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:**
  \[
  \frac{B \lor x \quad C \lor \neg x}{B \lor C}
  \]

- **Length of** \( \pi \) **= # of clauses in** \( \pi \)
- **Clause Space of** \( \pi \) **= max # of clauses in memory simultaneously during** \( \pi \)
- **Variable Space of** \( \pi \) **= max # of variables in memory simultaneously during** \( \pi \)
- **Tree-Res**, if refutation DAG is a tree (→ 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.
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:

∃ a family \((F_n)_{n \in \mathbb{N}}\) of unsatisfiable formulas in \(O(n)\) variables with

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

Matching upper bound of \(\exp\left(O\left(\frac{L \log \log L}{\log L}\right)\right)\) 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)

$$\pi = (M_0, \ldots, M_t),$$

s. th. $M_0 = \emptyset$, $\square \in M_t$ and for each $i \in [t]$, the configuration $M_i$ is obtained from $M_{i-1}$ by applying exactly one of the following rules:

- **Axiom Download**: $M_i = M_{i-1} \cup \{C\}$ for some axiom $C \in F$.
- **Erasure**: $M_i = M_{i-1} \setminus \{C\}$ for some $C \in M_{i-1}$.
- **Inference**:

  $$M_i = M_{i-1} \cup \{D\}$$

  for some resolvent $D$ inferred from $C_1, C_2 \in M_i$ by the resolution rule.

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

**Tree-like Inference**: $M_i = (M_{i-1} \cup \{D\}) \setminus \{C_1, C_2\}$ for some resolvent $D$ inferred from $C_1, C_2 \in M_i$, 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)

$$\pi = (M_0, \ldots, M_t),$$

s. th. $M_0 = \emptyset$, $\square \in M_t$ and for each $i \in [t]$, the configuration $M_i$ is obtained from $M_{i-1}$ by applying exactly one of the following rules:

- **Axiom Download:** $M_i = M_{i-1} \cup \{C\}$ for some axiom $C \in F$.
- **Erasure:** $M_i = M_{i-1} \setminus \{C\}$ for some $C \in M_{i-1}$.
- **Inference:**

  $$M_i = M_{i-1} \cup \{D\}$$

  for some resolvent $D$ inferred from $C_1, C_2 \in M_i$ by the resolution rule.

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

**Tree-like Inference:** $M_i = (M_{i-1} \cup \{D\}) \setminus \{C_1, C_2\}$ for some resolvent $D$ inferred from $C_1, C_2 \in M_i$, ie we delete both parent clauses immediately.
Complexity Measures for Resolution

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

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

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

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

For a complexity measure $\mu$ 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

<table>
<thead>
<tr>
<th>Prover</th>
<th>Delayer</th>
</tr>
</thead>
</table>
| • Wants to falsify $C \in F$  
(then Game Over)  
• Queries a variable $x$ of $F$  
• Plugs answer of Delayer in / chooses value for *  
| • Answers  
– $x = 0$,  
– $x = 1$ or  
– $x = *$ (“you choose“)  
|  
Score of Delayer = # of *’s
The Prover-Delayer Game
A Combinatorial Characterisation for Tree-CS

Definition (Game value of the Prover-Delayer game)
Let $F$ be an unsatisfiable CNF formula.
$PD(F) := \max$ pts. of Delayer on $F$ against optimal strategy of Prover.

Theorem ([Esteban, T. '03])
Let $F$ be an unsatisfiable CNF formula. Then
$$\text{Tree-CS}(F \vdash \Box) = PD(F') + 2.$$
The Black Pebble Game

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

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

$max \# \text{ of pebbles used at any point}$
The Black Pebble Game

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

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

max \# of pebbles used at any point:

1
The Black Pebble Game

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

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

\[
\text{max \# of pebbles used at any point: } \parallel
\]
The Black Pebble Game

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

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

$max \ # \ of \ pebbles \ used \ at \ any \ point: \ III$
The Black Pebble Game

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

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

\[ \text{max \# of pebbles used at any point: } \Box \Box \Box \]
The Black Pebble Game

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

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

- **Pebble Removal:** At any time

max # of pebbles used at any point: III
The Black Pebble Game

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

- **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.

- **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.

- **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.

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

max # of pebbles used at any point: I I I I
The Black Pebble Game

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

- 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.

