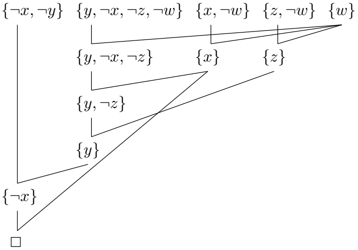
Reversible Pebble Games and the Relation Between Tree-Like and General Resolution Space

Jacobo Torán & Florian Wörz
Universität Ulm

Dagstuhl Seminar „*SAT and Interactions*“
February 6, 2020

Just to Check We Are on the Same Page...

Proof π



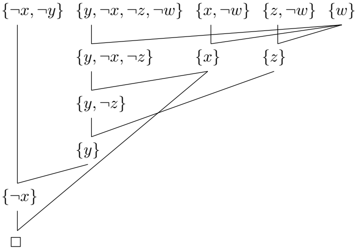
- The proof system Resolution has only one derivation rule:

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

- Complexity Measures:
  - Length of π = # of clauses in π
  - Clause Space of $\pi = \max \#$ of clauses in memory simultaneously during π
  - $\mu(F \vdash \Box) := \min_{\pi : F \vdash \Box} \mu(\pi)$.
  - Prefix "Tree-" before a complexity measure indicates tree-like resolution.

Just to Check We Are on the Same Page...

Proof π



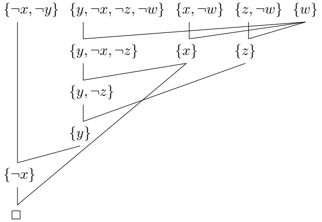
- The proof system Resolution has only one derivation rule:

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

- Complexity Measures:
  - Length of $\pi = \#$ of clauses in π
  - Clause Space of $\pi = \max \#$ of clauses in memory simultaneously during π
  - $\mu(F \vdash \square) := \min_{\pi : F \vdash \square} \mu(\pi)$.
  - Prefix "Tree-" before a complexity measure indicates tree-like resolution.

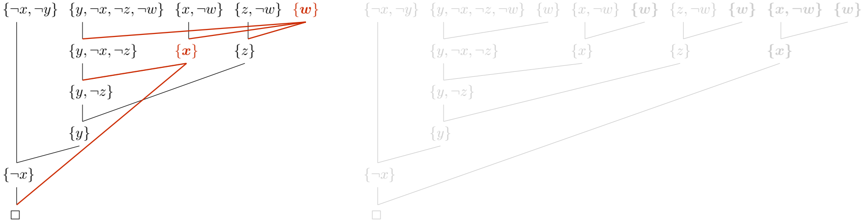
Tree-like vs. General Resolution Refutations

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



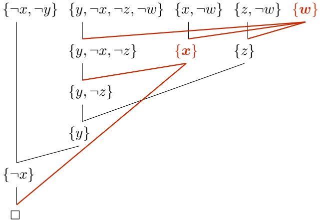
Tree-like vs. General Resolution Refutations

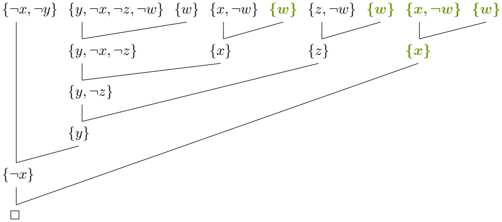
If a clause is needed more than once in a refutation, it has to be rederived each time.



Tree-like vs. General Resolution Refutations

If a clause is needed more than once in a refutation, it has to be rederived each time. ($\rightarrow$ refutation DAG G_π becomes a tree)





After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know 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.

¿What can we say about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know 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.

¿What can we say about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know 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.

¿What can we say about space?

After We've Set the Stage: Motivation of This Talk

Thanks to [Ben-Sasson, Impagliazzo, Wigderson '04: Near optimal separation...] we know 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 $\exp\big(\Omega(\frac{L}{\log L})\big)$.

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

¿What can we say about space?

