Tutte Colloquium Winter 2026

Parallel Reversible Pebbling: Time-Space Tradeoffs on DAGs (with Cryptographic Motivation)

Seunghoon Lee (Joint work with Jeremiah Blocki and Blake Holman)

March 13, 2026

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS Department of Combinatorics and Optimization

PURDUE UNIVERSITY

UNIVERSITY OF WATERLOO

FACULTY OF MATHEMATICS
Department of Combinatorics and Optimization

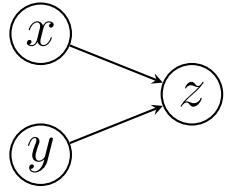
UNIVERSITY OF WATERLOO

FACULTY OF MATHEMATICS
Department of Combinatorics and Optimization

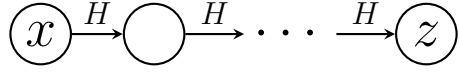
Function Evaluation

Evaluating a function means executing a dependency DAG

$$f(x, y) = xy \quad \Rightarrow$$

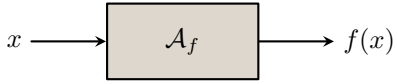


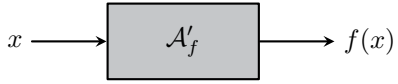
$$f(x) = H(H(\cdots H(x))) \quad \Rightarrow$$

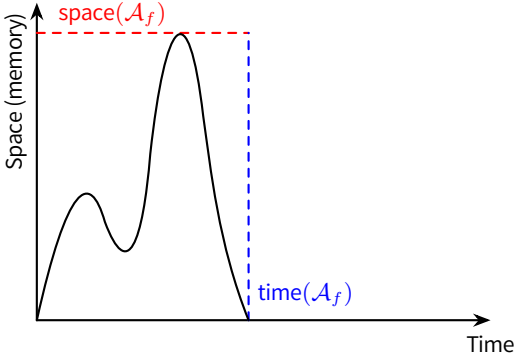


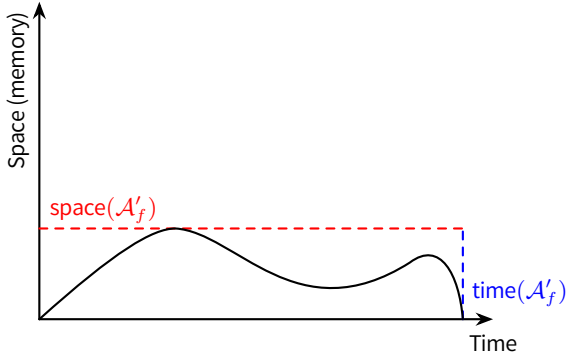
Compute + store intermediates (memory)

DAG Evaluation forces a memory-recomputation tradeoff



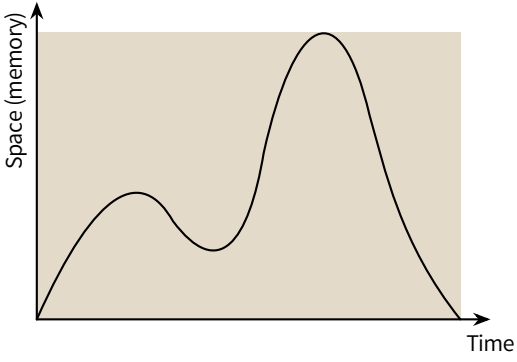






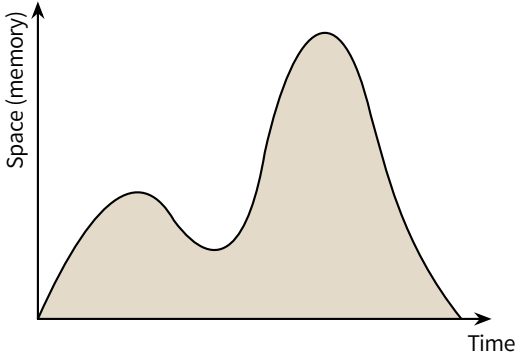
Space–time and cumulative cost capture this tradeoff

[Space-Time Cost]



(peak memory) x (time)

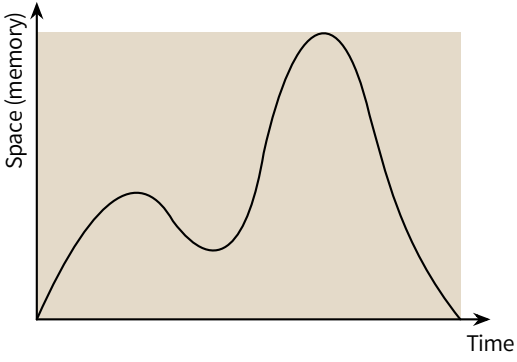
[Cumulative Cost]