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

max # of pebbles used at any point: 

```
  I I I I
```
The Reversible Pebble Game

Same Goal: Get a single black pebble on the sink of the graph.
Same measure: \textit{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: \( \text{max \ # of pebbles used at any point: I} \)

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: \( \text{max } \# \text{ of pebbles used at any point}: \# \)

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: \( \text{max } \# \text{ of pebbles used at any point}: III \)

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: \( \text{max } \# \text{ of pebbles used at any point} \): 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: \( \max \# \text{ of pebbles used at any point} \): \( \text{III} \)

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: \( \text{max } \# \text{ of pebbles used at any point: IIII} \)

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: \( \text{max } \# \text{ of pebbles used at any point} \): I I I 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: I 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: \( \max \ # \ of \ pebbles \ used \ at \ any \ point: \) Ifour

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: \( \text{max } \# \text{ of pebbles used at any point: } \boxed{1111} \)

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: \( \text{max } \# \text{ of pebbles used at any point: } \)

\[ \text{IIII} \]

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: IILL

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: \( \text{max } \# \text{ of pebbles used at any point} \): I||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: \textbf{max [# of pebbles used at any point]: I I I 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: \textbf{max} \ # \ of \ pebbles \ used \ at \ any \ point: \ \textbf{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: I I I 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: $\text{max } \# \text{ of pebbles used at any point}: \text{IIII}$

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: \textit{max} \# of pebbles used at any point: \( \text{IIII} \)

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: \( \max \text{ # of pebbles used at any point: } \) I\text{I}\text{I}\text{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)
Complexity Measures for the Pebble Games

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

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

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

We will show:
\[ \text{Tree-CS}(F \vdash \square) \leq \min_{\pi : F \vdash \square} \text{Rev}(G_{\pi}) + 2. \]
The minimum is over all refutation, not only tree-like ones.
Complexity Measures for the Pebble Games

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

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

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

We will show:
\( \text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \text{Rev}(G_{\pi}) + 2. \)
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:**

<table>
<thead>
<tr>
<th>Pebbler</th>
<th>Colourer</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Places pebble on sink</td>
<td>● Colours it with red $\equiv 0$</td>
</tr>
<tr>
<td>● Chooses empty vertex</td>
<td>● Colours it red $\equiv 0$ or blue $\equiv 1$</td>
</tr>
</tbody>
</table>
Until

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

\[ \text{R-Mc}(G) : = \text{smallest } r \text{ s. th. Pebbler wins in } \leq r \text{ rounds regardless of how Colourer plays} \]
Theorem ([Chan ’13])

For any single-sink DAG $G$:

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

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

**Theorem ([Chan '13])**

For any single-sink DAG \( G \):

\[ \text{Rev}(G) = R-Mc(G) \]

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

**Theorem ([Chan '13])**

For any single-sink DAG \(G\):

\[ \text{Rev}(G) = R-Mc(G) \]

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

\[
\begin{array}{c}
\text{Diagram of } P_n \\
\end{array}
\]
\[ \text{Rev}(G) = \text{R-Mc}(G) \]

**Theorem ([Chan '13])**

*For any single-sink DAG* \( G \):

\[ \text{Rev}(G) = \text{R-Mc}(G) \]

**Example:** \( \text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n) \quad \forall n \in \mathbb{N} \)
Theorem ([Chan '13])

For any single-sink DAG $G$:

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

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

**Theorem ([Chan '13])**

*For any single-sink DAG \( G \):*

\[ \text{Rev}(G) = \text{R-Mc}(G) \]

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

Theorem ([Chan '13])

For any single-sink DAG G:

Rev(G) = R-Mc(G)

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

**Theorem ([Chan '13])**

*For any single-sink DAG \( G \):*

\[
\text{Rev}(G) = \text{R-Mc}(G)
\]

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

**Theorem ([Chan '13])**

For any single-sink DAG $G$:

\[ \text{Rev}(G) = R\text{-Mc}(G) \]

**Example:** $\text{Rev}(P_n) = R\text{-Mc}(P_n) = \Theta(\log n) \ \forall n \in \mathbb{N}$
Theorem ([Chan '13])

For any single-sink DAG $G$:

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

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

**Theorem ([Chan '13])**

*For any single-sink DAG G:*

\[ \text{Rev}(G) = \text{R-Mc}(G) \]

**Example:** \( \text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n) \ \forall n \in \mathbb{N} \)
\textbf{Theorem ([Chan '13])}

For any single-sink DAG $G$:

\[ \text{Rev}(G) = \text{R-Mc}(G) \]

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

**Theorem ([Chan '13])**

*For any single-sink DAG* \( G \):

\[
\text{Rev}(G) = \text{R-Mc}(G)
\]

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

**Theorem ([Chan '13])**

For any single-sink DAG \( G \):

\[
\text{Rev}(G) = R-Mc(G)
\]

**Example:** \( \text{Rev}(P_n) = R-Mc(P_n) = \Theta(\log n) \ \forall n \in \mathbb{N} \)
\[ \text{Rev}(G) = \text{R-Mc}(G) \]

