Subsumption, Recursive Split and Greedy Pebbling

Andreas Fellner

Joseph Boudou, Bruno Woltzenlogel Paleo

Third Workshop of the Amadeus Project on Proof Compression
Monday 16th September, 2013

Overview

Propopositional Resolution Calculus

Literal

- Variable v or negated variable $\bar{v}$

Clause

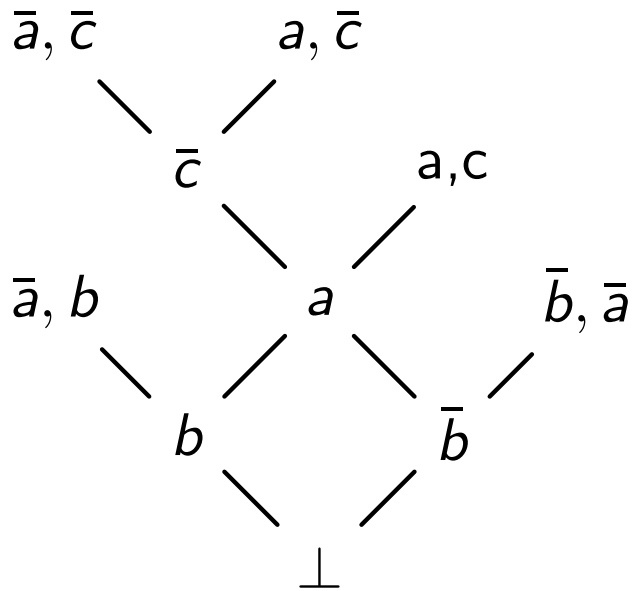
- Disjunction of literals
- Represented as a set

Resolution rule

$$\frac{\Gamma \vee v \qquad \bar{v} \vee \Delta}{\Gamma \vee \Delta}\, v$$

Proof as a directed acyclic graph (DAG)

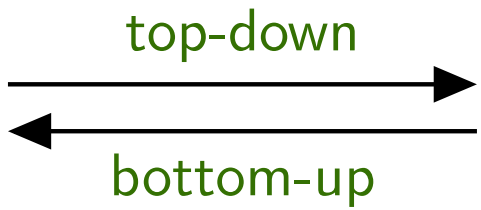
- node, conclusion, pivot, premise, child



Proof as sequence

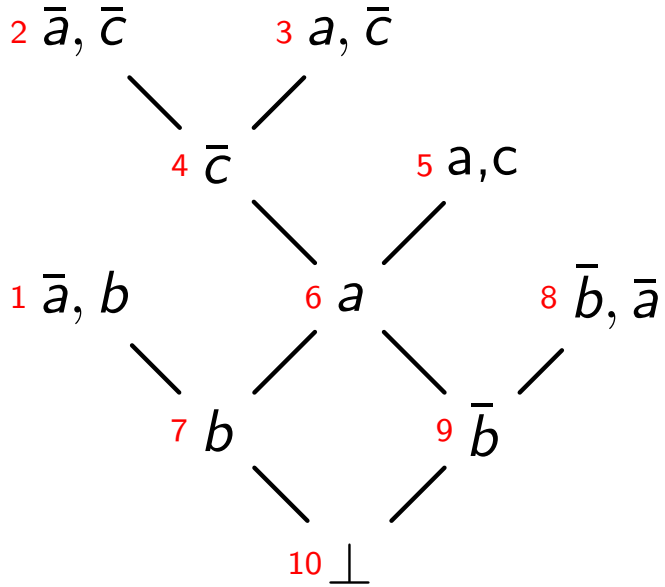
- Use topological order

$\bar{a}, b$ $\bar{a}, \bar{c}$ $a, \bar{c}$ $\bar{c}$ a, c a b $\bar{b}, \bar{a}$ $\bar{b}$ $\bot$



Proof as a directed acyclic graph (DAG)

- node, conclusion, pivot, premise, child



Proof as sequence

- Use topological order

$\bar{a}, b$ [1] $\bar{a}, \bar{c}$ [2] $a, \bar{c}$ [3] $\bar{c}$ [4] a, c [5] a [6] b [7] $\bar{b}, \bar{a}$ [8] $\bar{b}$ [9] $\bot$ [10]

top-down →

← bottom-up

Subsumption for Proof Compression

Idea

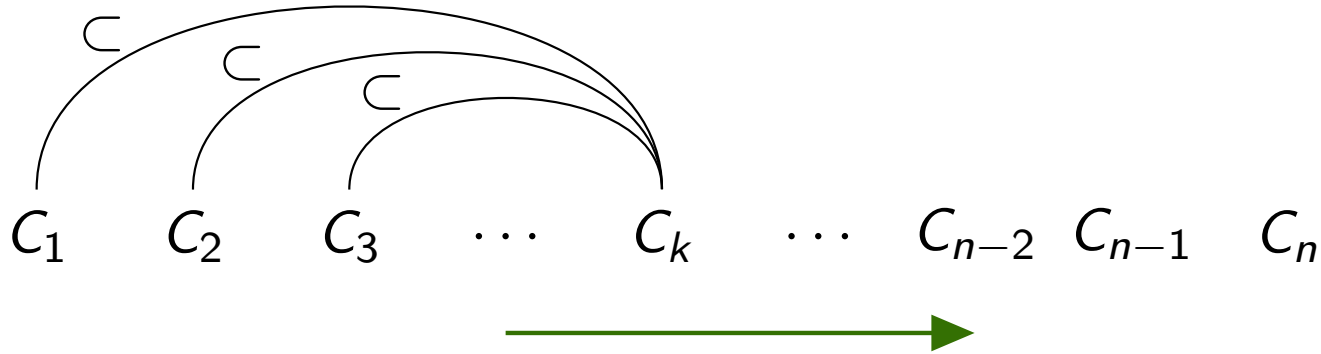
- Subsumption
  - C_1 subsumes C_2 iff $C_1 \subset C_2$
- Replace subsumed clauses by their subsumers
- Fix nodes with changed premises
  - Pivot in both premises $\rightarrow$ resolve premises
  - Pivot missing in a premise $\rightarrow$ use this premise

Performance

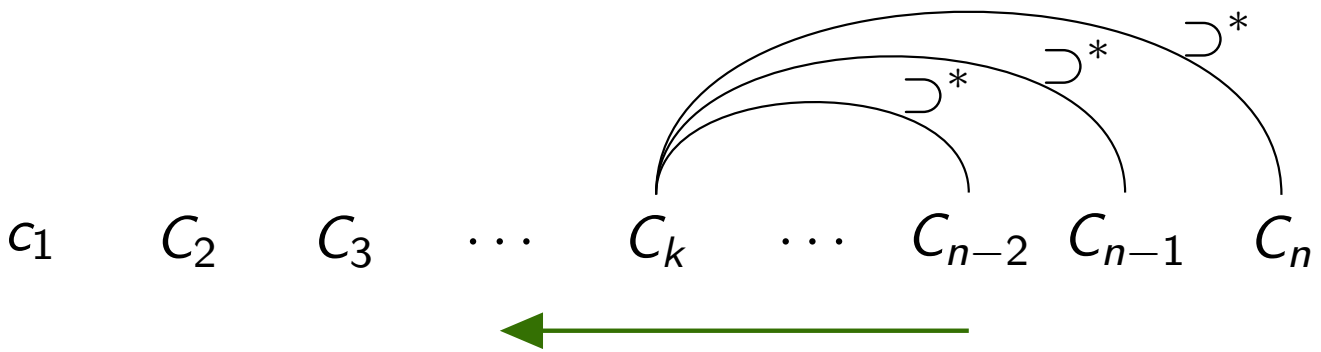
- Worst case quadratic runtime in the proof size
- Use literal-tree structure to store visited nodes and check subsumption

