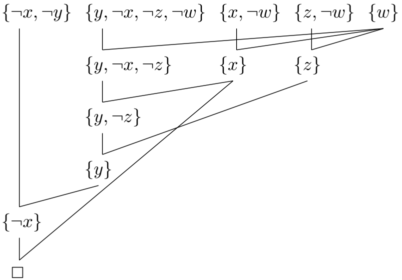
Reversible Pebble Games and the Relation Between Tree-Like and General Resolution Space

Jacobo Torán and Florian Wörz
Universität Ulm

Resolution

- only one derivation rule:

$$\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$$

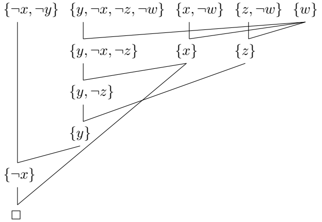


- Length of π = # of clauses in π
- Clause Space of $\pi = \max$ # of clauses in memory simultaneously during π
- Variable Space of $\pi = \max$ # of variables in memory simultaneously during π

- Tree-Res, if refutation DAG is a tree ($\rightarrow$ maybe need to rederive clauses)

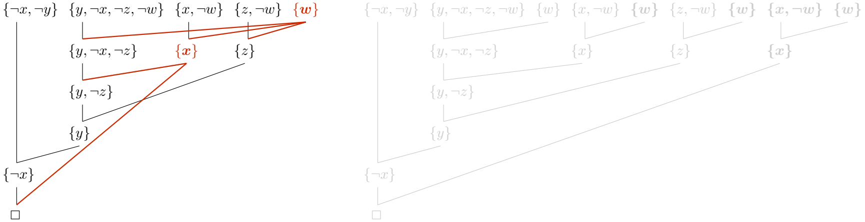
General vs. Tree-like Resolution Refutations

If a clause is needed more than once in a refutation, it has to be rederived each time.



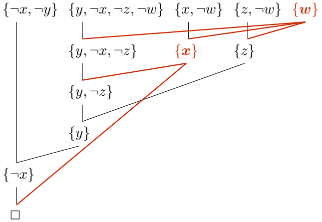
General vs. Tree-like Resolution Refutations

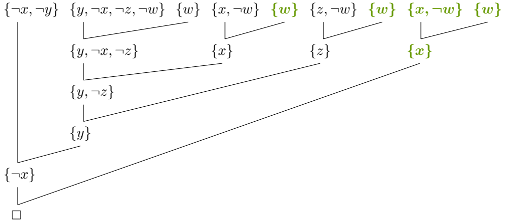
If a clause is needed more than once in a refutation, it has to be rederived each time.



General vs. Tree-like Resolution Refutations

If a clause is needed more than once in a refutation, it has to be rederived each time.





There is an almost optimal separation between general and tree-like resolution w. r. t. length:

$\exists$ a family $(F_n)_{n\in\mathbb{N}}$ of unsatsfiable formulas in $\mathrm{O}(n)$ variables with

- resolution refutations of length L (linear in n),
- **but** any tree-like resolution refutation requires length $\exp\left(\Omega(\frac{L}{\log L})\right)$.

Matching upper bound of $\exp\left(\mathrm{O}\left(\frac{L \log\log L}{\log L}\right)\right)$ for tree-like resolution length of any formula that can be refuted in length L by general resolution.

[Ben-Sasson, Impagliazzo, Wigderson 04]

¿What about space?

Configuration-style Resolution

A resolution refutation of an unsatisfiable CNF formula F is an ordered sequence of memory configurations (sets of clauses)

$$\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t),$$

s. th. $\mathbb{M}_0 = \varnothing$, $\square \in \mathbb{M}_t$ and for each $i \in [t]$, the configuration $\mathbb{M}_i$ is obtained from $\mathbb{M}_{i-1}$ by applying exactly one of the following rules:

- **Axiom Download:** $\mathbb{M}_i = \mathbb{M}_{i-1} \cup \{C\}$ for some axiom $C \in F$.
- **Erasure:** $\mathbb{M}_i = \mathbb{M}_{i-1} \setminus \{C\}$ for some $C \in \mathbb{M}_{i-1}$.
- **Inference:**

$$\mathbb{M}_i = \mathbb{M}_{i-1} \cup \{D\}$$

for some resolvent D inferred from $C_1, C_2 \in \mathbb{M}_i$ by the resolution rule.