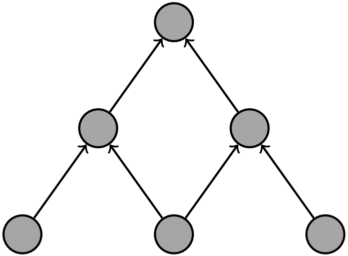
Part I

Separations for Pebbling Formulas

Pebble Games (games played on graphs)

The Black Pebble Game

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

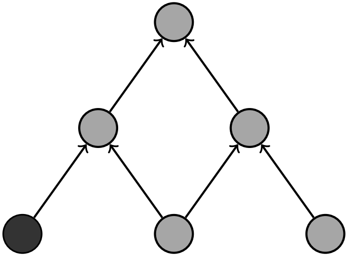


$\max$ # of pebbles used at any point:

- **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.



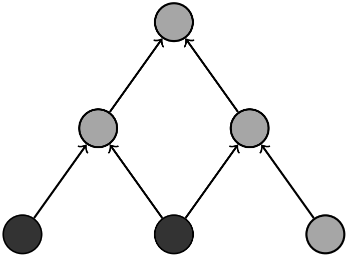
max # of pebbles used at any point:

1

- **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.

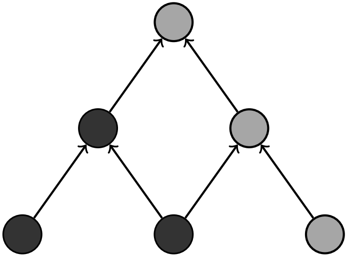


$\max$ # of pebbles
used at any point:
II

- **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.

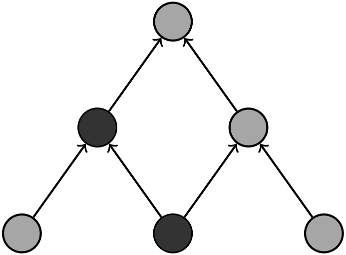


$\max$ # of pebbles
used at any point:
III

- **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.



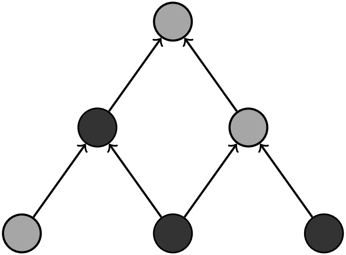
max # of pebbles used at any point:

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.



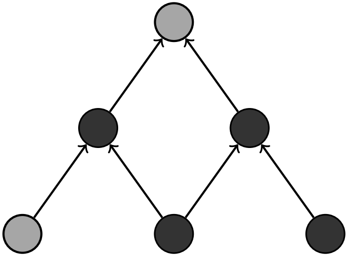
$\max$ # of pebbles used at any point:

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.

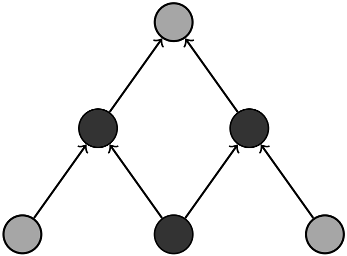


max # of pebbles
used at any point:
IIII

- **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.

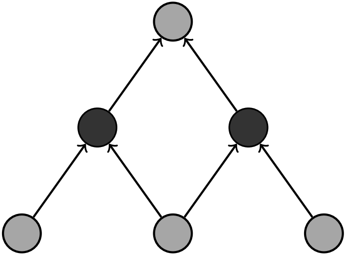


max # of pebbles
used at any point:
IIII

- **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.



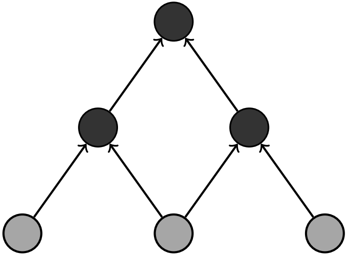
$\max$ # of pebbles used at any point:

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.

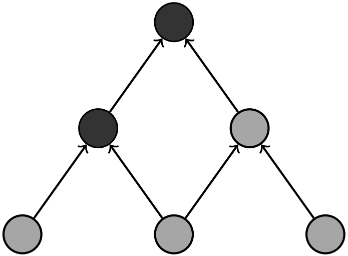


max # of pebbles used at any point:
IIII

- **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.