Top-down vs Bottom-up

Top-down

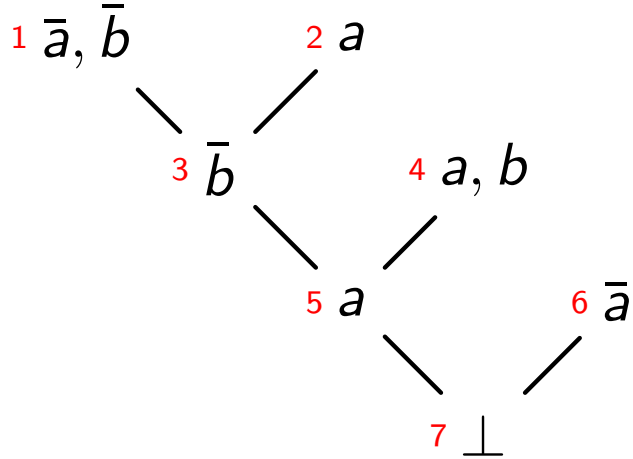


Bottom-up

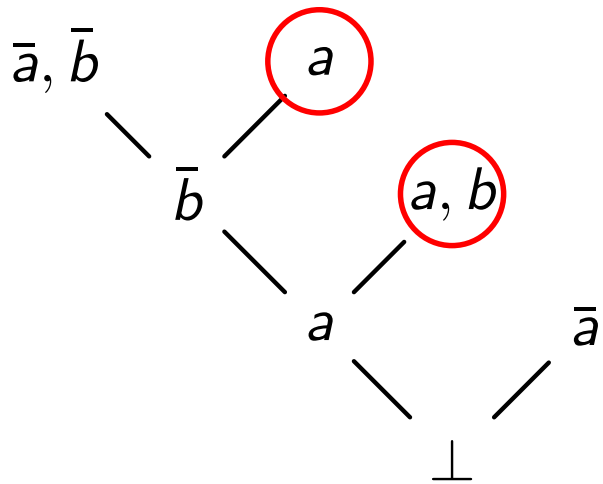


$C \subset^* D$ iff $C \subset D$ and C is not an ancestor of D

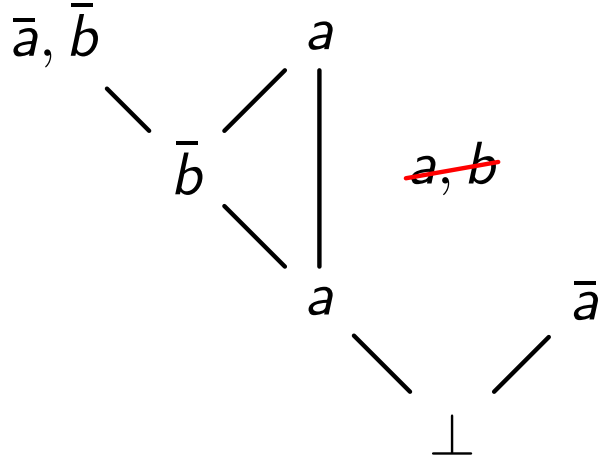
Top-down Subsumption Example



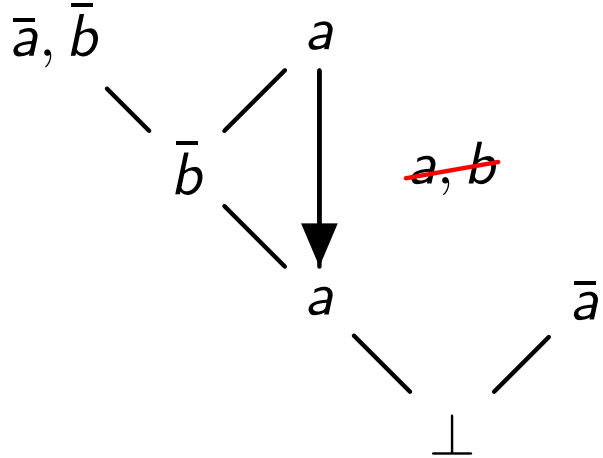
Top-down Subsumption Example



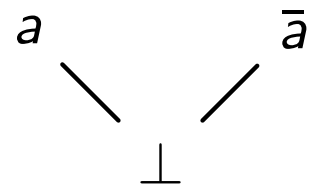
Top-down Subsumption Example



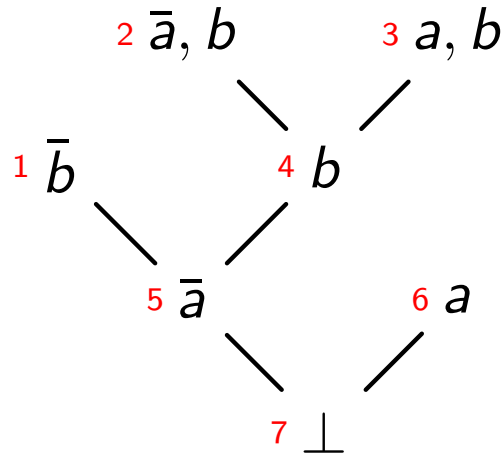
Top-down Subsumption Example



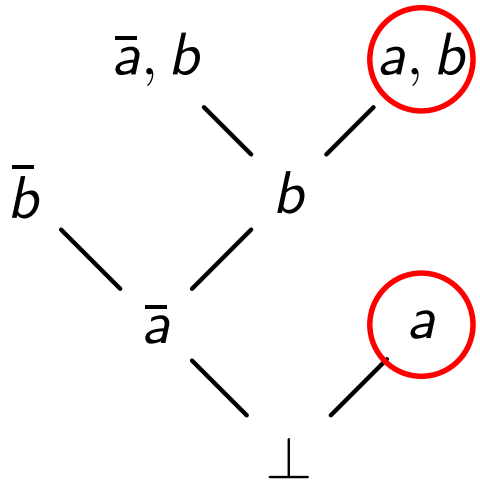
Top-down Subsumption Example



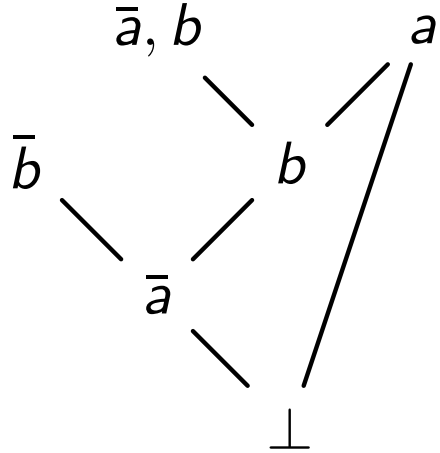
Bottom-up Subsumption Example



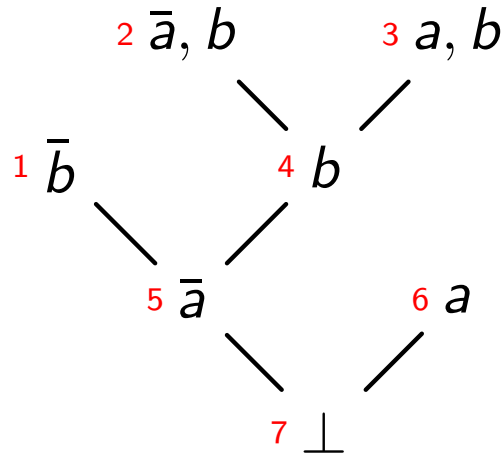
Bottom-up Subsumption Example



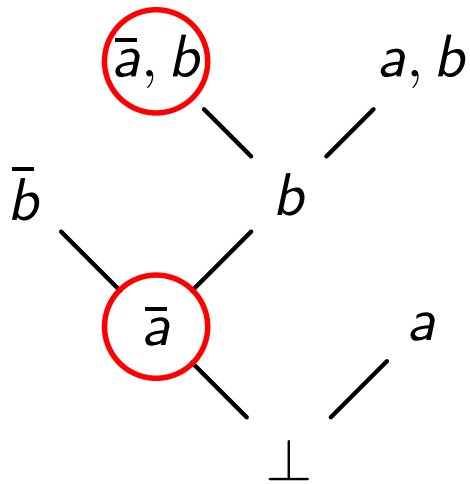
Bottom-up Subsumption Example



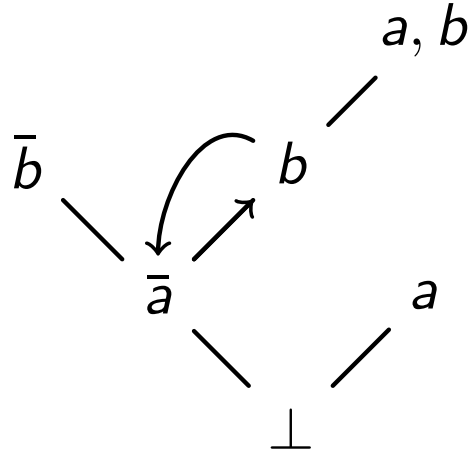
Bottom-up Subsumption Example



Bottom-up Subsumption Example



Bottom-up Subsumption Example



Problems with Bottom-up Subsumption

Performance

- Ancestor check is costly

Copy node issue

- Fixing has to be done top-down
- Nodes are replaced bottom-up

RecycleUnits

Omer Bar-Ilan et al., 2009

- IBM Haifa Research Laboratory

Special case of bottom-up subsumption

- Check only for subsuming unit clauses
- Replace subsumption check by comparing units to pivots
- Worst case quadratic runtime in the amount of unit clauses

Experiments

Setting

- 500 proofs, provided by VeriT SMT Solver
- 659,584 nodes in total
- Average 1320 nodes per proof

Results

Algorithm	Length Compression	Speed
Top-down Subsumption	3.7 %	2.3 nodes/ms
Bottom-up Subsumption	1%	0.4 nodes/ms
RecycleUnits	2%	1.0 nodes/ms
DAGify	0.6 %	7.3 nodes/ms

Split Algorithm

Author

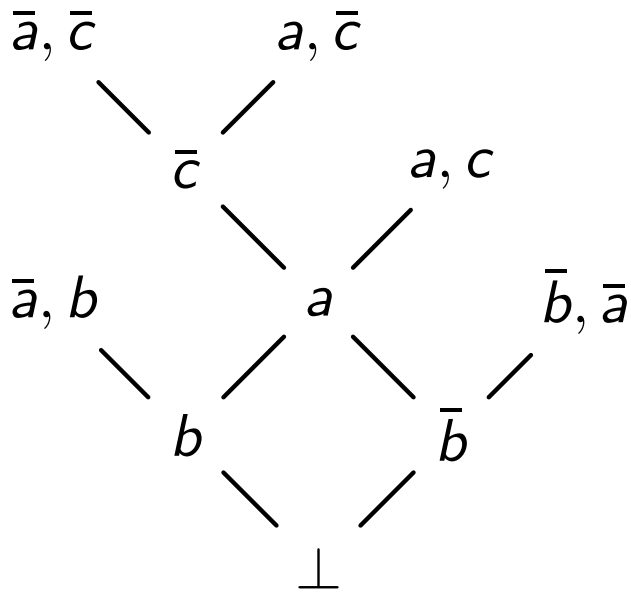
- Scott Cotton, 2010

Idea

- Let P be a proof with root clause C
- Choose a variable v, occuring in P, heuristically
- Extract proofs for $C \vee v$ and $C \vee \bar{v}$
  - By deleting positive/negative branches of nodes with pivot v
  - And fixing nodes like at subsumption algorithms
- Combine proofs by resolving roots