Total memory over time

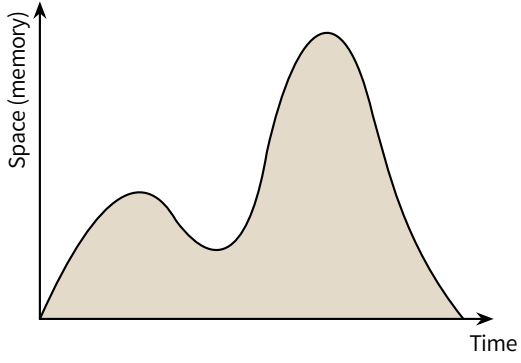
Space–time and cumulative cost capture this tradeoff

[Space-Time Cost]



(peak memory) x (time)

[Cumulative Cost]



Total memory over time

Pebbling games are the combinatorial model for these costs

Three takeaways

Parallel Reversible Pebbling

LANGUAGE **a clean combinatorial way** to reason about time-space tradeoffs **on DAGs (directed acyclic graphs)**

Upper and Lower Bounds

CONCRETE BOUNDS **upper/lower bounds** for reversible pebbling on simple and structured DAGs

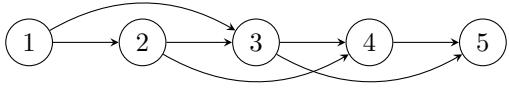
Generic Transformations

REUSABLE TOOL a **generic transformation** from classical to reversible pebbling strategies

Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G

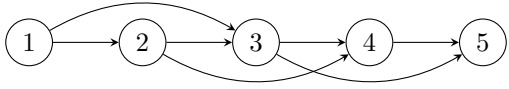


Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G

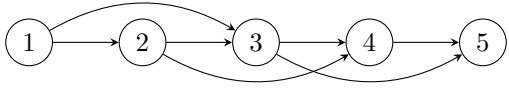


Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**
- **All the parents** need to be previously pebbled to place a **new pebble**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G

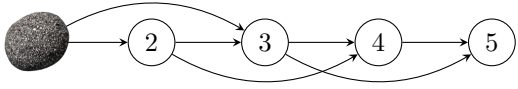


Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**
- **All the parents** need to be previously pebbled to place a **new pebble**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G



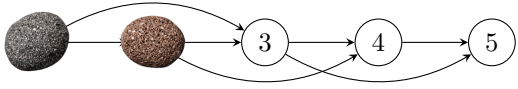
$P_1 = \{1\}$

Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**
- **All the parents** need to be previously pebbled to place a **new pebble**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G



$P_1 = \{1\}$
$P_2 = \{1, 2\}$

Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**
- **All the parents** need to be previously pebbled to place a **new pebble**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G

$$P_1 = \{1\}$$
$$P_2 = \{1, 2\}$$
$$P_3 = \{2, 3\}$$

Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**
- **All the parents** need to be previously pebbled to place a **new pebble**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G

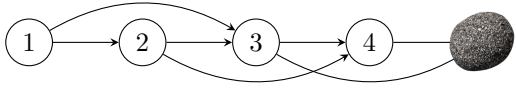
$$P_1 = \{1\}$$
$$P_2 = \{1, 2\}$$
$$P_3 = \{2, 3\}$$
$$P_4 = \{3, 4\}$$

Pebbles model memory in DAG evaluation

(Classical) Pebbling Game Rules

- **Start with no pebbles** on the graph and **end with the target nodes**
- **All the parents** need to be previously pebbled to place a **new pebble**

Example. A pebbling $P = (P_1, \ldots, P_t)$ for a DAG G

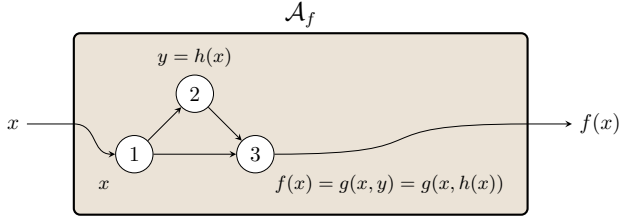


$$P_1 = \{1\}$$
$$P_2 = \{1, 2\}$$
$$P_3 = \{2, 3\}$$
$$P_4 = \{3, 4\}$$
$$P_5 = \{5\}$$

Quantum computation reveals the missing piece: cleanup

Classical pebbling deletes for free; reversible computation cannot

Suppose we have $f(x) := g(x, h(x))$ for some function g, h.