The proof π is said to be tree-like, if we replace the inference rule with the following rule [Esteban T. 01]:

Tree-like Inference: $\mathbb{M}_i = \big(\mathbb{M}_{i-1} \cup \{D\}\big) \setminus \{C_1, C_2\}$ for some resolvent D inferred from $C_1, C_2 \in \mathbb{M}_i$, ie we delete both parent clauses immediately.

Configuration-style Resolution

A resolution refutation of an unsatisfiable CNF formula F is an ordered sequence of memory configurations (sets of clauses)

$$\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t),$$

s. th. $\mathbb{M}_0 = \varnothing$, $\Box \in \mathbb{M}_t$ and for each $i \in [t]$, the configuration $\mathbb{M}_i$ is obtained from $\mathbb{M}_{i-1}$ by applying exactly one of the following rules:

- **Axiom Download:** $\mathbb{M}_i = \mathbb{M}_{i-1} \cup \{C\}$ for some axiom $C \in F$.
- **Erasure:** $\mathbb{M}_i = \mathbb{M}_{i-1} \setminus \{C\}$ for some $C \in \mathbb{M}_{i-1}$.
- **Inference:**

  $$\mathbb{M}_i = \mathbb{M}_{i-1} \cup \{D\}$$

  for some resolvent D inferred from $C_1, C_2 \in \mathbb{M}_i$ by the resolution rule.

The proof π is said to be tree-like, if we replace the inference rule with the following rule [Esteban T. 01]:

Tree-like Inference: $\mathbb{M}_i = \big(\mathbb{M}_{i-1} \cup \{D\}\big) \setminus \{C_1, C_2\}$ for some resolvent D inferred from $C_1, C_2 \in \mathbb{M}_i$, ie we delete both parent clauses immediately.

Complexity Measures for Resolution

For a memory configuration $\mathbb{M}$:

- $\mathrm{CS}(\mathbb{M}) := |\mathbb{M}|$, i. e., number of clauses in $\mathbb{M}$,

For a refutation $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$:

- $\mathrm{CS}(\pi) := \max_{i \in [t]} \mathrm{CS}(\mathbb{M}_i)$, i. e., max. # of clauses in a config,
- $\mathrm{L}(\pi) := t$.

For a complexity measure μ and a formula F

$$\mu(F \vdash \Box) := \min_{\pi : F \vdash \Box} \mu(\pi).$$

Prefix "Tree-" indicated tree-like resolution.

Games as tools

The Prover-Delayer Game

[Pudlák, Impagliazzo '00]
Given: An unsatisfiable CNF formula F
Two players take rounds until a clause in F is falsified

Prover	**Delayer**
• Wants to falisify $C \in F$ (then Game Over)	
• Queries a variable x of F	
	• Answers – $x = 0$, – $x = 1$ or – $x = *$ ("you choose“)
• Plugs answer of Delayer in / chooses value for $*$	

Score of Delayer = $\#$ of $*$'s

The Prover-Delayer Game

A Combinatorial Characterisation for $\mathrm{Tree\text{-}CS}$

Definition (Game value of the Prover-Delayer game)

Let F be an unsatisfiable CNF formula.
$\mathsf{PD}(F) := \max$ pts. of Delayer on F against optimal strategy of Prover.

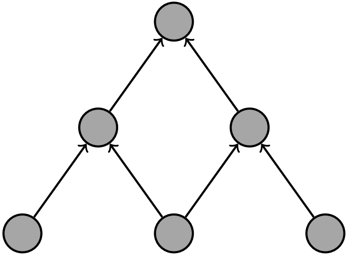
Theorem ([Esteban, T. '03])

Let F be an unsatisfiable CNF formula. Then

$$\mathrm{Tree\text{-}CS}(F \vdash \square) = \mathsf{PD}(F) + 2.$$

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

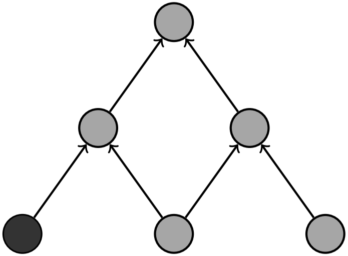


$\max$ # of pebbles used at any point:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.