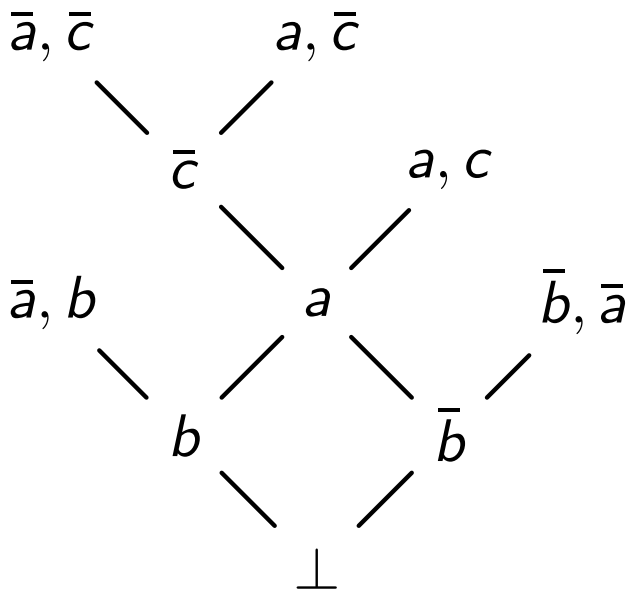
Split Algorithm Example

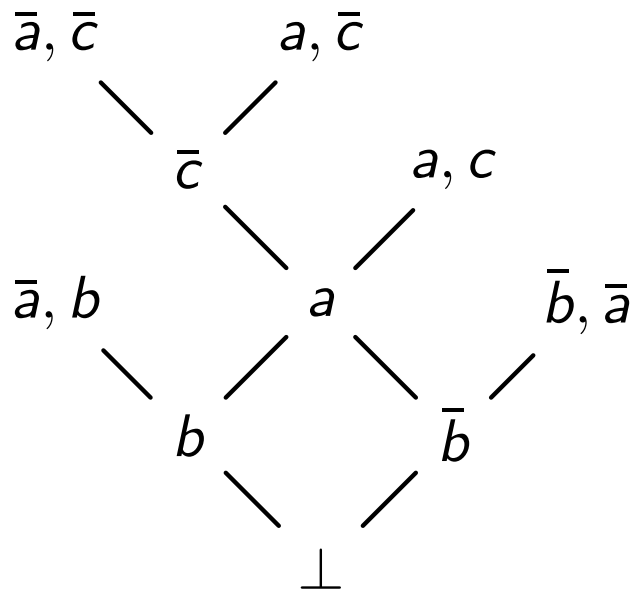
Split variable a



Split Algorithm Example

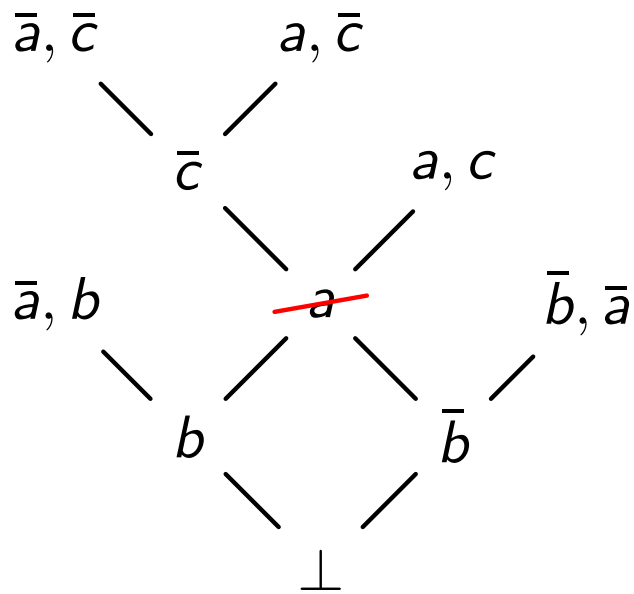
Duplicate the proof



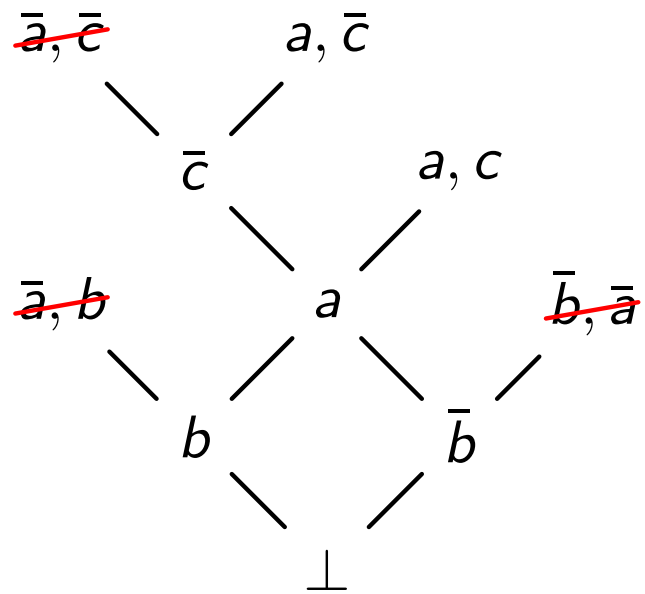


Split Algorithm Example

Delete positive branch

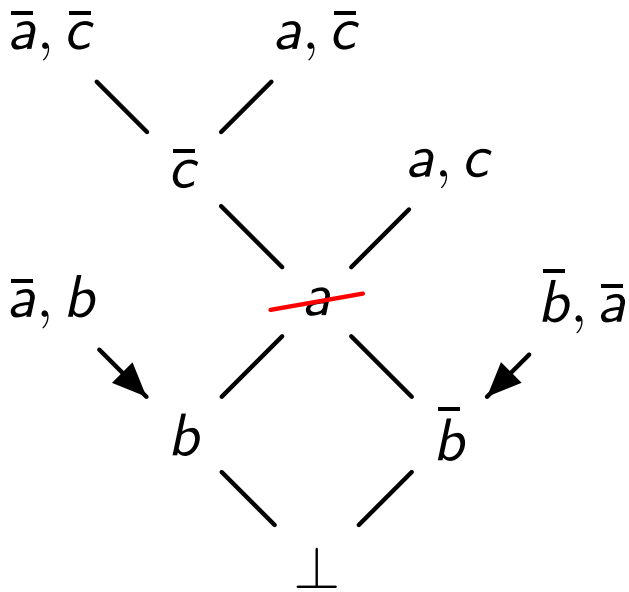


Delete negative branch

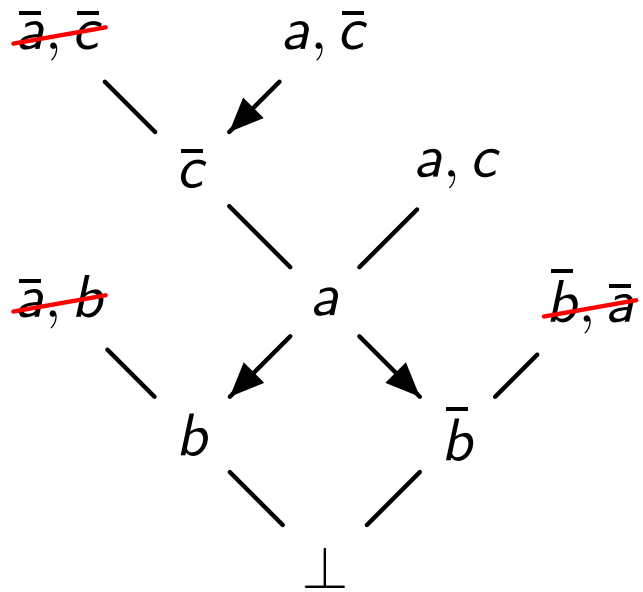


Split Algorithm Example

Fix positive branch

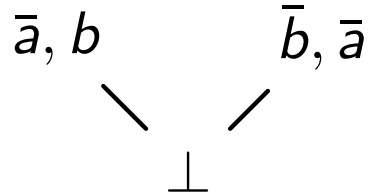


Fix negative branch

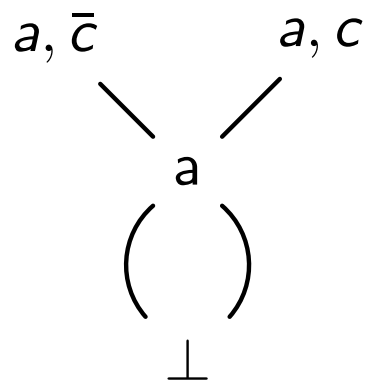


Split Algorithm Example

Fix positive branch

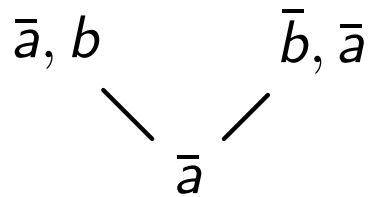


Fix negative branch

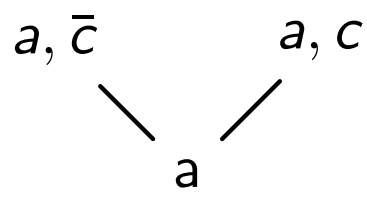


Split Algorithm Example

Fix positive branch

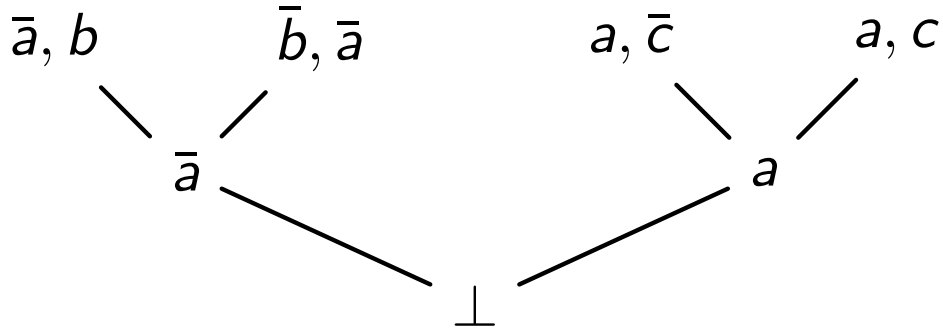


Fix negative branch



Split Algorithm Example

Combine branches by resolving



Iterative Split

Idea

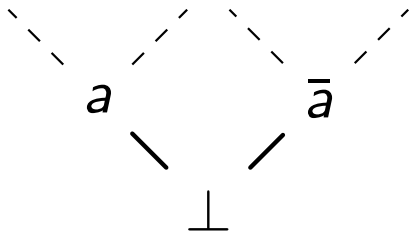
- Apply split to result of split

Issues

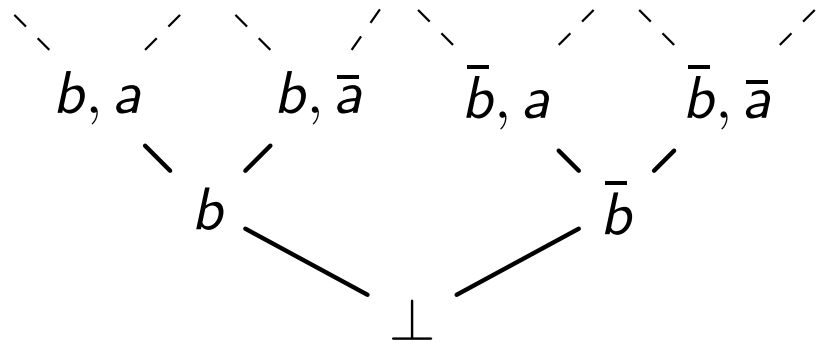
- Best variables don't end up lowest
- Same variables for positive and negative branches

Example

Split once, best variable: a



Split again, best variable: b



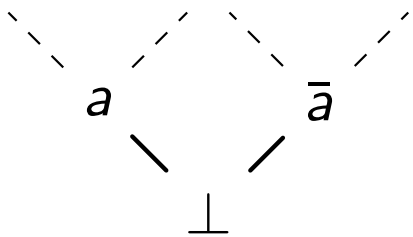
Recursive Split

Idea

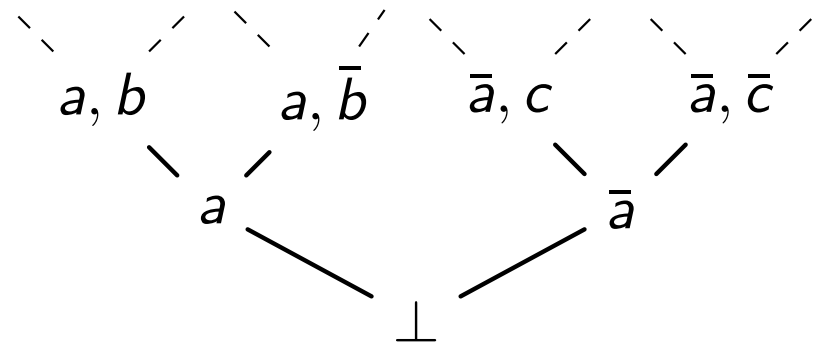
- Apply split to positive and negative branches before combining
- Use stopping criteria (depth or time)

Example

Split once, best variable: a



Split positive/negative branches, best variables: b/c;



Issue

- Equal nodes may be computed in both branches

Experiments

Results

Algorithm	Length Compression	Speed
Recursive Split (depth 3)	2.3%	3.0 nodes/ms
Recursive Split (depth 5)	2.0%	2.0 nodes/ms
Iterative Split (depth 3)	2.9%	3.2 nodes/ms
Iterative Split (depth 5)	3.4%	2.3 nodes/ms

Space Compression

Space measure

- Maximal amount of nodes that have to be kept in memory at once

Deletion information

- Extra lines in proof output
- Example: y is the last child of x
  - Read and check node x

    ...
  - Read and check node y
  - Delete node x

    ...

Interesting scenario

- Proof checker has much less memory than proof producer

Black Pebbling Game

A pebble is a small stone

Rules

- If all premises of a node p are pebbled, p may be pebbled
  - In this case, a pebble may be moved from a premise to p
- Nodes can be unpebbled at any time

Goal

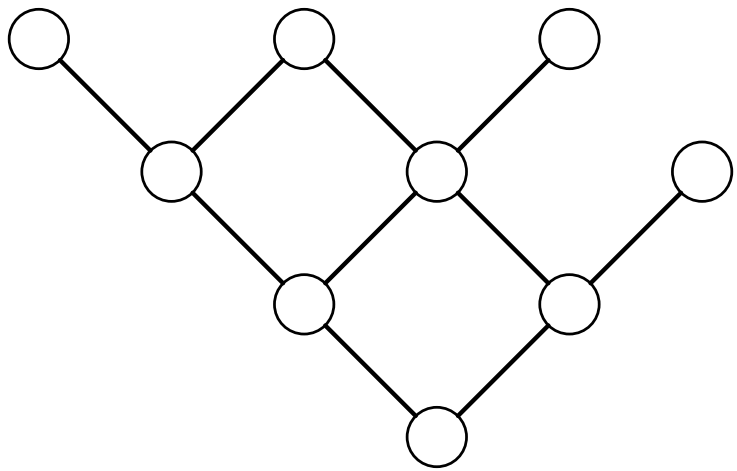
- Pebble some node v

Pebbling problem

- For a given DAG and a node v, can v be pebbled using no more than n pebbles in total?
- PSPACE-complete (John R. Gilbert et al., 1980)