**Theorem ([Chan '13])**

For any single-sink DAG \( G \):

\[ \text{Rev}(G) = \text{R-Mc}(G) \]

**Example:** \( \text{Rev}(P_n) = \text{R-Mc}(P_n) = \Theta(\log n) \ \forall n \in \mathbb{N} \)
Upper bounds for Tree-CS
Tree-CS(\(F \vdash \Box\)) \leq \min_{\pi:F \vdash \Box} \Rev(G_\pi) + 2

**Proof sketch:**

**Given:** a res. refutation \(\pi\) of \(F\) with a ref.-graph \(G_\pi\) and \(\Rev(G_\pi) =: k\).

**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 → a falsifying part. assignment \(\alpha\) of init. clause will be produced

**Stages of the game:** Pebbler chooses \(C\) → Prover queries vars. in \(C\) not yet assigned by \(\alpha\) (& extends with Delayer’s answers) until either

1. the clause \(C\) ist sat./fals. by \(\alpha\)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \(C\) is given colour \(C|_\alpha\)

2. a variable is given * by Delayer
   → Prover extends \(\alpha\) with value of \(x\) that sat’s \(C\) and simulates corresponding strategy of Pebbler (assuming \(C\) has colour blue/1)
\[
\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi:F \vdash \Box} \text{Rev}(G_\pi) + 2
\]

\textit{Proof sketch:}

\textbf{Given:} a res. refutation \( \pi \) of \( F \) with a ref.-graph \( G_\pi \) and \( \text{Rev}(G_\pi) =: k \).

\textbf{AIM:} Give a strategy for Prover in the PD-game under which he has to pay at most \( k \) points.

\textbf{Idea:} Simulate the strategy of Pebbler in the Raz–McKenzie game → a falsifying part. assignment \( \alpha \) of init. clause will be produced

\textbf{Stages of the game:} Pebbluer chooses \( C \) → Prover queries vars. in \( C \) not yet assigned by \( \alpha \) (& extends with Delayer’s answers) until either

1. the clause \( C \) is sat./fals. by \( \alpha \)
   \( \rightarrow \) Prover moves to next stage, simulating the corresponding strategy of Pebbler when \( C \) is given colour \( C \upharpoonright_\alpha \)

2. a variable is given \( * \) by Delayer
   \( \rightarrow \) Prover extends \( \alpha \) with value of \( x \) that sat’s \( C \) and simulates corresponding strategy of Pebbler (assuming \( C \) has colour blue/1)
\[
\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi:F \vdash \Box} \text{Rev}(G_\pi) + 2
\]

*Proof sketch:*

**Given:** a res. refutation \( \pi \) of \( F \) with a ref.-graph \( G_\pi \) and \( \text{Rev}(G_\pi) =: k \).

**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 → a falsifying part. assignment \( \alpha \) of init. clause will be produced

**Stages of the game:** Pebbler chooses \( C \) → Prover queries vars. in \( C \) not yet assigned by \( \alpha \) (& extends with Delayer’s answers) until either

1. the clause \( C \) is sat./fals. by \( \alpha \)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \( C \) is given colour \( C \upharpoonright_\alpha \)

2. a variable is given \( \ast \) by Delayer
   → Prover extends \( \alpha \) with value of \( x \) that sat’s \( C \) and simulates corresponding strategy of Pebbler (assuming \( C \) has colour blue/1)
Tree-CS($F \vdash \square$) ≤ min$_{\pi:F \vdash \square}$ Rev($G_{\pi}$) + 2

Proof sketch:

**Given:** a res. refutation $\pi$ of $F$ with a ref.-graph $G_{\pi}$ and 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 → a falsifying part. assignment $\alpha$ of init. clause will be produced

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

1. the clause $C$ ist sat./fals. by $\alpha$
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when $C$ is given colour $C|_{\alpha}$

2. a variable is given * by Delayer
   → Prover extends $\alpha$ with value of $x$ that sat’s $C$ and simulates corresponding strategy of Pebbler (assuming $C$ has colour blue/1)
\[
\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi:F \vdash \Box} \text{Rev}(G_\pi) + 2
\]

Proof sketch:

**Given:** a res. refutation \( \pi \) of \( F \) with a ref.-graph \( G_\pi \) and \( \text{Rev}(G_\pi) =: k \).

**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 → a falsifying part. assignment \( \alpha \) of init. clause will be produced

**Stages of the game:** Pebbler chooses \( C \) ⟷ Prover queries vars. in \( C \) not yet assigned by \( \alpha \) (& extends with Delayer’s answers) until either

1. the clause \( C \) ist sat./fals. by \( \alpha \)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \( C \) is given colour \( C \upharpoonright \alpha \)

2. a variable is given \(*\) by Delayer
   → Prover extends \( \alpha \) with value of \( x \) that sat’s \( C \) and simulates corresponding strategy of Pebbler (assuming \( C \) has colour blue/1)
Tree-CS($F \vdash \Box$) $\leq \min_{\pi:F \vdash \Box} \text{Rev}(G_\pi) + 2$

Proof sketch:

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

**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 → a falsifying part. assignment $\alpha$ of init. clause will be produced.