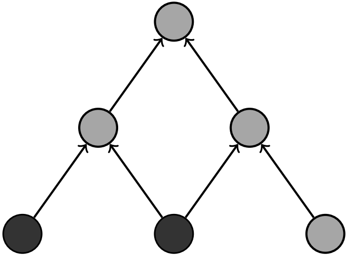
$\max$ # of pebbles used at any point:

I

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

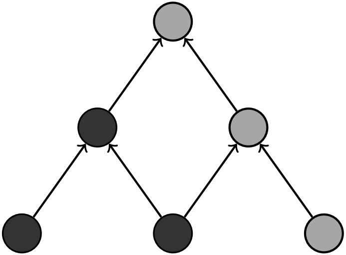


$\max$ # of pebbles used at any point:
II

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

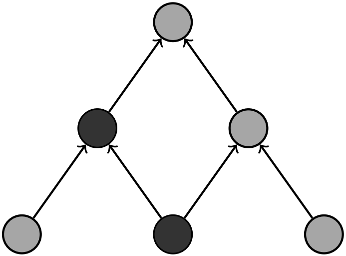


$\max$ # of pebbles
used at any point:
III

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.



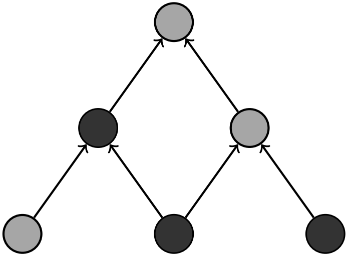
max # of pebbles used at any point:

III

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

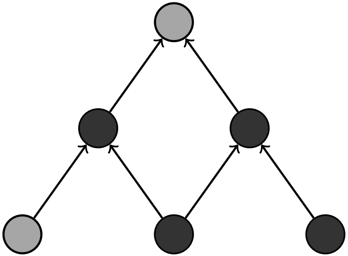


max # of pebbles
used at any point:
III

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

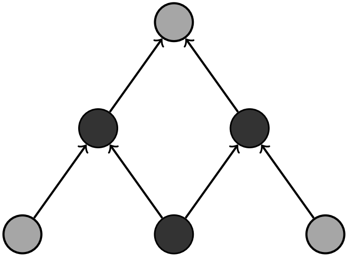


max # of pebbles used at any point: IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

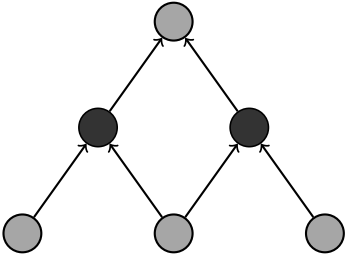


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.



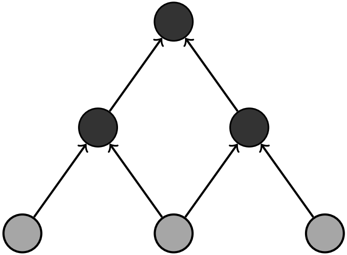
max # of pebbles used at any point:

IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

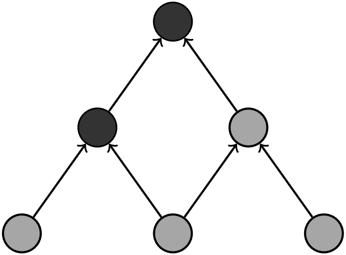


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

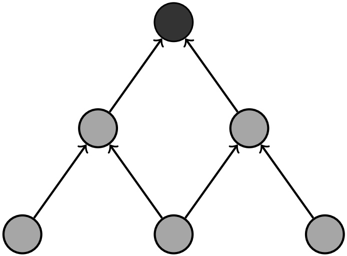


max # of pebbles used at any point: IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Black Pebble Game

Goal: Get a single black pebble on the sink of the graph.

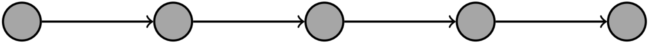


max # of pebbles
used at any point:
IIII

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** At any time

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max \#$ of pebbles used at any point:

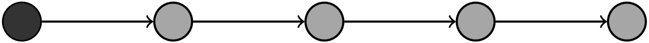


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max \#$ of pebbles used at any point: I