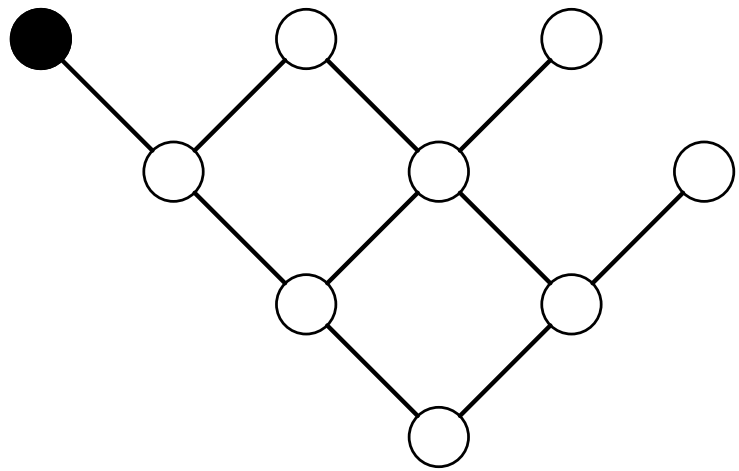
Pebbling Game Example

Maximum pebbles used: 0



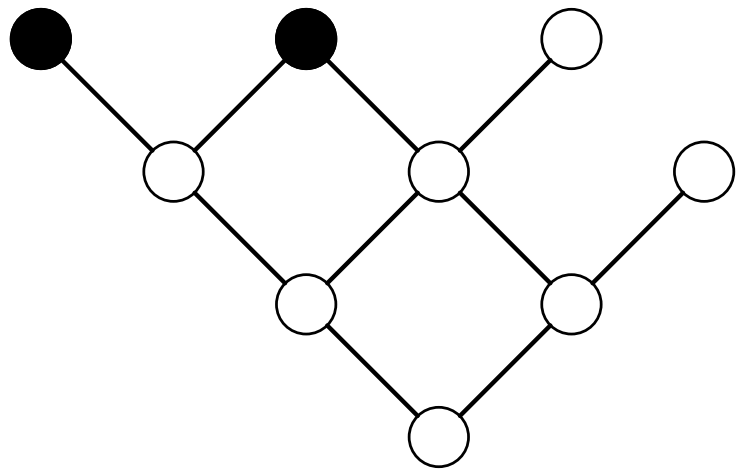
Pebbling Game Example

Maximum pebbles used: 1



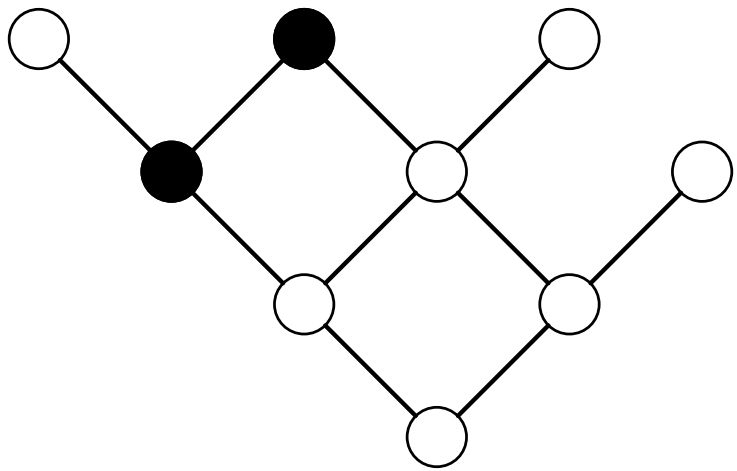
Pebbling Game Example

Maximum pebbles used: 2



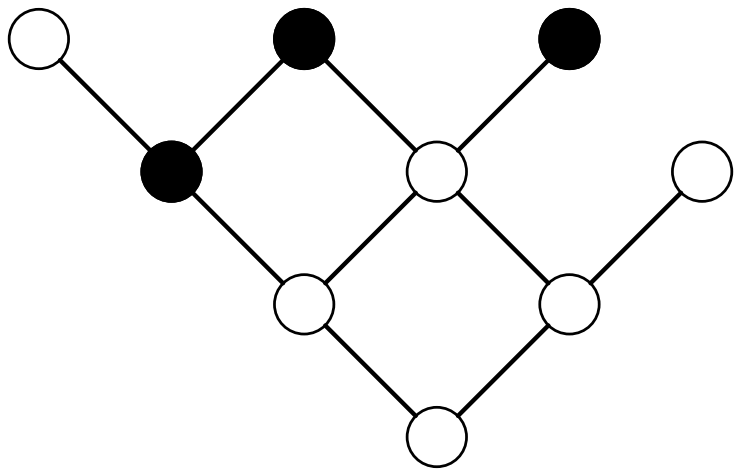
Pebbling Game Example

Maximum pebbles used: 2



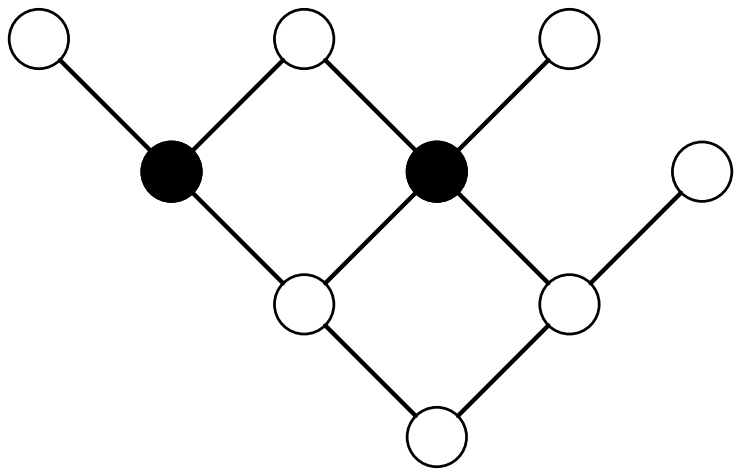
Pebbling Game Example

Maximum pebbles used: 3



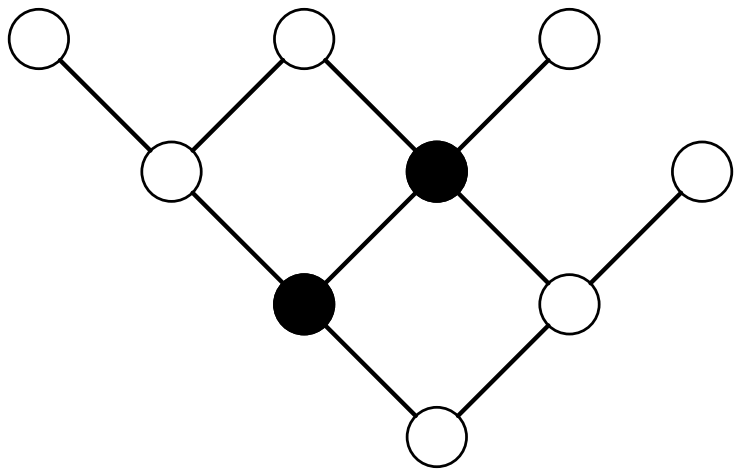
Pebbling Game Example

Maximum pebbles used: 3



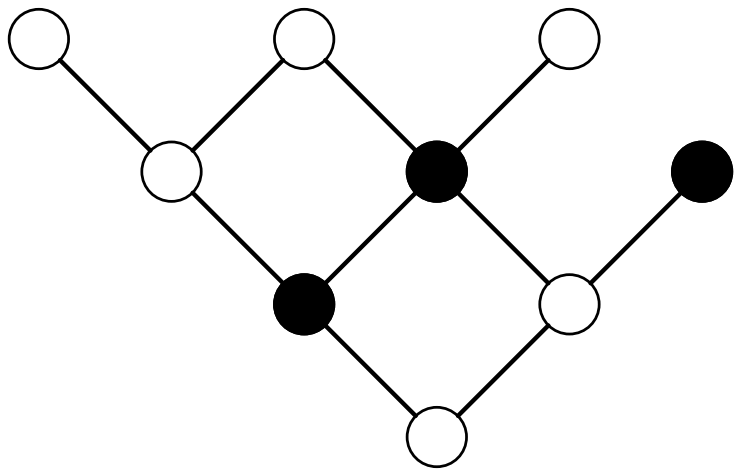
Pebbling Game Example

Maximum pebbles used: 3



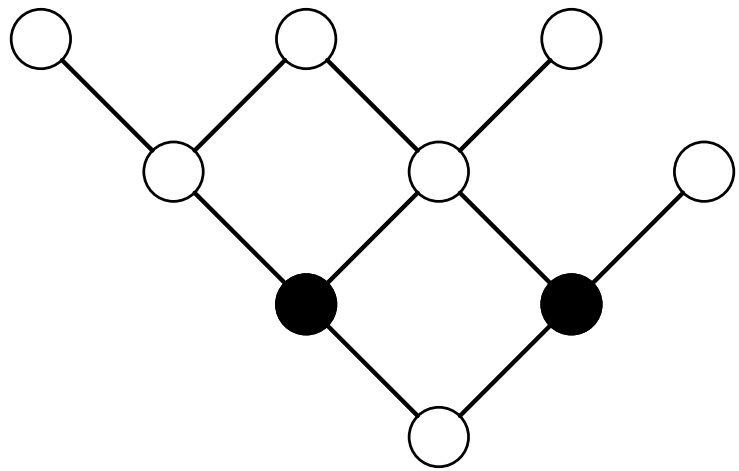
Pebbling Game Example

Maximum pebbles used: 3



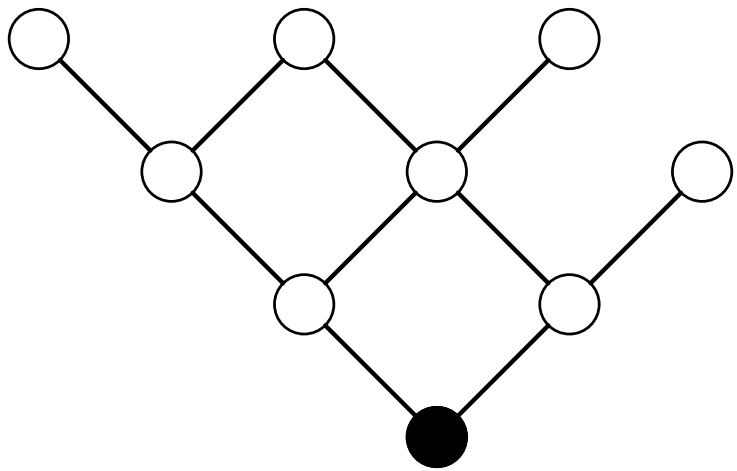
Pebbling Game Example

Maximum pebbles used: 3



Pebbling Game Example

Maximum pebbles used: 3



Greedy Pebbling

Topological Order + Deletion Information

- Correspond to a strategy for the pebbling game

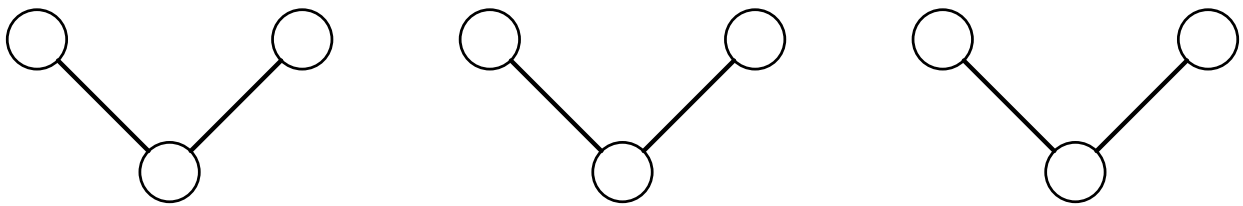
Greedy heuristics

- Find better order w.r.t. space measure
- Choose next node w.r.t.:
  - Number of pebbles that can be removed
  - Pebbled premises
  - Children with pebbled premises
  - Number of children
  - Number of premises, which the node is the last children of
  - ...