**Stages of the game:** Pebbller chooses $C$ $\rightarrow$ Prover queries vars. in $C$ not yet assigned by $\alpha$ (& extends with Delayer’s answers) until either

1. the clause $C$ ist sat./fals. by $\alpha$
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when $C$ is given colour $C \upharpoonright \alpha$

2. a variable is given $\ast$ by Delayer
   $\rightarrow$ Prover extends $\alpha$ with value of $x$ that sat’s $C$ and simulates corresponding strategy of Pebbler (assuming $C$ has colour blue/1)
Tree-CS(\(F \vdash \Box\)) \leq \min_{\pi: F \vdash \Box} \text{Rev}(G_\pi) + 2

Proof sketch:

**Given:** a res. refutation \(\pi\) of \(F\) with a ref.-graph \(G_\pi\) and \(\text{Rev}(G_\pi) =: k\).

**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 → a falsifying part. assignment \(\alpha\) of init. clause will be produced

**Stages of the game:** Pebbler chooses \(C\) → Prover queries vars. in \(C\) not yet assigned by \(\alpha\) (& extends with Delayer’s answers) until either

1. the clause \(C\) ist sat./fals. by \(\alpha\)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \(C\) is given colour \(C\upharpoonright_\alpha\)

2. a variable is given \(*\) by Delayer
   → Prover extends \(\alpha\) with value of \(x\) that sat’s \(C\) and simulates corresponding strategy of Pebbler (assuming \(C\) has colour blue/1)
Tree-CS($F \vdash \Box$) $\leq$ min$_{\pi:F \vdash \Box}$ Rev($G_{\pi}$) + 2

Proof sketch:

**Given:** a res. refutation $\pi$ of $F$ with a ref.-graph $G_{\pi}$ and 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 → a falsifying part. assignment $\alpha$ of init. clause will be produced

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

1. **the clause $C$ ist sat./fals. by $\alpha$**
   $\rightarrow$ Prover moves to next stage, simulating the corresponding strategy of Pebbler when $C$ is given colour $C|_\alpha$

2. **a variable is given $*$ by Delayer**
   $\rightarrow$ Prover extends $\alpha$ with value of $x$ that sat’s $C$ and simulates corresponding strategy of Pebbler (assuming $C$ has colour blue/1)
Tree-CS(\(F \vdash \Box\)) \leq \min_{\pi: F \vdash \Box} \text{Rev}(G_\pi) + 2

Proof sketch:

**Given:** a res. refutation \(\pi\) of \(F\) with a ref.-graph \(G_\pi\) and \(\text{Rev}(G_\pi) =: k\).

**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 → a falsifying part. assignment \(\alpha\) of init. clause will be produced

**Stages of the game:** Pebbler chooses \(C \rightarrow\) Prover queries vars. in \(C\) not yet assigned by \(\alpha\) (& extends with Delayer’s answers) until either

1. the clause \(C\) ist sat./fals. by \(\alpha\)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \(C\) is given colour \(C|_\alpha\)

2. a variable is given \(*\) by Delayer
   → Prover extends \(\alpha\) with value of \(x\) that sat’s \(C\) and simulates corresponding strategy of Pebbler (assuming \(C\) has colour blue/1)
Tree-CS\( (F \vdash \square) \leq \min_{\pi:F \vdash \square} \operatorname{Rev}(G_\pi) + 2 \)

**Proof sketch:**

**Given:** a res. refutation \( \pi \) of \( F \) with a ref.-graph \( G_\pi \) and \( \operatorname{Rev}(G_\pi) =: k \).