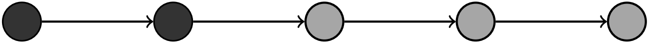


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: II

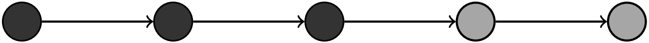


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

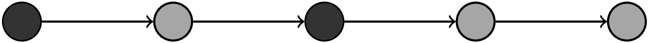


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

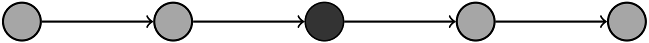


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

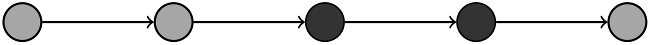


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

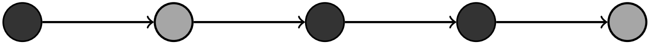


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: III

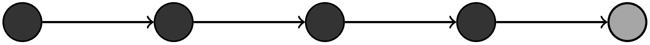


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

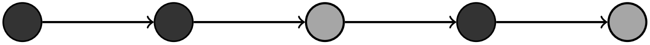


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

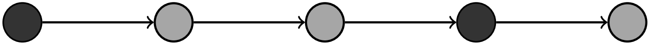


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

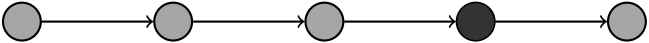


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

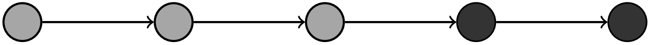


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

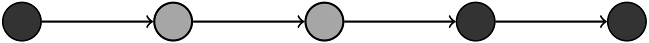


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

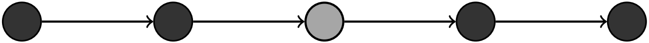


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

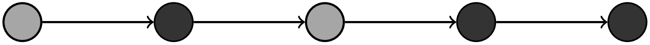


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

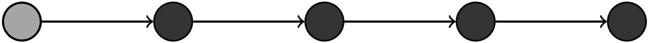


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

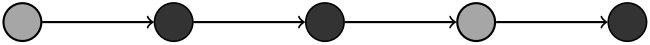


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

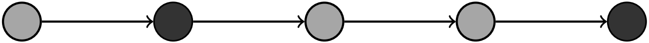


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

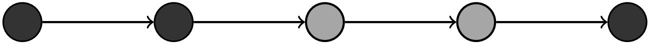


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

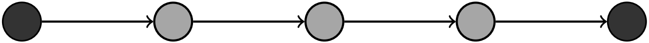


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII

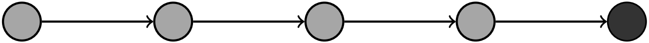


Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: $\max$ # of pebbles used at any point: IIII



Different rules:

- **Pebble Placement:** On empty vertex if all direct predecessors have a pebble (in particular: can always pebble sources)
- **Pebble Removal:** Only if all direct predecessors have a pebble (in particular: can always unpebble sources)

Complexity Measures for the Pebble Games

$$\text{Black}(G) := \min_{\text{black pebblings } \mathcal{P}} \Big(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

$$\text{Rev}(G) := \min_{\text{rev. pebblings } \mathcal{P}} \Big(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

Plethora of connections to resolution i. a.:
$\mathrm{CS}(\pi) = \min_{\pi} \mathsf{Black}(G_\pi)$ $\pi : F \vdash \Box$ [Esteban, T. '01].

We will show:
$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$.
The minimum is over all refutation, not only tree-like ones.

Complexity Measures for the Pebble Games

$$\mathsf{Black}(G) := \min_{\text{black pebblings } \mathcal{P}} \Big(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

$$\mathsf{Rev}(G) := \min_{\text{rev. pebblings } \mathcal{P}} \Big(\max \ \# \text{ of pebbles used at any point in } \mathcal{P} \Big)$$

Plethora of connections to resolution i. a.:
$\mathrm{CS}(\pi) = \min_\pi \mathsf{Black}(G_\pi)$ $\pi : F \vdash \Box$ [Esteban, T. '01].

We will show:
$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$.
The minimum is over all refutation, not only tree-like ones.

Yet another game

Rev(G) is hard to compute

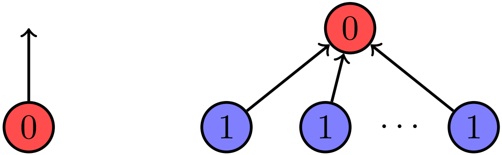
Raz–McKenzie Game to the help [Raz, McKenzie '97]

Given: A single sink DAG G

Two players take rounds... until Game Over..., i. e., when we have:

Pebbler	**Colourer**
• Places pebble on sink	
	• Colours it with red $\hat{=}\, 0$
• Chooses empty vertex	
	• Colours it red $\hat{=}\, 0$ or blue $\hat{=}\, 1$

Until



Either a red source **or** red vertex with all predecessors blue.