Top-down vs Bottom-up

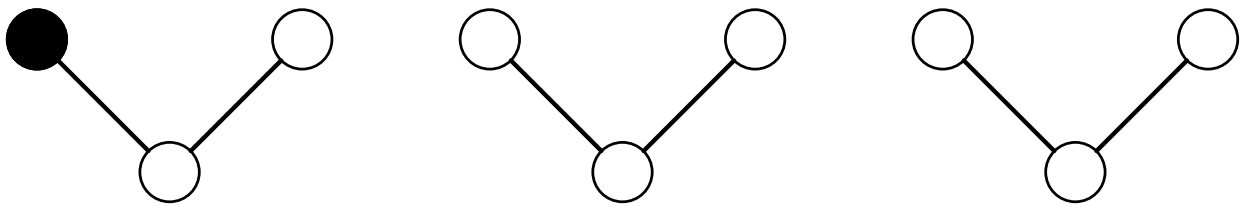
Greedy Top-down Pebbling

Maximum pebbles used: 0



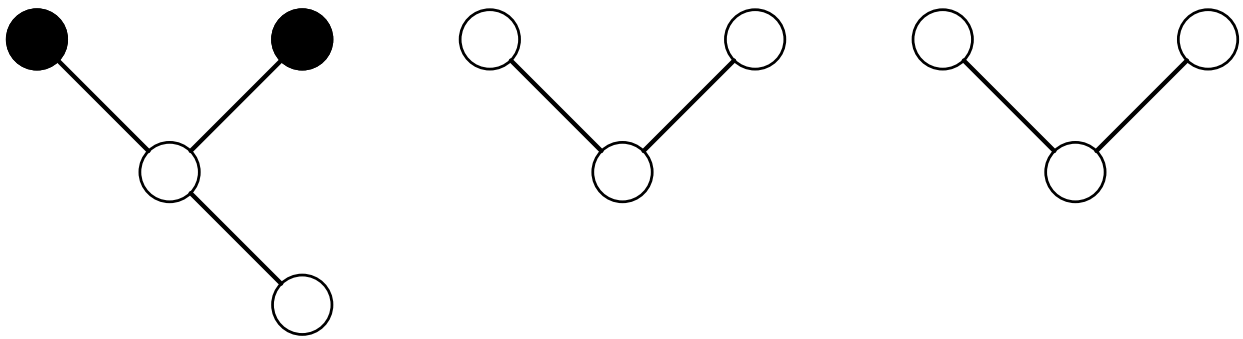
Greedy Top-down Pebbling

Maximum pebbles used: 1



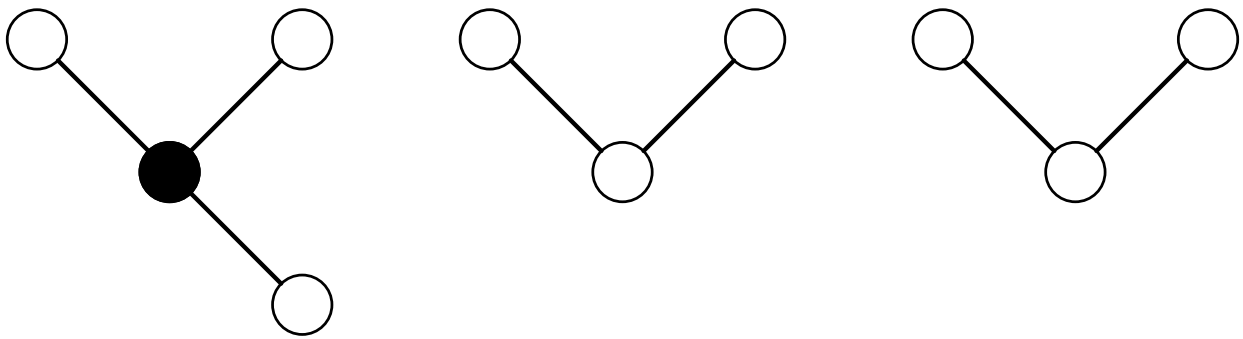
Greedy Top-down Pebbling

Maximum pebbles used: 2



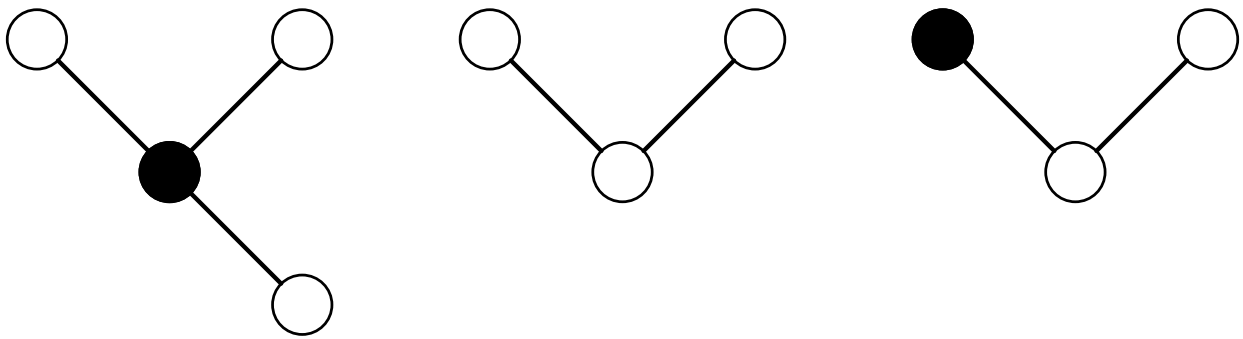
Greedy Top-down Pebbling

Maximum pebbles used: 2



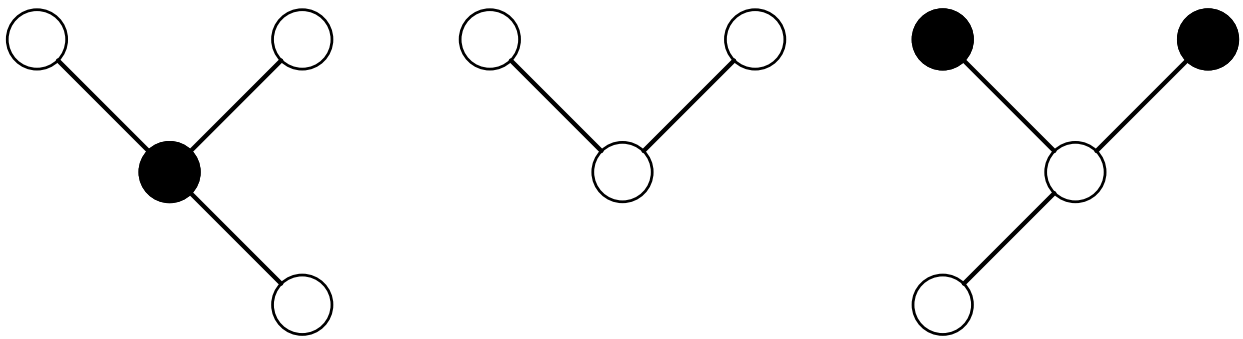
Greedy Top-down Pebbling

Maximum pebbles used: 2



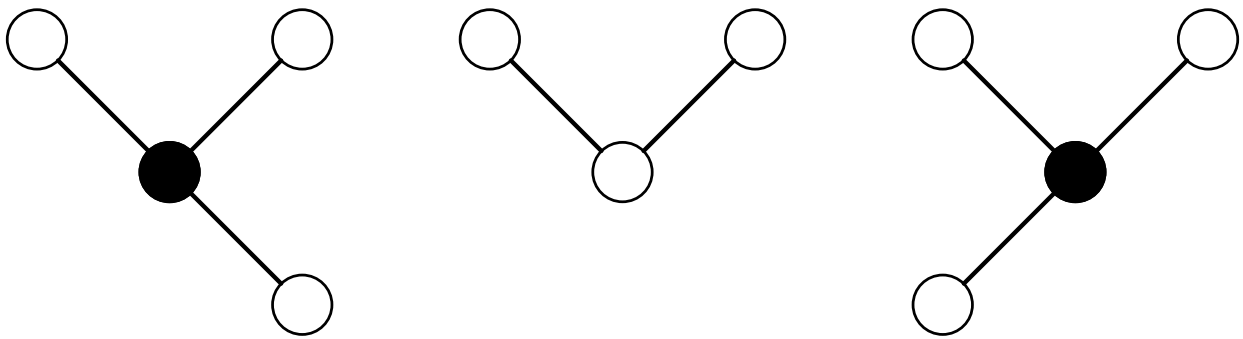
Greedy Top-down Pebbling

Maximum pebbles used: 3



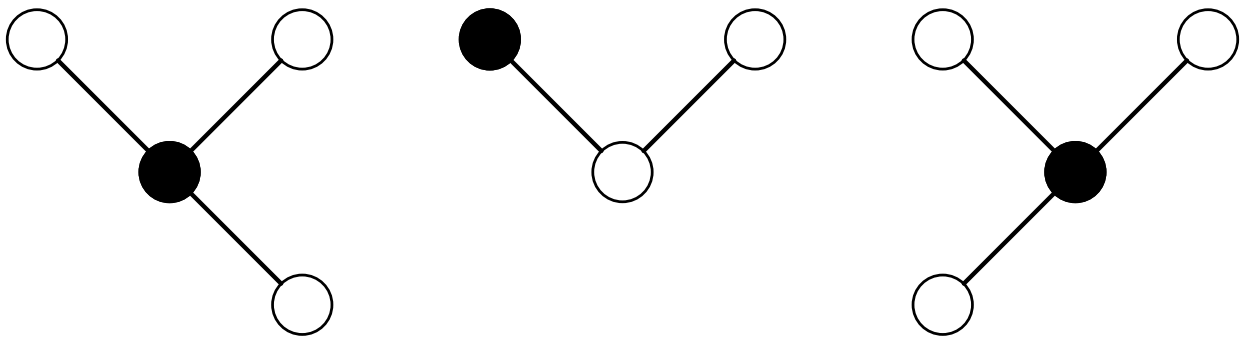
Greedy Top-down Pebbling

Maximum pebbles used: 3



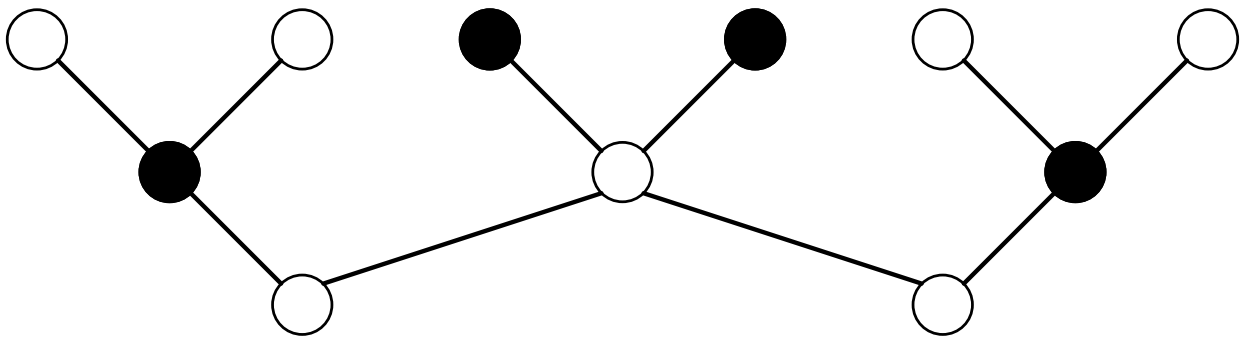
Greedy Top-down Pebbling

Maximum pebbles used: 3