**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 → a falsifying part. assignment \( \alpha \) of init. clause will be produced

**Stages of the game:** Pebbler chooses \( C \) → Prover queries vars. in \( C \) not yet assigned by \( \alpha \) (& extends with Delayer’s answers) until either

1. the clause \( C \) ist sat./fals. by \( \alpha \)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \( C \) is given colour \( C \upharpoonright \alpha \)

2. a variable is given \( * \) by Delayer
   → Prover extends \( \alpha \) with value of \( x \) that sat’s \( C \) and simulates corresponding strategy of Pebbler (assuming \( C \) has colour blue/1)
Tree-CS\((F ⊢ □)\) \(≤ \min_{\pi: F ⊢ □} \text{Rev}(G_\pi) + 2\)

Proof sketch:

**Given:** a res. refutation \(\pi\) of \(F\) with a ref.-graph \(G_\pi\) and \(\text{Rev}(G_\pi) =: k\).

**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 → a falsifying part. assignment \(\alpha\) of init. clause will be produced

**Stages of the game:** Pebbler chooses \(C \rightarrow\) Prover queries vars. in \(C\) not yet assigned by \(\alpha\) (& extends with Delayer’s answers) until either

1. the clause \(C\) ist sat./fals. by \(\alpha\)
   → Prover moves to next stage, simulating the corresponding strategy of Pebbler when \(C\) is given colour \(C\upharpoonright_\alpha\)

2. a variable is given \(\ast\) by Delayer
   → Prover extends \(\alpha\) with value of \(x\) that sat’s \(C\) and simulates corresponding strategy of Pebbler (assuming \(C\) has colour blue/1)
Tree-CS($F \vdash \Box$) \leq \min_{\pi:F\vdash \Box} \text{Rev}(G_\pi) + 2

**Proof sketch:**

The game is played until $\alpha$ falsifies a clause in $F$.

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

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

When Raz–McKenzie finishes:

1. either a source vertex in $G_\pi$ is assigned colour 0 by Colourer,
   $\implies$ since $\alpha$ defines Colourer’s answer: $\alpha$ fals. a clause in $F$.

2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   $\implies$ not possible, since no $\alpha$ can sat’y two parent clauses in a
   resolution proof, while falsifying their resolvent!
Tree-CS($F \vdash \square$) ≤ min$_{F \vdash \square}$ Rev($G_{\pi}$) + 2

Proof sketch:

The game is played until $\alpha$ falsifies a clause in $F$.

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

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

When Raz–McKenzie finishes:

1. either a source vertex in $G_{\pi}$ is assigned colour 0 by Colourer,
   → since $\alpha$ defines Colourer’s answer: $\alpha$ fals. a clause in $F$.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   → not possible, since no $\alpha$ can sat’y two parent clauses in a resolution proof, while falsifying their resolvent!
$$\text{Tree-CS}(F \vdash \square) \leq \min_{\pi:F \vdash \square} \text{Rev}(G_\pi) + 2$$

*Proof sketch:*

The game is played until $\alpha$ 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 $\alpha$.

When Raz–McKenzie finishes:

1. either a source vertex in $G_\pi$ is assigned colour 0 by Colourer,
   $\rightarrow$ since $\alpha$ defines Colourer’s answer: $\alpha$ 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 $\alpha$ can sat’y two parent clauses in a resolution proof, while falsifying their resolvent!
Tree-\text{CS}(F \vdash \Box) \leq \min_{\pi: F \vdash \Box} \text{Rev}(G_\pi) + 2

\text{Proof sketch:}

The game is played until \(\alpha\) 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 \(\alpha\).

When Raz–McKenzie finishes:
\begin{enumerate}
\item either a source vertex in \(G_\pi\) is assigned colour 0 by Colourer,
\(\rightarrow\) since \(\alpha\) defines Colourer’s answer: \(\alpha\) fals. a clause in \(F\).
\item or a vertex with all its direct predecessors being coloured 1 is coloured 0.
\(\rightarrow\) not possible, since no \(\alpha\) can sat’y two parent clauses in a resolution proof, while falsifying their resolvent!
\end{enumerate}
Tree-CS($F \vdash \square$) ≤ min$_{\pi:F \vdash \square}$Rev($G_\pi$) + 2

Proof sketch:

The game is played until $\alpha$ falsifies a clause in $F$.

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

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

When Raz–McKenzie finishes:

1. either a source vertex in $G_\pi$ is assigned colour 0 by Colourer,
   → since $\alpha$ defines Colourer’s answer: $\alpha$ fals. a clause in $F$.

2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
   → not possible, since no $\alpha$ can sat’y two parent clauses in a resolution proof, while falsifying their resolvent!

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

Proof sketch:

The game is played until \( \alpha \) 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 \( \alpha \).

When Raz–McKenzie finishes:

1. either a source vertex in \( G_\pi \) is assigned colour 0 by Colourer,
\( \rightarrow \) since \( \alpha \) defines Colourer’s answer: \( \alpha \) 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 \( \alpha \) can sat’y two parent clauses in a resolution proof, while falsifying their resolvent!
Tree-CS($F \vdash \square$) \leq \min_{\pi:F \vdash \square} \text{Rev}(G_\pi) + 2

Proof sketch:

The game is played until $\alpha$ 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 $\alpha$.