R-Mc(G) := smallest r s. th. Pebbler wins in $\leq r$ rounds
regardless of how Colourer plays

$\mathrm{Rev}(G) = \text{R-Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathrm{Rev}(G) = \text{R-Mc}(G)$$

Example: $\mathrm{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

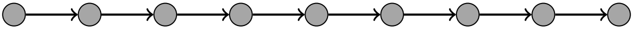
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



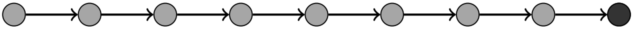
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



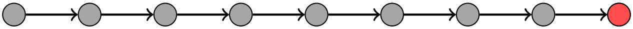
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

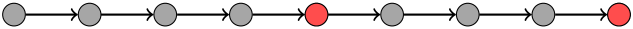
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



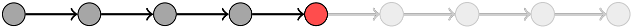
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



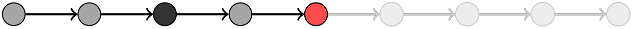
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



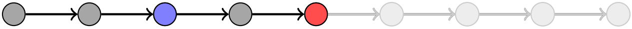
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



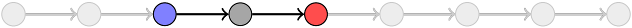
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



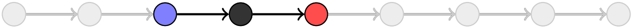
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



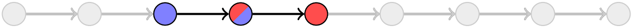
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



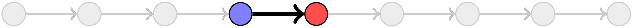
$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



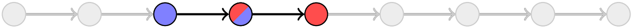
Rev(G) = R-Mc(G)

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \text{R-Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$



$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$

Theorem ([Chan '13])

For any single-sink DAG G:

$$\mathsf{Rev}(G) = \mathsf{R\text{-}Mc}(G)$$

Example: $\mathsf{Rev}(P_n) = \mathsf{R\text{-}Mc}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

Upper bounds for Tree-CS

$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathrm{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game

$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.
Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses C $\longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\text{Tree-CS}(F \vdash \Box) \le \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.
Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game
$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game

$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C{\restriction}_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\mathrm{Tree\text{-}CS}(F \vdash \square) \leq \min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game

$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\lceil_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

Given: a res. refutation π of F with a ref.-graph G_π and $\mathsf{Rev}(G_\pi) =: k$.

AIM: Give a strategy for Prover in the PD-game under which he has to pay at most k points.

Idea: Simulate the strategy of Pebbler in the Raz–McKenzie game

$\rightarrow$ a falsifying part. assignment α of init. clause will be produced

Stages of the game: Pebbler chooses $C \longrightarrow$ Prover queries vars. in C not yet assigned by α (& extends with Delayer's answers) until either

1. the clause C ist sat./fals. by α
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when C is given colour $C\restriction_\alpha$
2. a variable is given $*$ by Delayer
   $\rightarrow$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

$\text{Tree-CS}(F \vdash \Box) \le \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

Tree-CS$(F \vdash \Box) \le \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

Tree-CS$(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$

Tree-CS$(F \vdash \square) \leq \min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

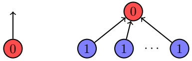
The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\square$



$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

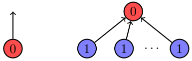
The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$



$\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

Proof sketch:

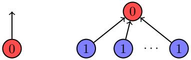
The game is played until α falsifies a clause in F.

After at most k stages the Raz–McKenzie game finished
$\Rightarrow$ Delayer can score at most k points.

Only left to show: At the end of the game a clause of F is fals. by α.

When Raz–McKenzie finishes:

1. either a source vertex in G_π is assigned colour 0 by Colourer,
   $\rightarrow$ since α defines Colourer's answer: α fals. a clause in F.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\rightarrow$ not possible, since no α can sat'y two parent clauses in a resolution proof, while falsifying their resolvent! $\Box$



$$\text{Tree-CS}(F \vdash \square) \leq \min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) + 2$$

On the other hand:

$$\min_{\pi : F \vdash \square} \mathsf{Rev}(G_\pi) \leq \text{Tree-CS}(F \vdash \square)(\lceil \log n \rceil + 1)$$

and there are formulas for which this bound is tight.

An upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS^*

[Razborov '18] introduced the concept of amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04] $\mathrm{Rev}(G_\pi) \leq \min_{\mathcal{P}} \big(\mathrm{space}(\mathcal{P}) \cdot \log \mathrm{time}(\mathcal{P})\big)$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathrm{space}(\mathcal{P})$ and length $\mathrm{time}(\mathcal{P})$. $\Box$

An upper bound for Tree-CS in terms of CS*

[Razborov '18] introduced the concept of amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\text{Tree-CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04] $\mathrm{Rev}(G_\pi) \leq \min_{\mathcal{P}} \big(\text{space}(\mathcal{P}) \cdot \log \text{time}(\mathcal{P})\big)$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to space$(\mathcal{P})$ and length time$(\mathcal{P})$. $\Box$

An upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS^*

[Razborov '18] introduced the concept of amortised clause space:

$$\mathrm{CS}^*(F \vdash \square) := \min_{\pi : F \vdash \square} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\mathrm{Tree\text{-}CS}(F \vdash \square) \leq \mathrm{CS}^*(F \vdash \square) + 2.$

Proof.

- [Královič '04] $\mathsf{Rev}(G_\pi) \leq \min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big)$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathsf{space}(\mathcal{P})$ and length $\mathsf{time}(\mathcal{P})$. $\square$

An upper bound for $\mathrm{Tree\text{-}CS}$ in terms of CS^*

[Razborov '18] introduced the concept of amortised clause space:

$$\mathrm{CS}^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} \big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\big)$$

Corollary

$\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \mathrm{CS}^*(F \vdash \Box) + 2.$

Proof.

- [Královič '04] $\mathrm{Rev}(G_\pi) \leq \min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big)$, where the minimum is taken over all black pebblings $\mathcal{P}$ of G_π.
- Every black pebbling $\mathcal{P}$ of G_π defines a configurational refutation of F with clause space equal to $\mathsf{space}(\mathcal{P})$ and length $\mathsf{time}(\mathcal{P})$. $\Box$

How large can be the gap between CS and $\mathrm{Tree\text{-}CS}$?

Pebbling Formulas (formulas over DAGs)

Pebbling Formula

Clauses of Peb_G:

u

v

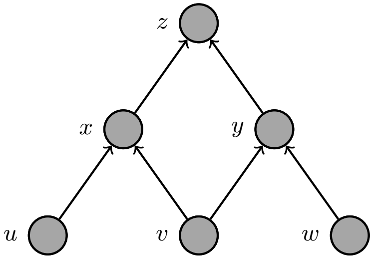
w

$(u \wedge v) \rightarrow x \ = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z \ = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

Pebbling Formula

Clauses of Peb_G:

u

v

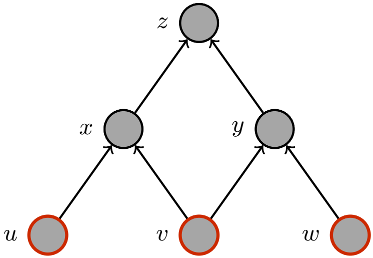
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

Pebbling Formula

Clauses of Peb_G:

u

v

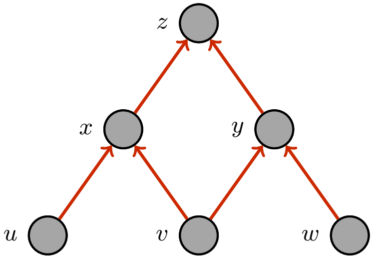
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

Pebbling Formula

Clauses of Peb_G:

u

v

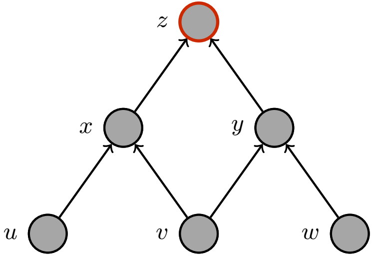
w

$(u \wedge v) \rightarrow x = \overline{u} \vee \overline{v} \vee x$

$(v \wedge w) \rightarrow y = \overline{v} \vee \overline{w} \vee y$

$(x \wedge y) \rightarrow z = \overline{x} \vee \overline{y} \vee z$

$\overline{z}$



Encode the rules of the black pebble game in a formula (i. e., formula is defined over an underlying DAG):

- source vertices are true
- truth propagates upwards
- but the sink vertex is false

XORification $\oplus_2$

Make formulas slightly harder to refute

- For a technical reason we need the XORification of our pebbling formulas.
- (XORification being a common technique used in proof complexity).

- **Simple Idea:** Substitute each variable x with $x_1 \oplus x_2$ and expand result into CNF.

XORification $\oplus_2$

Make formulas slightly harder to refute

- For a technical reason we need the XORification of our pebbling formulas.
- (XORification being a common technique used in proof complexity).