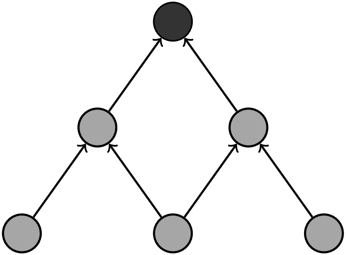


$\max$ # of pebbles
used at any point:
IIII

- **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.

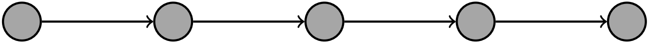


max # of pebbles
used at any point:
IIII

- **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:

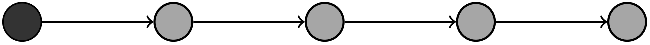


Different rules:

- **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

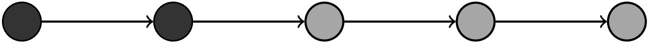


Different rules:

- **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

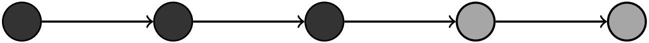


Different rules:

- **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

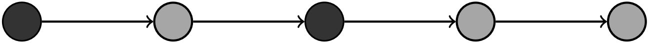


Different rules:

- **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

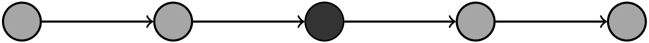


Different rules:

- **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

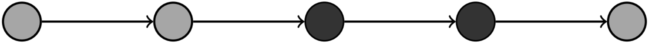


Different rules:

- **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

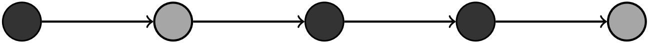


Different rules:

- **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

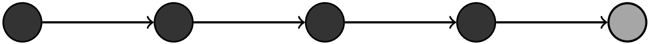


Different rules:

- **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

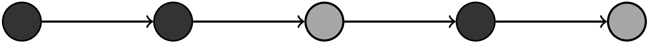


Different rules:

- **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

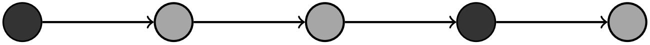


Different rules:

- **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

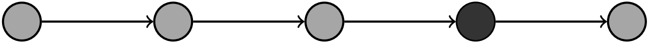


Different rules:

- **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

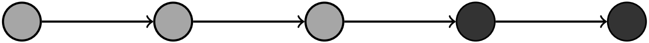


Different rules:

- **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

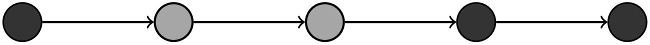


Different rules:

- **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

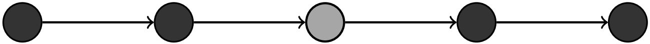


Different rules:

- **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

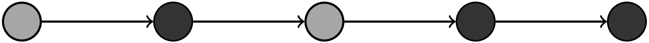


Different rules:

- **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

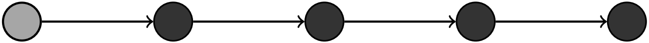


Different rules:

- **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

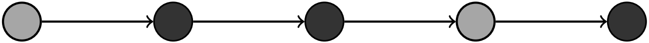


Different rules:

- **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

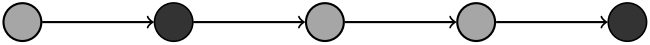


Different rules:

- **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

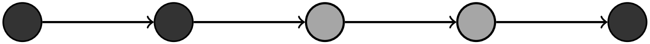


Different rules:

- **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

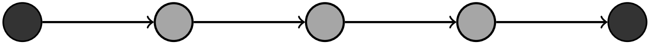


Different rules:

- **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

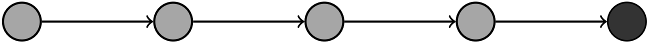


Different rules:

- **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

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

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

Why even care about these pebbling prices?

$\rightsquigarrow$ Plethora connections to resolution! [Nordström '15: New Wine . . .]

$$\mathrm{CS}(F \vdash \Box) = \min_{\pi : F \vdash \Box} \text{Black}(G_\pi)$$ [Esteban, Torán '01: Space bdds. for res.]

Complexity Measures for the Pebble Games

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

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

Why even care about these pebbling prices?