When Raz–McKenzie finishes:
1. either a source vertex in $G_\pi$ is assigned colour 0 by Colourer,
   $\rightarrow$ since $\alpha$ defines Colourer’s answer: $\alpha$ 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 $\alpha$ can sat’y two parent clauses in a resolution proof, while falsifying their resolvent!
Tree-CS($F \vdash \Box$) $\leq \min_{\pi:F \vdash \Box} \text{Rev}(G_{\pi}) + 2$

*Proof sketch:*

The game is played until $\alpha$ falsifies a clause in $F$.

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

Only left to show: *At the end of the game a clause of $F$ is fals. by $\alpha$.*

When Raz–McKenzie finishes:

1. either a source vertex in $G_{\pi}$ is assigned colour 0 by Colourer,\n→ since $\alpha$ defines Colourer’s answer: $\alpha$ fals. a clause in $F$.
2. or a vertex with all its direct predecessors being coloured 1 is coloured 0.
→ not possible, since no $\alpha$ can sat'y two parent clauses in a resolution proof, while falsifying their resolvent!
Tree-CS(\(F \vdash \Box\)) \leq \min_{\pi:F \vdash \Box} Rev(G_\pi) + 2

Proof sketch:

The game is played until \(\alpha\) falsifies a clause in \(F\).

After at most \(k\) stages the Raz–McKenzie game finished
⇒ Delayer can score at most \(k\) points.

Only left to show: At the end of the game a clause of \(F\) is fals. by \(\alpha\).

When Raz–McKenzie finishes:

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

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

On the other hand:

\[ \min_{\pi: F \vdash \Box} \text{Rev}(G_\pi) \leq \text{Tree-CS}(F \vdash \Box)(\lceil \log n \rceil + 1) \]

and there are formulas for which this bound is tight.
An upper bound for Tree-CS in terms of CS* 

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

\[ CS^*(F \vdash \Box) := \min_{\pi : F \vdash \Box} (CS(\pi) \cdot \log L(\pi)) \]

Corollary

Tree-CS(F \vdash \Box) \leq CS^*(F \vdash \Box) + 2.

Proof.

• [Královič ’04] \( Rev(G_{\pi}) \leq \min_{\mathcal{P}} (\text{space}(\mathcal{P}) \cdot \log \text{time}(\mathcal{P})) \), where the minimum is taken over all black pebblings \( \mathcal{P} \) of \( G_{\pi} \).

• Every black pebbling \( \mathcal{P} \) of \( G_{\pi} \) defines a configurational refutation of \( F \) with clause space equal to \( \text{space}(\mathcal{P}) \) and length \( \text{time}(\mathcal{P}) \). \( \square \)
An upper bound for Tree-CS in terms of CS*

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

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

Corollary

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

Proof.

- [Královič ’04] $\text{Rev}(G_{\pi}) \leq \min_{\mathcal{P}} (\text{space}(\mathcal{P}) \cdot \log \text{time}(\mathcal{P}))$, where the minimum is taken over all black pebblings $\mathcal{P}$ of $G_{\pi}$.
- Every black pebbling $\mathcal{P}$ of $G_{\pi}$ defines a configurational refutation of $F$ with clause space equal to $\text{space}(\mathcal{P})$ and length $\text{time}(\mathcal{P})$. □
An upper bound for Tree-CS in terms of $\text{CS}^*$

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

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

**Corollary**

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

**Proof.**

- [Královič '04] $\text{Rev}(G_\pi) \leq \min_\mathcal{P} (\text{space}(\mathcal{P}) \cdot \log \text{time}(\mathcal{P}))$, where the minimum is taken over all black pebblings $\mathcal{P}$ of $G_\pi$.

- Every black pebbling $\mathcal{P}$ of $G_\pi$ defines a configurational refutation of $F$ with clause space equal to $\text{space}(\mathcal{P})$ and length $\text{time}(\mathcal{P})$. $\square$
An upper bound for Tree-CS in terms of CS* 

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

$$\text{CS}^*(F \vdash \Box) := \min_{\pi: F \vdash \Box} \left( \text{CS}(\pi) \cdot \log L(\pi) \right)$$

**Corollary**

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

**Proof.**

- [Královič '04] $\text{Rev}(G_\pi) \leq \min_\mathcal{P} \left( \text{space}(\mathcal{P}) \cdot \log \text{time}(\mathcal{P}) \right)$, where the minimum is taken over all black pebblings $\mathcal{P}$ of $G_\pi$.
- Every black pebbling $\mathcal{P}$ of $G_\pi$ defines a configurational refutation of $F$ with clause space equal to $\text{space}(\mathcal{P})$ and length $\text{time}(\mathcal{P})$. \qed
How large can be the gap between CS and Tree-CS?
Pebbling Formulas (formulas over DAGs)
Pebbling Formula