- **Simple Idea:** Substitute each variable x with $x_1 \oplus x_2$ and expand result into CNF.

Reversible Pebbling meets Tree-CS in the Special Case of Pebbling Formulas

Theorem

For all DAGs G with a unique sink:

$$\mathsf{Rev}(G) + 2 \leq \text{Tree-CS}\big(\text{Peb}_G[\oplus_2] \vdash \Box\big) \leq 2 \cdot \mathsf{Rev}(G) + 2.$$

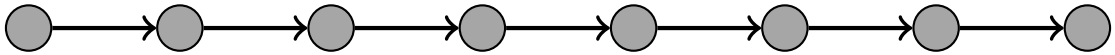
Obtaining Space-Separations with Pebble games

Idea:

- $\mathrm{CS}(\mathrm{Peb}_G[\oplus_2] \vdash \Box) = \mathrm{O}\big(\mathsf{Black}(G)\big)$
- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) = \Omega\big(\mathsf{Rev}(G)\big)$

$\Longrightarrow$ Construct a graph family with a gap between its black and reversible pebbling price

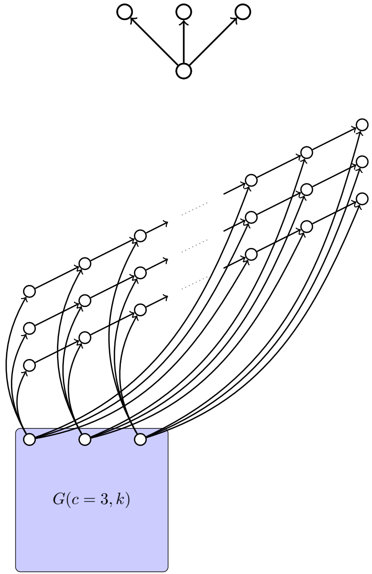
Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

Obtaining Space-Separations with Pebble games

Non-constant black pebbling number and Black-Rev-separation:



Obtaining Space-Separations with Pebble games

The best known separation

For "slowly enough" growing space functions $s(n)$ there is a family of pebbling formulas $\left(\mathrm{Peb}_{G_n}[\oplus_2]\right)_{n=1}^{\infty}$ with $\Theta(n)$ variables such that

- $\mathrm{CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \Box) = \mathrm{O}\big(s(n)\big)$
- $\mathrm{Tree\text{-}CS}(\mathrm{Peb}_{G_n}[\oplus_2] \vdash \Box) = \Omega\big(s(n) \log n\big)$.

¿Can we do any better?

The Tseitin formula case

The Tseitin formula case

Theorem

- For any connected graph G with n vertices and odd marking χ
  $\mathrm{Tree\text{-}CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \leq \mathrm{CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$
- There are graph families $\{G_n\}$ for which $\forall n :$
  $\mathrm{Tree\text{-}CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) = \Omega(\mathrm{CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \cdot \log n)$

$\text{Tree-CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \leq \mathrm{CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$

Proof sketch:

Let $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ be a refutation of $\mathrm{Ts}(G,\chi)$ with $\mathrm{CS}(\pi) =: k$. We use π to give a strategy for Prover in the Prover-Delayer game for which he has to pay at most $k \log n$ points.

A partial assignment α of some of the variables in $\mathrm{Ts}(G,\chi)$ is non-splitting if after applying α to the formula, the resulting graph still has an odd connected component of size at least $\frac{n}{2}$ and the rest are components are even.

There is a last step s in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$ and

(ii) α is non-splitting.

The only new clause in configuration $\mathbb{M}_{s+1}$ must be an axiom of $\mathrm{Ts}(G,\chi)$

$\text{Tree-CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \leq \text{CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$

Proof sketch:

Let $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ be a refutation of $\text{Ts}(G,\chi)$ with $\text{CS}(\pi) =: k$. We use π to give a strategy for Prover in the Prover-Delayer game for which he has to pay at most $k \log n$ points.

A partial assignment α of some of the variables in $\text{Ts}(G,\chi)$ is non-splitting if after applying α to the formula, the resulting graph still has an odd connected component of size at least $\frac{n}{2}$ and the rest are components are even.

There is a last step s in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$ and

(ii) α is non-splitting.