$\rightsquigarrow$ Plethora connections to resolution! [Nordström '15: New Wine . . .]

$$\mathrm{CS}(F \vdash \Box) = \min_{\pi : F \vdash \Box} \mathsf{Black}(G_\pi)$$

[Esteban, Torán '01: Space bdds. for res.]

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

Pebbling Formula

Clauses of Peb_G:

u

v

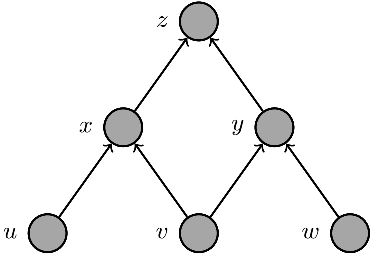
w

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

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

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

$\overline{z}$



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 Peb_G:

u

v

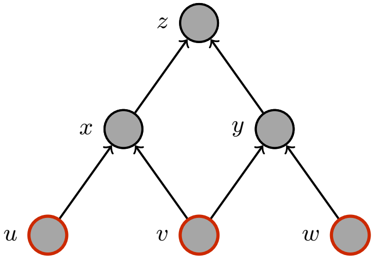
w

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

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

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

$\overline{z}$



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

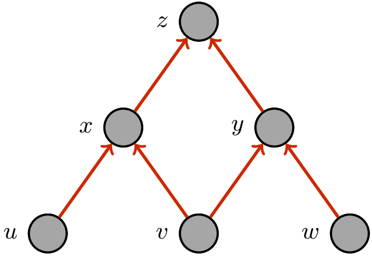
w

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

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

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

$\overline{z}$



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 Peb_G:

u

v

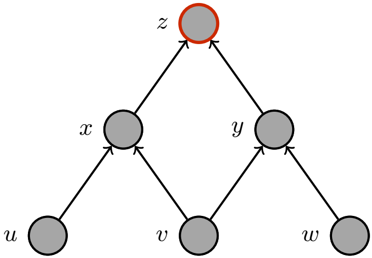
w

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

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

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

$\overline{z}$



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

Reversible Pebbling meets Tree-CS

Theorem

For all DAGs G with a unique sink:

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

Obtaining Space-Separations with Pebble games (1/3)

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

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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

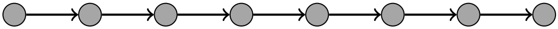
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

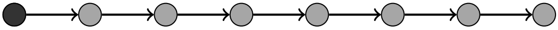
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

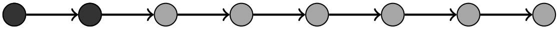
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

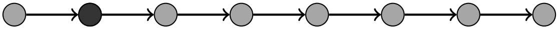
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

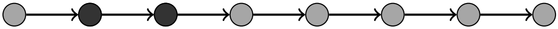
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

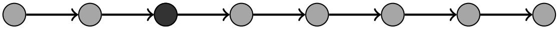
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

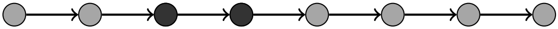
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

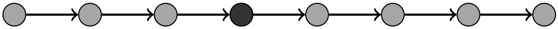
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$
  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

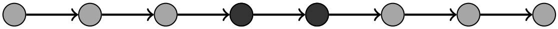
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

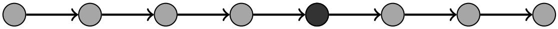
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

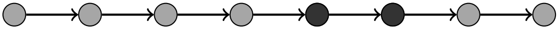
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

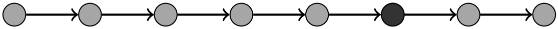
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

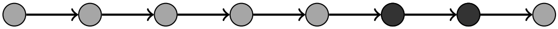
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

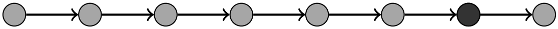
Obtaining Space-Separations with Pebble games (1/3)

Idea:

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

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

Example: Path graphs P_n of length n



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

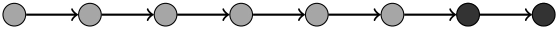
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

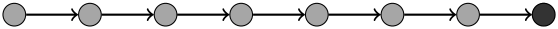
Obtaining Space-Separations with Pebble games (1/3)