Clauses of $\text{Peb}_G$:

$u$

$v$

$w$

$(u \land v) \rightarrow x = \overline{u} \lor \overline{v} \lor x$

$(v \land w) \rightarrow y = \overline{v} \lor \overline{w} \lor y$

$(x \land y) \rightarrow z = \overline{x} \lor \overline{y} \lor z$

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 $\text{Peb}_G$:

$u$

$v$

$w$

$(u \land v) \rightarrow x = \overline{u} \lor \overline{v} \lor x$

$(v \land w) \rightarrow y = \overline{v} \lor \overline{w} \lor y$

$(x \land y) \rightarrow z = \overline{x} \lor \overline{y} \lor z$

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 $\text{Peb}_G$:

$u$
$v$
$w$

$(u \land v) \rightarrow x = \overline{u} \lor \overline{v} \lor x$
$(v \land w) \rightarrow y = \overline{v} \lor \overline{w} \lor y$
$(x \land y) \rightarrow z = \overline{x} \lor \overline{y} \lor z$
$

$\overline{z}$

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 $\text{Peb}_G$:

- $u$
- $v$
- $w$
- $(u \land v) \rightarrow x = \overline{u} \lor \overline{v} \lor x$
- $(v \land w) \rightarrow y = \overline{v} \lor \overline{w} \lor y$
- $(x \land y) \rightarrow z = \overline{x} \lor \overline{y} \lor z$

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:

$$\text{Rev}(G) + 2 \leq \text{Tree-CS}(\text{Peb}_G[\oplus_2] \vdash \square) \leq 2 \cdot \text{Rev}(G) + 2.$$
Obtaining Space-Separations with Pebble games

Idea:

- $\text{CS}(\text{Peb}_G[\oplus 2] \vdash \square) = O(\text{Black}(G))$
- $\text{Tree-CS}(\text{Peb}_G[\oplus 2] \vdash \square) = \Omega(\text{Rev}(G))$

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

Example: Path graphs $P_n$ of length $n$

- $\text{Black}(P_n) = O(1) \ \forall n \in \mathbb{N}$
- $\text{Rev}(P_n) = \Theta(\log n) \ \forall n \in \mathbb{N}$
Obtaining Space-Separations with Pebble games

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

\[ G(c = 3, k) \]
The best known separation

For “slowly enough” growing space functions $s(n)$ there is a family of pebbling formulas $(\text{Peb}_{G_n} [\oplus 2])_{n=1}^{\infty}$ with $\Theta(n)$ variables such that

- $\text{CS}(\text{Peb}_{G_n} [\oplus 2] \vdash \Box) = O(s(n))$
- $\text{Tree-CS}(\text{Peb}_{G_n} [\oplus 2] \vdash \Box) = \Omega(s(n) \log n)$.

¿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 $\chi$
  \[ \text{Tree-CS} \left( \text{Ts}(G, \chi) \vdash \Box \right) \leq \text{CS} \left( \text{Ts}(G, \chi) \vdash \Box \right) \cdot \log n + 2 \]
- There are graph families $\{G_n\}$ for which $\forall n:$
  \[ \text{Tree-CS} \left( \text{Ts}(G, \chi) \vdash \Box \right) = \Omega \left( \text{CS} \left( \text{Ts}(G, \chi) \vdash \Box \right) \cdot \log n \right) \]
\[ \text{Tree-CS}(\text{Ts}(G, \chi) \vdash \square) \leq \text{CS}(\text{Ts}(G, \chi) \vdash \square) \cdot \log n + 2 \]

Proof sketch:

Let \( \pi = (M_0, \ldots, M_t) \) be a refutation of \( \text{Ts}(G, \chi) \) with \( \text{CS}(\pi) =: k \). We use \( \pi \) 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 \( \alpha \) of some of the variables in \( \text{Ts}(G, \chi) \) is non-splitting if after applying \( \alpha \) 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 \( \pi \) for which there is a partial assignment \( \alpha \) fulfilling:

(i) \( \alpha \) simultaneously satisfies all clauses in \( M_s \) and

(ii) \( \alpha \) is non-splitting.

The only new clause in configuration \( M_{s+1} \) must be an axiom of \( \text{Ts}(G, \chi) \).
Tree-CS(Ts(G, χ) ⊢ □) ≤ CS(Ts(G, χ) ⊢ □) · log n + 2

Proof sketch:

Let π = (M_0, . . . , M_t) be a refutation of Ts(G, χ) with CS(π) =: 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 Ts(G, χ) 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 M_s and
(ii) α is non-splitting.