The only new clause in configuration $\mathbb{M}_{s+1}$ must be an axiom of $\text{Ts}(G,\chi)$

$\text{Tree-CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \leq \mathrm{CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$

Proof sketch:

Let $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ be a refutation of $\mathrm{Ts}(G,\chi)$ with $\mathrm{CS}(\pi) =: k$. We use π to give a strategy for Prover in the Prover-Delayer game for which he has to pay at most $k \log n$ points.

A partial assignment α of some of the variables in $\mathrm{Ts}(G,\chi)$ is non-splitting if after applying α to the formula, the resulting graph still has an odd connected component of size at least $\frac{n}{2}$ and the rest are components are even.

There is a last step s in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$ and

(ii) α is non-splitting.

The only new clause in configuration $\mathbb{M}_{s+1}$ must be an axiom of $\mathrm{Ts}(G,\chi)$

$\text{Tree-CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \leq \mathrm{CS}\big(\mathrm{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$

Proof sketch:

Let $\pi = (\mathbb{M}_0, \ldots, \mathbb{M}_t)$ be a refutation of $\mathrm{Ts}(G,\chi)$ with $\mathrm{CS}(\pi) =: k$. We use π to give a strategy for Prover in the Prover-Delayer game for which he has to pay at most $k \log n$ points.

A partial assignment α of some of the variables in $\mathrm{Ts}(G,\chi)$ is non-splitting if after applying α to the formula, the resulting graph still has an odd connected component of size at least $\frac{n}{2}$ and the rest are components are even.

There is a last step s in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$ and

(ii) α is non-splitting.

The only new clause in configuration $\mathbb{M}_{s+1}$ must be an axiom of $\mathrm{Ts}(G,\chi)$

$\text{Tree-CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \leq \text{CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$

Proof sketch:

A partial assignment α of some of the variables in $\text{Ts}(G,\chi)$ is non-splitting if after applying α to the formula, the resulting graph still has an odd connected component of size at least $\frac{n}{2}$ and the rest are components are even.

There is a last step in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$ and

(ii) α is non-splitting.

The only new clause in configuration $\mathbb{M}_{s+1}$ must be an axiom of $\text{Ts}(G,\chi)$

There is a way to query variables at stage $s+1$ paying only k points to Delayer and splitting G or falsifying the axiom.

$\text{Tree-CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \leq \text{CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \cdot \log n + 2$

Proof sketch:

A partial assignment α of some of the variables in $\text{Ts}(G,\chi)$ is non-splitting if after applying α to the formula, the resulting graph still has an odd connected component of size at least $\frac{n}{2}$ and the rest are components are even.

There is a last step in π for which there is a partial assignment α fulfilling:

(i) α simultaneously satisfies all clauses in $\mathbb{M}_s$ and

(ii) α is non-splitting.

The only new clause in configuration $\mathbb{M}_{s+1}$ must be an axiom of $\text{Ts}(G,\chi)$

There is a way to query variables at stage $s+1$ paying only k points to Delayer and splitting G or falsifying the axiom.

Take-Home Message

$\mathrm{Tree\text{-}CS}$ *and* CS *are different measures but "not too far" from one another*

- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) \simeq \mathsf{Rev}(G)$
- Separations between $\mathrm{Tree\text{-}CS}$ and CS by graphs G exhibiting separation between $\mathsf{Rev}(G)$ and $\mathsf{Black}(G)$
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{VS}^*(F \vdash \Box)$ for general F

Take-Home Message

$\mathrm{Tree\text{-}CS}$ *and* CS *are different measures but "not too far" from one another*

- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) \simeq \mathsf{Rev}(G)$
- Separations between $\mathrm{Tree\text{-}CS}$ and CS by graphs G exhibiting separation between $\mathsf{Rev}(G)$ and $\mathsf{Black}(G)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{VS}^*(F \vdash \Box)$ for general F (*)

(*) Some open questions hidden here. We've solved these for Tseitin formulas.

Take-Home Message

$\mathrm{Tree\text{-}CS}$ *and* CS *are different measures but "not too far" from one another*

- $\mathrm{Tree\text{-}CS}\big(\mathrm{Peb}_G[\oplus_2] \vdash \Box\big) \simeq \mathsf{Rev}(G)$
- Separations between $\mathrm{Tree\text{-}CS}$ and CS by graphs G exhibiting separation between $\mathsf{Rev}(G)$ and $\mathsf{Black}(G)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{VS}^*(F \vdash \Box)$ for general F (*)

(*) Some open questions hidden here. We've solved these for Tseitin formulas.

Thank you for your attention!