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



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

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

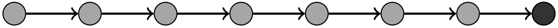
Obtaining Space-Separations with Pebble games (1/3)

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

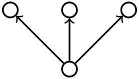


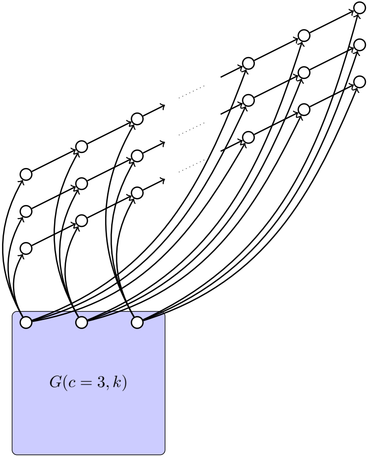
- $\mathsf{Black}(P_n) = \mathrm{O}(1)\ \forall n \in \mathbb{N}$ $\quad\exists$ Results for non-const. space?
- $\mathsf{Rev}(P_n) = \Theta(\log n)\ \forall n \in \mathbb{N}$

  [Bennett '89: Time/space trade-offs for reversible computation; Li, Vitányi '96: Reversibility and adiabatic computation: Trading time and space for energy]

Obtaining Space-Separations with Pebble games (2/3)

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





Obtaining Space-Separations with Pebble games (3/3)

Conclusion: The best known separation

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

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

¿Can we do any better?

Obtaining Space-Separations with Pebble games (3/3)

Conclusion: The best known separation

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

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

¿Can we do any better?

Part II

Upper Bounds for Tree-CS for General Formulas

An upper bound for Tree-CS

How large can the gap between CS and Tree-CS grow?

Theorem

For any unsatisfiable formula F with n variables it holds (2nd ineq. is tight)

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

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

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

Recall: $\boxed{\text{CS}(F \vdash \Box) = \min_{\pi : F \vdash \Box} \mathsf{Black}(G_\pi)}$ [ET01] $\rightsquigarrow$ Similarity to Thm!

We will now prove the inequality of the Theorem...

An upper bound for Tree-CS

How large can the gap between CS and Tree-CS grow?

Theorem

For any unsatisfiable formula F with n variables it holds (2nd ineq. is tight)

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

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

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

Recall: $\text{CS}(F \vdash \Box) = \min_{\pi : F \vdash \Box} \mathsf{Black}(G_\pi)$ [ET01] $\rightsquigarrow$ Similarity to Thm!

We will now prove the inequality of the Theorem...

An upper bound for $\mathrm{Tree\text{-}CS}$

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

Theorem

For any unsatisfiable formula F with n variables it holds (2nd ineq. is tight)

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

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

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

Recall: $\boxed{\mathrm{CS}(F \vdash \Box) = \min_{\pi : F \vdash \Box} \mathsf{Black}(G_\pi)}$ [ET01] $\rightsquigarrow$ Similarity to Thm!

We will now prove the inequality of the Theorem...

An upper bound for Tree-CS

How large can the gap between CS and Tree-CS grow?

Theorem

For any unsatisfiable formula F with n variables it holds (2nd ineq. is tight)

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

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

Note, that the minimum in the theorem is taken over all possible refutations of F, not only over the tree-like ones.

Recall: $\boxed{\text{CS}(F \vdash \Box) = \min_{\pi : F \vdash \Box} \mathsf{Black}(G_\pi)}$ [ET01] $\rightsquigarrow$ Similarity to Thm!

We will now prove the inequality of the Theorem...

A combinatorial characterization of Tree-CS (by a game played on formulas)

The Prover-Delayer Game

[Pudlák, Impagliazzo '00: A lower bound for DLL algorithms for k-SAT]

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...

Score of Delayer = # of $*$'s

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 $*$	

The Prover-Delayer Game

[Pudlák, Impagliazzo '00: A lower bound for DLL algorithms for k-SAT]

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...

Score of Delayer = # of $*$'s

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 $*$	

The Prover-Delayer Game

[Pudlák, Impagliazzo '00: A lower bound for DLL algorithms for k-SAT]

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...