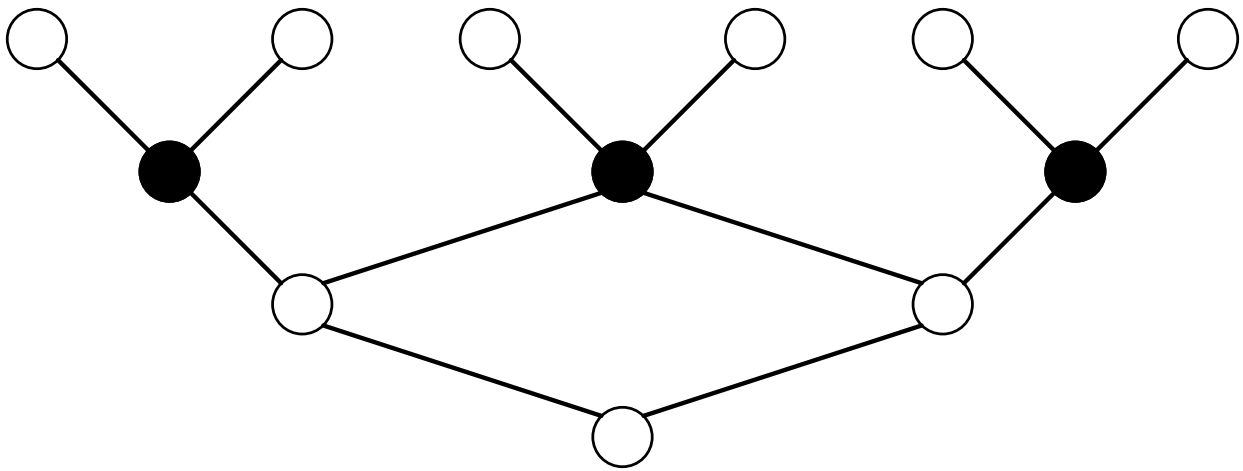
Greedy Top-down Pebbling

Maximum pebbles used: 4



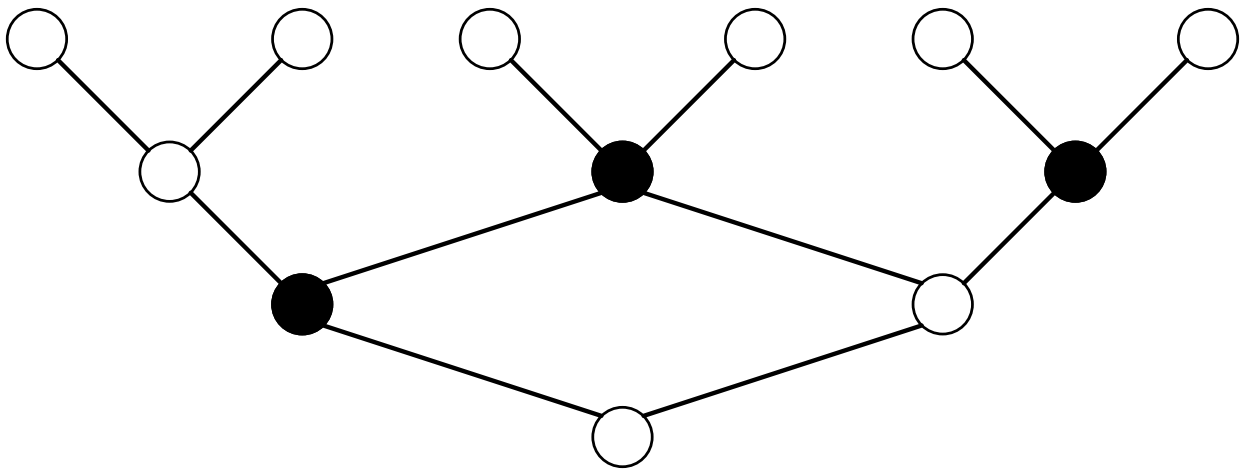
Greedy Top-down Pebbling

Maximum pebbles used: 4



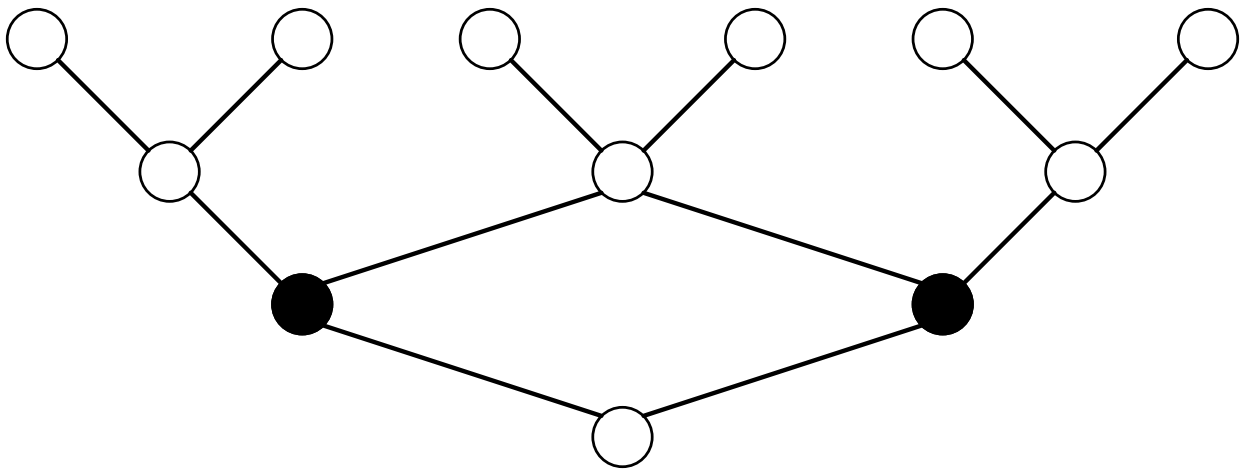
Greedy Top-down Pebbling

Maximum pebbles used: 4



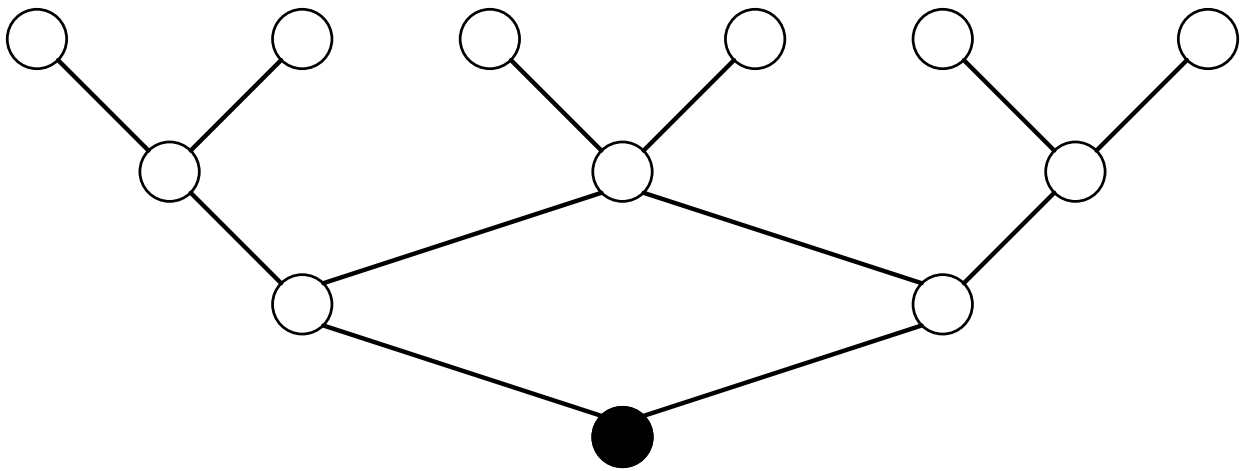
Greedy Top-down Pebbling

Maximum pebbles used: 4



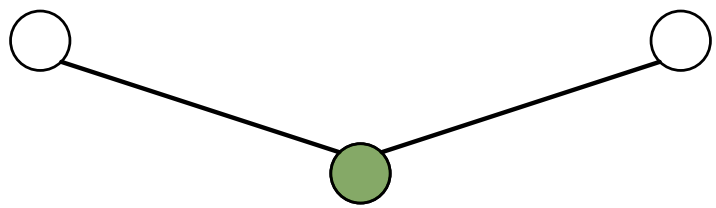
Greedy Top-down Pebbling

Maximum pebbles used: 4



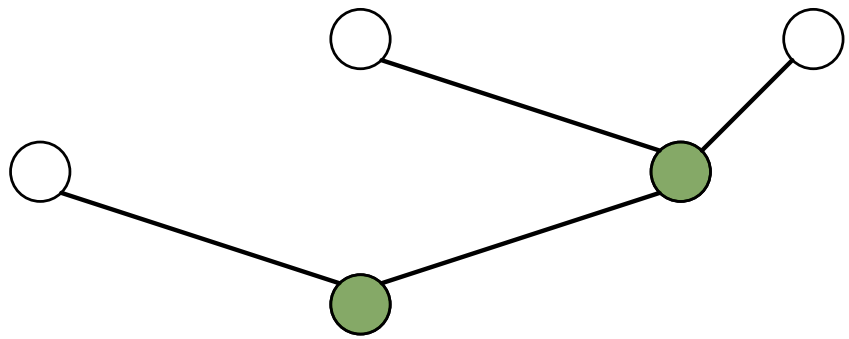
Greedy Bottom-up Pebbling

Maximum pebbles used: 0



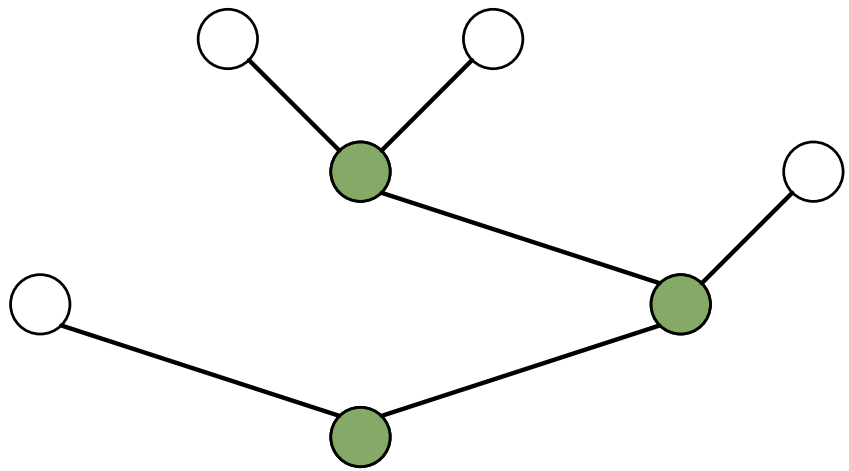
Greedy Bottom-up Pebbling

Maximum pebbles used: 0



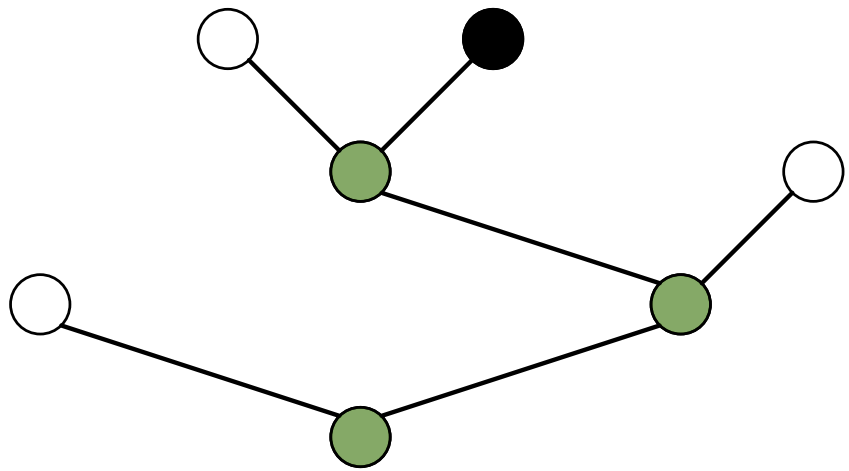
Greedy Bottom-up Pebbling

Maximum pebbles used: 0