The only new clause in configuration M_{s+1} must be an axiom of Ts(G, χ)
\[ \text{Tree-CS} \left( \text{Ts}(G, \chi) \vdash \square \right) \leq \text{CS} \left( \text{Ts}(G, \chi) \vdash \square \right) \cdot \log n + 2 \]

**Proof sketch:**

Let \( \pi = (M_0, \ldots, M_t) \) be a refutation of \( \text{Ts}(G, \chi) \) with \( \text{CS}(\pi) =: k \).

We use \( \pi \) 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 \( \alpha \) of some of the variables in \( \text{Ts}(G, \chi) \) is **non-splitting** if after applying \( \alpha \) 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 \( \pi \) for which there is a partial assignment \( \alpha \) fulfilling:

(i) \( \alpha \) simultaneously satisfies all clauses in \( M_s \) and

(ii) \( \alpha \) is non-splitting.

The only new clause in configuration \( M_{s+1} \) must be an axiom of \( \text{Ts}(G, \chi) \).
Tree-CS\((\text{Ts}(G, \chi) \vdash \square) \leq \text{CS}(\text{Ts}(G, \chi) \vdash \square) \cdot \log n + 2\)

**Proof sketch:**

Let \(\pi = (M_0, \ldots, M_t)\) be a refutation of \(\text{Ts}(G, \chi)\) with \(\text{CS}(\pi) =: k\).

We use \(\pi\) 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 \(\alpha\) of some of the variables in \(\text{Ts}(G, \chi)\) is **non-splitting** if after applying \(\alpha\) 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 \(\pi\) for which there is a partial assignment \(\alpha\) fulfilling:

(i) \(\alpha\) simultaneously satisfies all clauses in \(M_s\) and

(ii) \(\alpha\) is non-splitting.

The only new clause in configuration \(M_{s+1}\) must be an axiom of \(\text{Ts}(G, \chi)\)
Tree-CS(Ts(G, χ) ⊢ □) ≤ CS(Ts(G, χ) ⊢ □) · log n + 2

Proof sketch:

A partial assignment $\alpha$ of some of the variables in $Ts(G, χ)$ is non-splitting if after applying $\alpha$ 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 $\pi$ for which there is a partial assignment $\alpha$ fulfilling:

(i) $\alpha$ simultaneously satisfies all clauses in $M_s$ and

(ii) $\alpha$ is non-splitting.

The only new clause in configuration $M_{s+1}$ must be an axiom of $Ts(G, χ)$

There is a way to query variables at stage $s + 1$ paying only $k$ points to Delayer and splitting $G$ or falsifying the axiom.
Tree-CS\((Ts(G, \chi) \vdash \Box)\) \leq CS\((Ts(G, \chi) \vdash \Box)\) \cdot \log n + 2

Proof sketch:

A partial assignment \(\alpha\) of some of the variables in \(Ts(G, \chi)\) is **non-splitting** if after applying \(\alpha\) 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 \(\pi\) for which there is a partial assignment \(\alpha\) fulfilling:

(i) \(\alpha\) simultaneously satisfies all clauses in \(M_s\) and

(ii) \(\alpha\) is non-splitting.

The only new clause in configuration \(M_{s+1}\) must be an axiom of \(Ts(G, \chi)\)

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

Tree-CS and CS are different measures but “not too far” from one another

- Tree-CS\((Peb_G[⊕_2] \vdash □)\) ≃ Rev(G)
- Separations between Tree-CS and CS by graphs G exhibiting separation between Rev(G) and Black(G)
- Tree-CS\((F \vdash □)\) ≲ CS*\((F \vdash □)\) for general F
- Tree-CS\((F \vdash □)\) ≲ VS*\((F \vdash □)\) for general F
Take-Home Message

Tree-CS and CS are different measures but “not too far” from one another

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

(*) Some open questions hidden here. We’ve solved these for Tseitin formulas.
Take-Home Message

Tree-CS and CS are different measures but “not too far” from one another

- Tree-CS\(\left(Peb_G[⊕_2] ⊢ □\right)\) \(≃\) Rev\(\left(G'\right)\)
- Separations between Tree-CS and CS by graphs \(G\) exhibiting separation between Rev\(\left(G\right)\) and Black\(\left(G\right)\) (*)
- Tree-CS\(\left(F ⊢ □\right)\) \(≲\) CS\(^*\)\(\left(F ⊢ □\right)\) for general \(F\) (*)
- Tree-CS\(\left(F ⊢ □\right)\) \(≲\) VS\(^*\)\(\left(F ⊢ □\right)\) for general \(F\) (*)

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

Thank you for your attention!