Score of Delayer = # of $*$'s

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 $*$	

The Prover-Delayer Game

[Pudlák, Impagliazzo '00: A lower bound for DLL algorithms for k-SAT]

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...

Score of Delayer = # of $*$'s

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 $*$	

The Prover-Delayer Game

[Pudlák, Impagliazzo '00: A lower bound for DLL algorithms for k-SAT]

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...

Score of Delayer = # of $*$'s

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 $*$	

The Prover-Delayer Game

[Pudlák, Impagliazzo '00: A lower bound for DLL algorithms for k-SAT]

Given: An unsatisfiable CNF formula F

Two players take rounds... until Game Over...
Score of Delayer = # of $*$'s

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 $*$	

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, Torán '03: A combinatorial char. of treelike res. space])

Let F be an unsatisfiable CNF formula. Then

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

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, Torán '03: A combinatorial char. of treelike res. space])

Let F be an unsatisfiable CNF formula. Then

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

The equivalence of Rev and R-Mc

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

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

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

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

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

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

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

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

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

Given: A single sink DAG G

Two players take rounds... until Game Over...

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

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

Given: A single sink DAG G

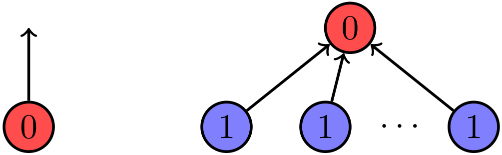
Two players take rounds... until Game Over...

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

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

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



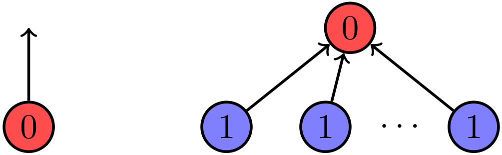
Either a red source **or** red vertex with all predecessors blue.

R-Mc(G) := smallest r s. th. Pebbler wins in $\leq r$ rounds
against an optimal strategy of Colourer

Reversible pebbling is hard to analyse

Raz–McKenzie Game to the help

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



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

R-Mc(G) := smallest r s. th. Pebbler wins in $\leq r$ rounds
against an optimal strategy of Colourer

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

Theorem ([Chan '13: Just a pebble game])

For any single-sink DAG G:

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

The Actual Proof

An upper bound for Tree-CS

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

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)

An upper bound for Tree-CS

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

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)

An upper bound for Tree-CS

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

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)

An upper bound for Tree-CS

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

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
$\to$ 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
   $\to$ 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
   $\to$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

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

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
$\to$ 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
   $\to$ 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
   $\to$ Prover extends α with value of x that sat's C and simulates corresponding strategy of Pebbler (assuming C has colour blue/1)

An upper bound for Tree-CS

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

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)

An upper bound for Tree-CS

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

Given: a res. refutation π of F with a ref.-graph G_π 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
$\rightarrow$ 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)

An upper bound for Tree-CS

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

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! $\Box$

An upper bound for Tree-CS

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

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! $\Box$

An upper bound for Tree-CS

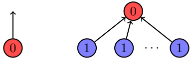
Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

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! $\Box$



An upper bound for Tree-CS

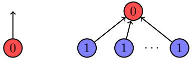
Proof sketch of $\mathrm{Tree\text{-}CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

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! $\Box$



An upper bound for Tree-CS

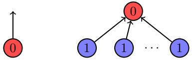
Proof sketch of $\text{Tree-CS}(F \vdash \Box) \leq \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi) + 2$

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! $\Box$



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

[Razborov '18: On space and depth in resolution] introduced 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: Time and Space Complexity of Reversible Pebbling] $\mathsf{Rev}(G_\pi) + 2 \leq \min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big) + 2$, 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 $\mathsf{time}(\mathcal{P})$. □

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

[Razborov '18: On space and depth in resolution] introduced 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: Time and Space Complexity of Reversible Pebbling] Rev$(G_\pi) + 2 \leq \min_{\mathcal{P}} \big(\text{space}(\mathcal{P}) \cdot \log \text{time}(\mathcal{P})\big) + 2$, 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: On space and depth in resolution] introduced amortised clause space:

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

Corollary

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

Proof.