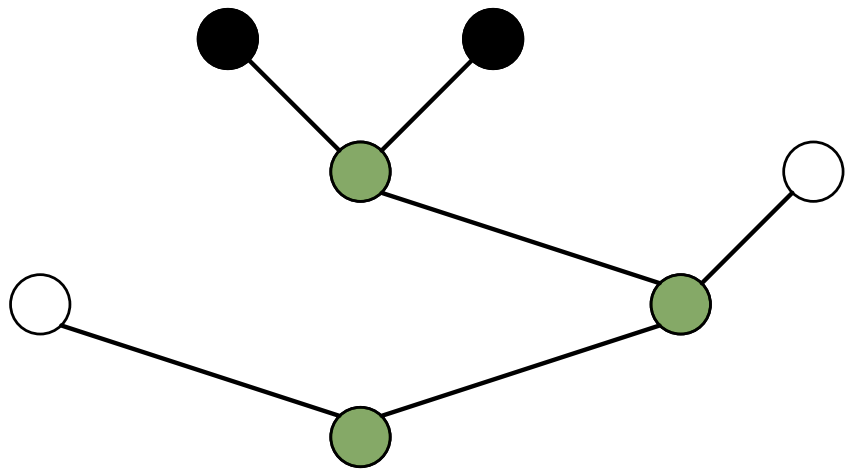
Greedy Bottom-up Pebbling

Maximum pebbles used: 1



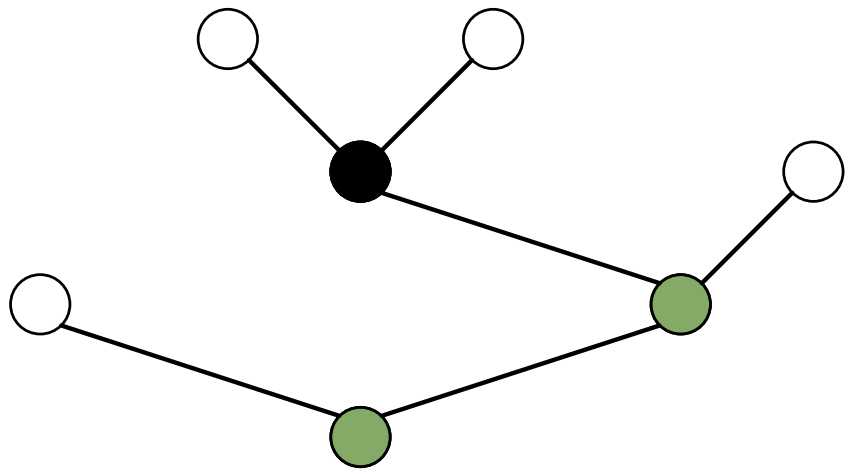
Greedy Bottom-up Pebbling

Maximum pebbles used: 2



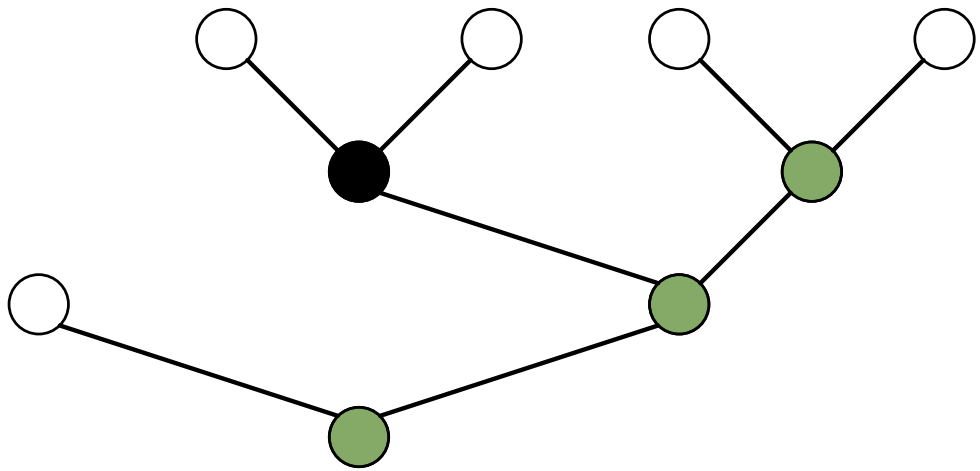
Greedy Bottom-up Pebbling

Maximum pebbles used: 2



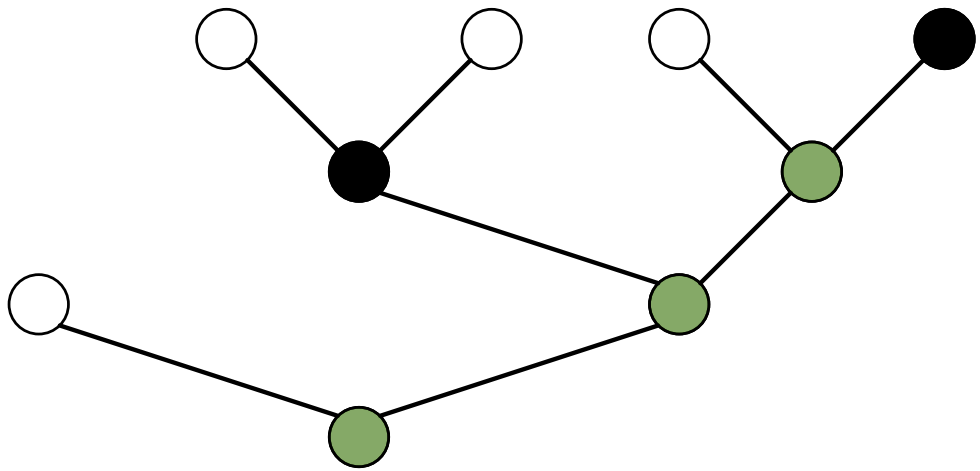
Greedy Bottom-up Pebbling

Maximum pebbles used: 2



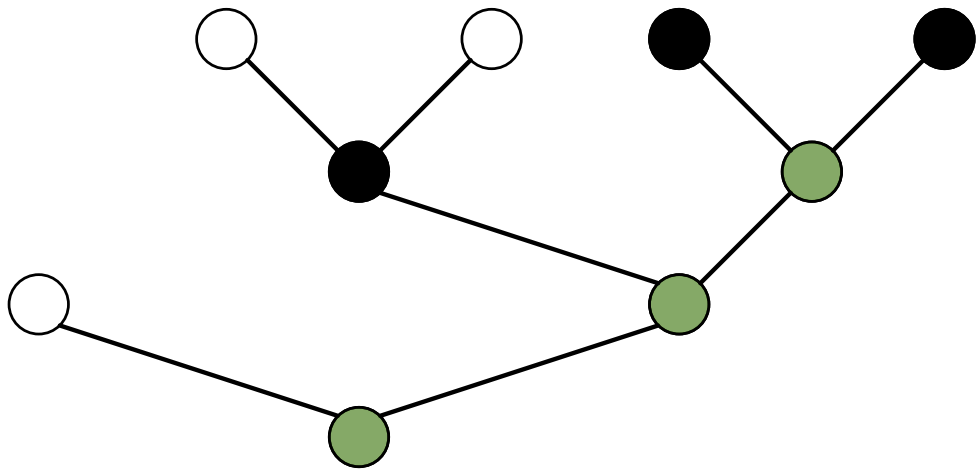
Greedy Bottom-up Pebbling

Maximum pebbles used: 2



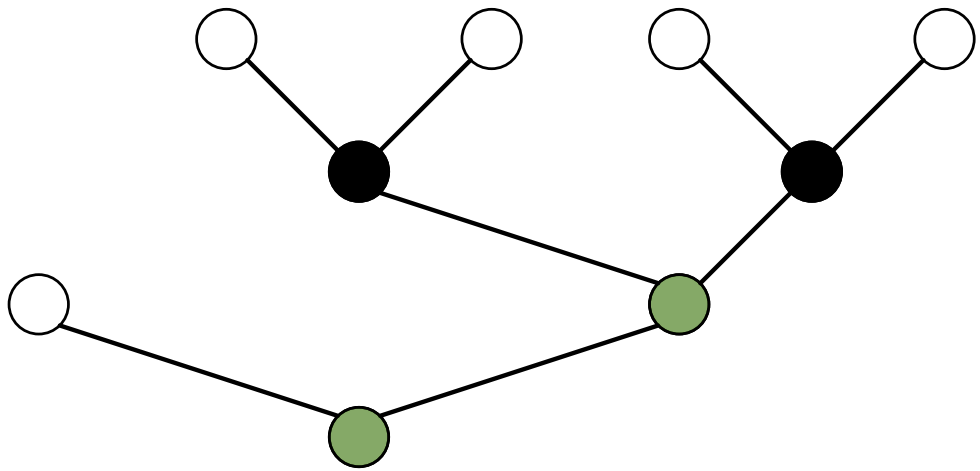
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



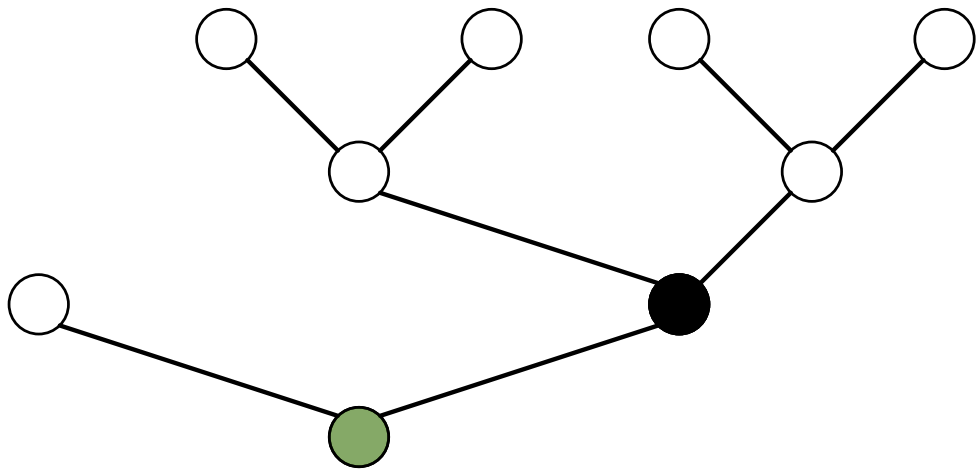
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



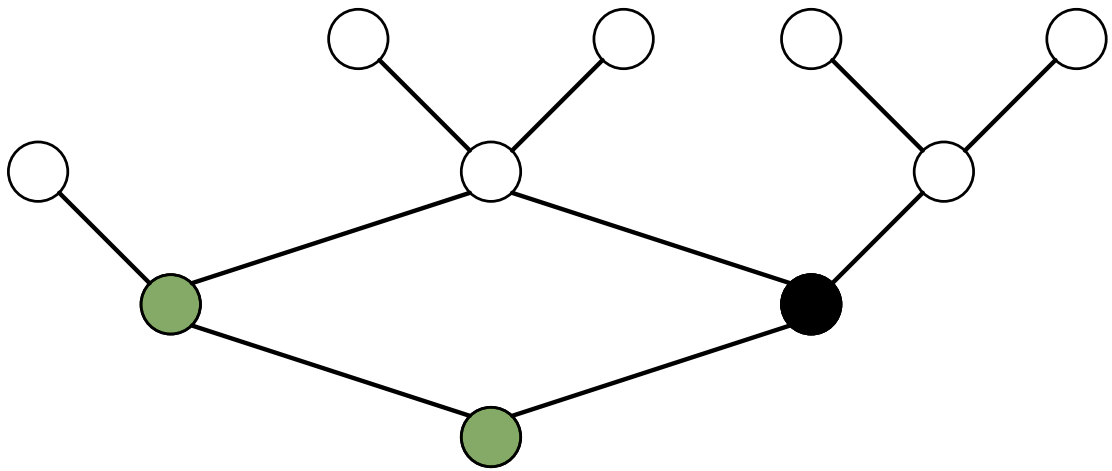
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



