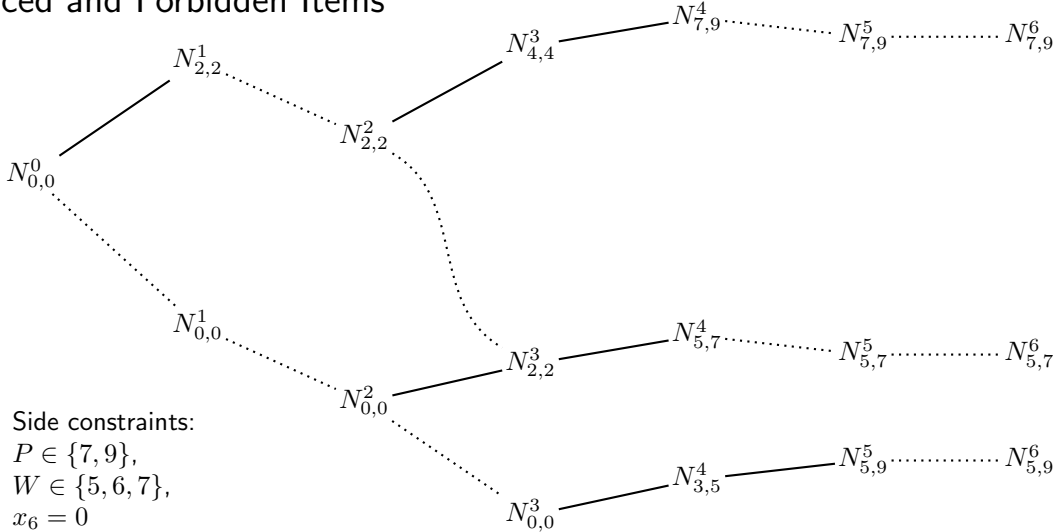
Forced and Forbidden Items



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

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

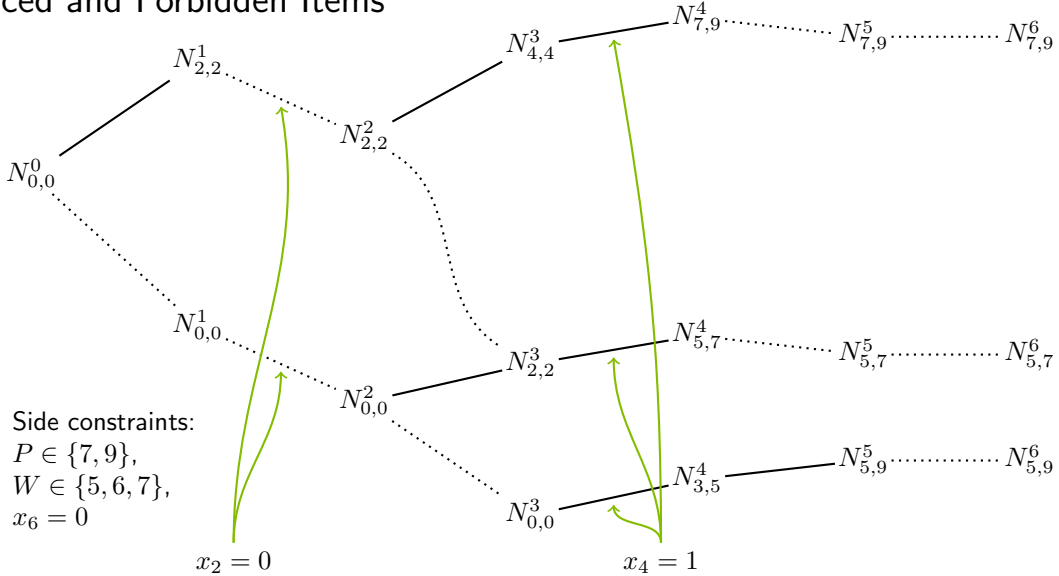
Forced and Forbidden Items



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



$\boldsymbol{w}_1$=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

It Works

- Standalone implementation.
- Another independent implementation at TU Delft.
- Also implemented as a propagator inside the Glasgow Constraint Solver.

So What?

- This is a general framework, not specific to knapsack.
  - Implications between extension variables describe *structure*.
  - Key proof technique is that we can wrap and unwrap extension variables and implications.
- We can do dynamic programming / decision diagram propagators for CP.
- All of this is in a unified proof setting.
  - In particular, nothing in the proof system knows about "search" or "states".
  - We can reason about complex states using the full power of cutting planes.
  - We could do symmetry reasoning (two identical items) and dominance reasoning (one item better than another) too.
- Might help us start to trust some of these clever hybrid algorithms for graph parameters like treewidth?
- Does this give us a better way of explaining and teaching dynamic programming?

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk

University of Glasgow

Royal Academy of Engineering