- [Královič '04: Time and Space Complexity of Reversible Pebbling] $\mathsf{Rev}(G_\pi) + 2 \le \min_{\mathcal{P}} \big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\big) + 2$, 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 $\mathsf{time}(\mathcal{P})$. $\Box$

Part III

The Tseitin Formula Case

Better Bounds for Tseitin Formulas

Tseitin Formulas: "Sum of degrees of vertices in a graph is even"

A quick Recap: We have just seen

$$\text{Tree-CS}(F \vdash \Box) \lesssim \min_{\pi : F \vdash \Box} \Big(\text{CS}(\pi) \cdot \boxed{\log \text{L}(\pi)}\Big).$$

Theorem: Matching Upper and Lower Bounds for Tseitin Formulas

- For any connected graph G with n vertices and odd marking χ:

$$\text{Tree-CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \lesssim \text{CS}\big(\text{Ts}(G,\chi) \vdash \Box\big) \cdot \boxed{\log n}.$$

- $\exists$ a family of Tseitin formulas $\big(\text{Ts}(G_n,\chi_n)\big)_{n=1}^{\infty}$ s. th. $\forall n \in \mathbb{N}$:

$$\text{Tree-CS}\big(\text{Ts}(G_n,\chi_n) \vdash \Box\big) = \Omega\Big(\text{CS}\big(\text{Ts}(G_n,\chi_n) \vdash \Box\big) \cdot \log n\Big).$$

Better Bounds for Tseitin Formulas

Tseitin Formulas: "Sum of degrees of vertices in a graph is even"

A quick Recap: We have just seen

$$\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \min_{\pi : F \vdash \Box} \Big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\Big).$$

Theorem: Matching Upper and Lower Bounds for Tseitin Formulas

- For any connected graph G with n vertices and odd marking χ:

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

- $\exists$ a family of Tseitin formulas $\big(\mathrm{Ts}(G_n,\chi_n)\big)_{n=1}^{\infty}$ s. th. $\forall n \in \mathbb{N}$:

$$\mathrm{Tree\text{-}CS}\big(\mathrm{Ts}(G_n,\chi_n) \vdash \Box\big) = \Omega\Big(\mathrm{CS}\big(\mathrm{Ts}(G_n,\chi_n) \vdash \Box\big) \cdot \log n\Big).$$

Open Questions and Conjectures

Is it possible to improve

$$\text{Tree-CS}(F \vdash \Box) \lesssim \min_{\pi : F \vdash \Box} \Big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi) \Big).$$

to

$$\text{Tree-CS}\big(F \vdash \Box\big) \lesssim \mathrm{CS}\big(F \vdash \Box\big) \cdot \log n,$$

where $n = \#$ vertices / formula size?

Conjecture / Strong gut feeling

Yes (and we know: this is the only room for improvement).

Open Questions and Conjectures

Is it possible to improve

$$\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \min_{\pi : F \vdash \Box} \Big(\mathrm{CS}(\pi) \cdot \log \mathrm{L}(\pi)\Big).$$

to

$$\mathrm{Tree\text{-}CS}\big(F \vdash \Box\big) \lesssim \mathrm{CS}\big(F \vdash \Box\big) \cdot \log n,$$

where $n = \#$ vertices / formula size?

Conjecture / Strong gut feeling

Yes (and we know: this is the only room for improvement).

Related Question in the World of Pebbling

Can [Královič]'s bound

$$\mathsf{Rev}(G) \leq \min_{\mathcal{P}} \Big(\mathsf{space}(\mathcal{P}) \cdot \log \mathsf{time}(\mathcal{P})\Big)$$

be improved to

$$\mathsf{Rev}(G) \leq \min_{\mathcal{P}} \Big(\mathsf{space}(\mathcal{P}) \cdot \log |V(G)|\Big) \quad ?$$

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 $\mathsf{Black}(G)$
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \min\limits_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi)$
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ 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 $\mathsf{Black}(G)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ 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 $\mathsf{Black}(G)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \min_{\pi : F \vdash \Box} \mathsf{Rev}(G_\pi)$ (*)
- $\mathrm{Tree\text{-}CS}(F \vdash \Box) \lesssim \mathrm{CS}^*(F \vdash \Box)$ for general F (*)

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

Thank you for your attention!