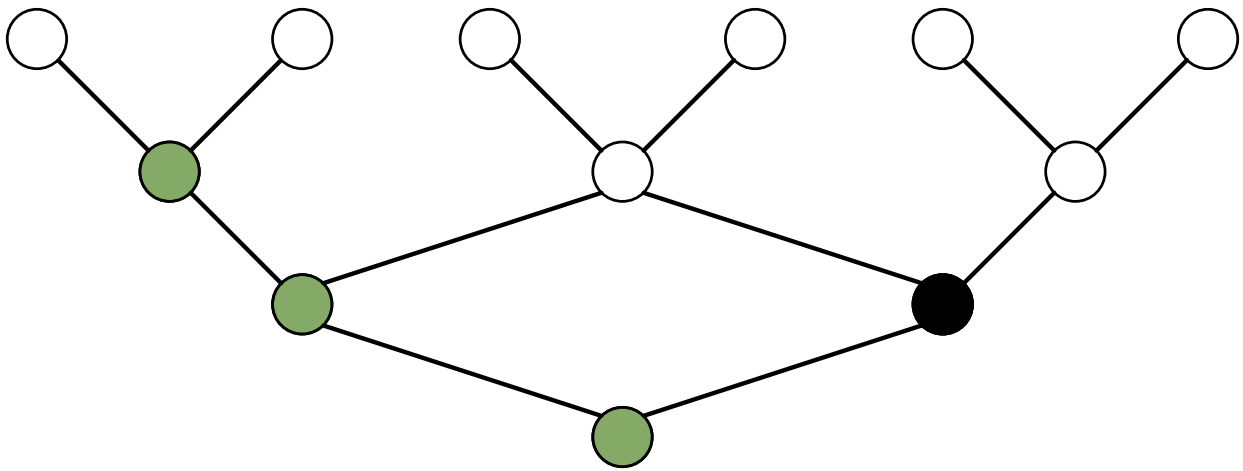
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



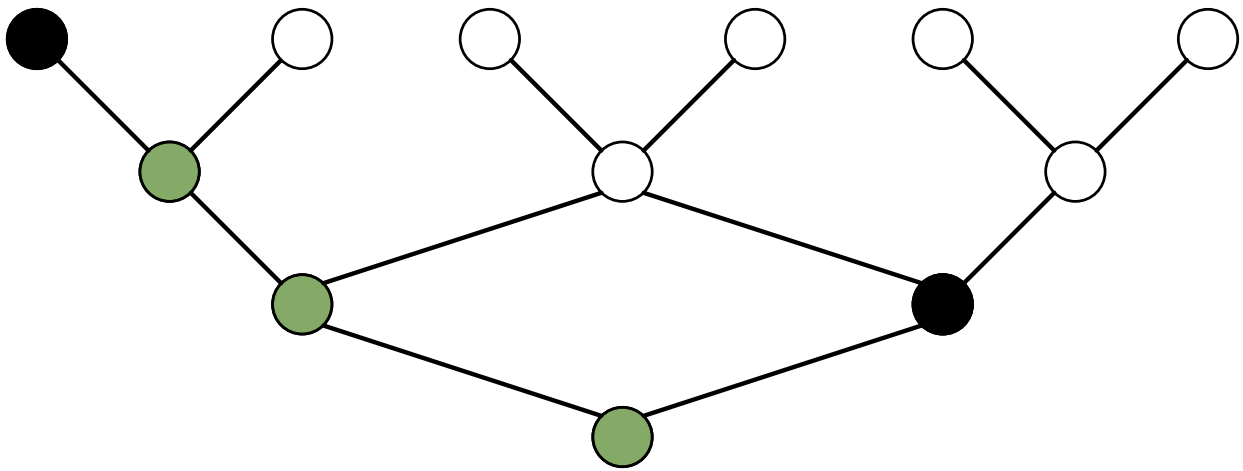
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



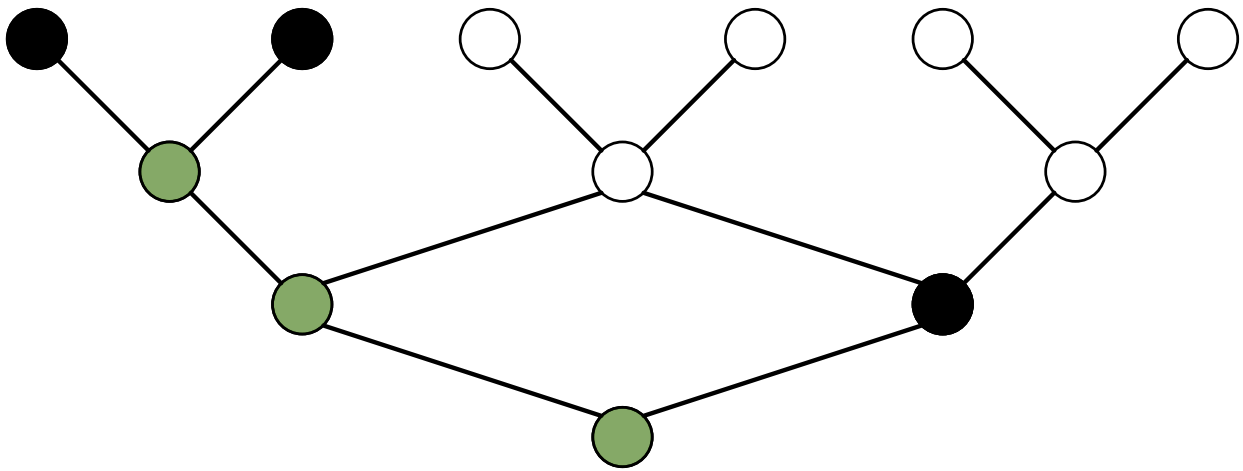
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



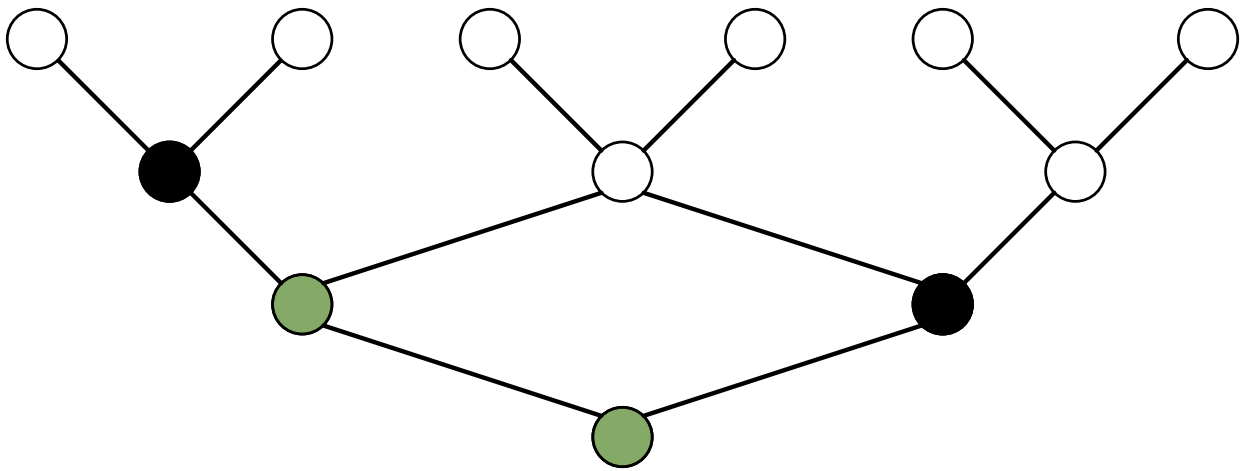
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



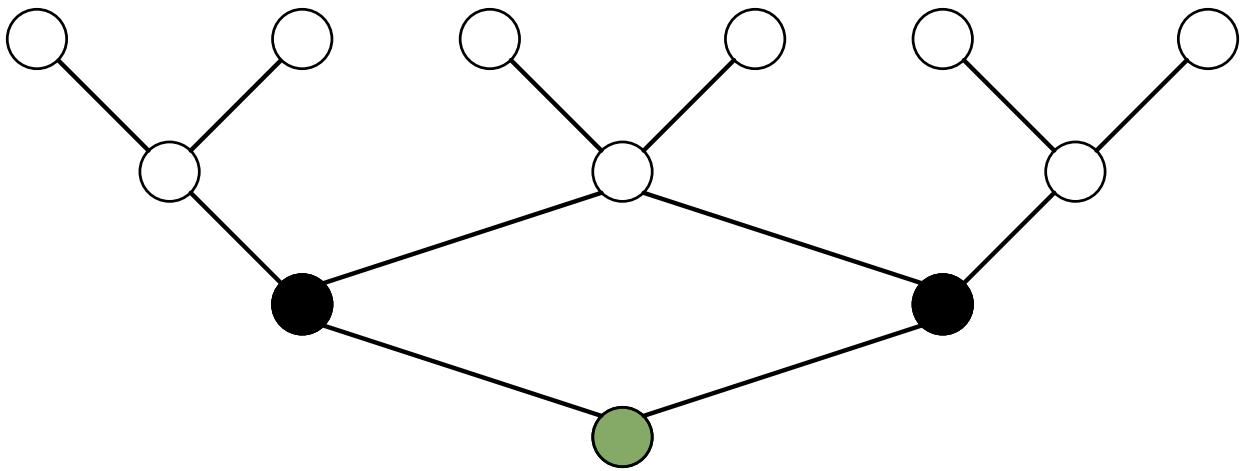
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



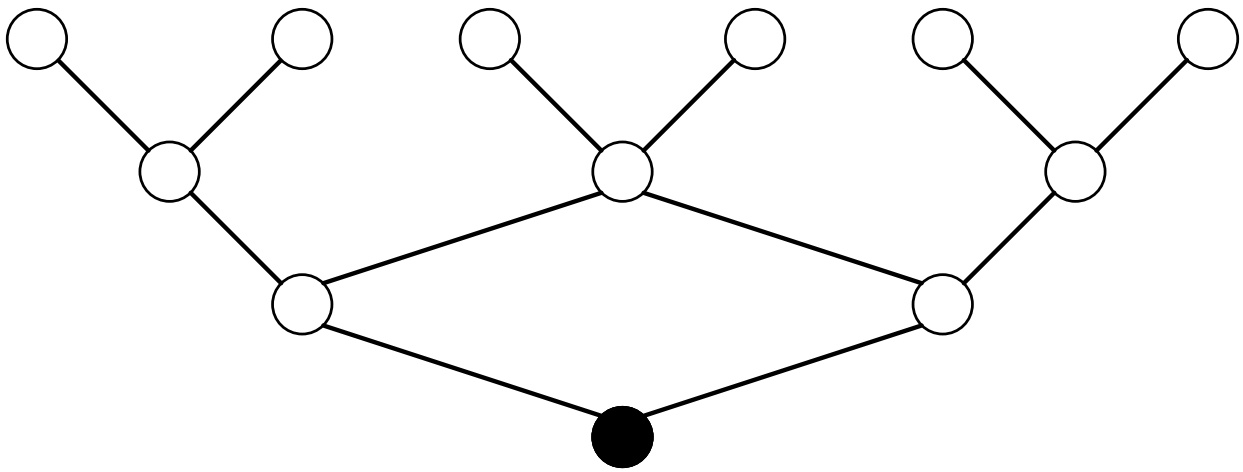
Greedy Bottom-up Pebbling

Maximum pebbles used: 3



Greedy Bottom-up Pebbling

Maximum pebbles used: 3



Experiments

Results

Algorithm	Space Compression	Speed
Top-down Pebbling*	-40.4776%	1.4 nodes/ms
Bottom-up Pebbling**	6.8%	12.2 nodes/ms

Used heuristics

- *: removes pebbles $>$ pebbled premises $>$ children with pebbled premises $>$ index in the original proof
- **: last child of a node $>$ number of children $>$ index in the original proof

Conclusion & Future Work

Subsumption

- Top-down-S interesting replacement for DAGification
- Bottom-up-S could possibly do much
- Boost performance
- Fix problems with Bottom-up-S

Recursive Split

- Promising Idea
- Still needs some fine tuning

Greedy Pebbling

- Top-down version is not clever enough
- Bottom-up version shows nice results
- Find better heuristics for Bottom-up

Special Thanks to:

Google &

European Master's Program in Computational Logic (EMCL)

for financial support

Thank you for your attention

Feel free to ask questions

http://github.com/Paradoxika/Skeptik