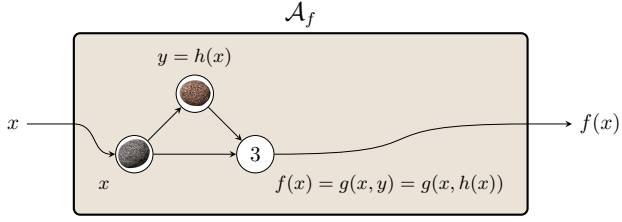
Classically:

$$(x, y, 0) \xmapsto{\text{deletion is free}} (x, 0, f(x))$$

Quantum computation reveals the missing piece: cleanup

Classical pebbling deletes for free; reversible computation cannot

Suppose we have $f(x) := g(x, h(x))$ for some function g, h.



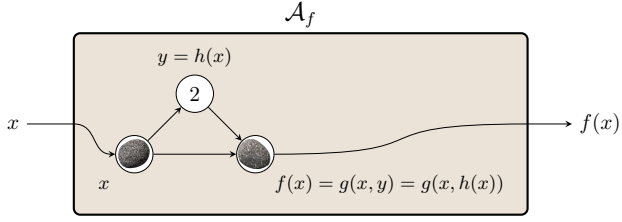
Classically:

$$(x, y, 0) \xmapsto{\text{deletion is free}} (x, 0, f(x))$$

Quantum computation reveals the missing piece: cleanup

Classical pebbling deletes for free; reversible computation cannot

Suppose we have $f(x) := g(x, h(x))$ for some function g, h.



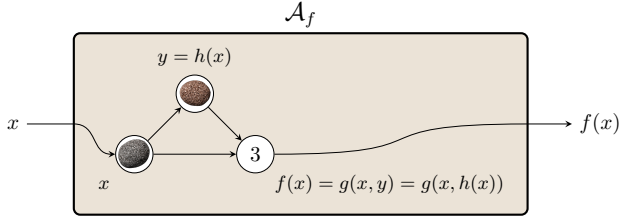
Classically:

$$(x, y, 0) \xmapsto{\text{deletion is free}} (x, 0, f(x))$$

Quantum computation reveals the missing piece: cleanup

Classical pebbling deletes for free; reversible computation cannot

Suppose we have $f(x) := g(x, h(x))$ for some function g, h.



Classically:

$$(x, y, 0) \xmapsto{\text{deletion is free}} (x, 0, f(x))$$

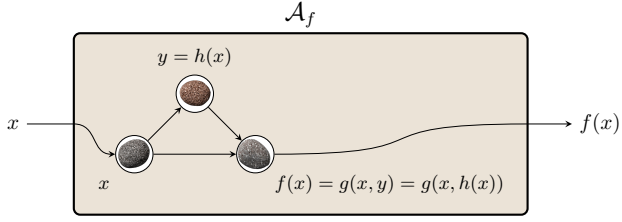
Quantum computation must be reversible:

$$|x, y, 0\rangle \longmapsto |x, y, f(x)\rangle \xmapsto{\text{uncompute}} |x, y \oplus y, f(x)\rangle = |x, 0, f(x)\rangle$$

Quantum computation reveals the missing piece: cleanup

Classical pebbling deletes for free; reversible computation cannot

Suppose we have $f(x) := g(x, h(x))$ for some function g, h.



Classically:

$$(x, y, 0) \xmapsto{\text{deletion is free}} (x, 0, f(x))$$

Quantum computation must be reversible:

$$|x, y, 0\rangle \longmapsto |x, y, f(x)\rangle \xmapsto{\text{uncompute}} |x, y \oplus y, f(x)\rangle = |x, 0, f(x)\rangle$$

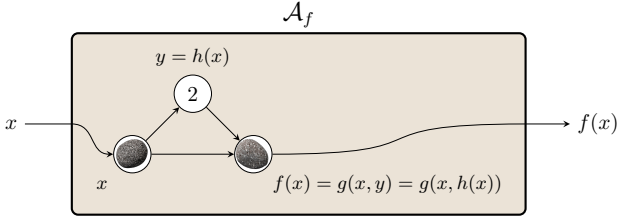
Observation 1

If a value is used to (un)compute something this round, you can't delete it in the same round.

Quantum computation reveals the missing piece: cleanup

Classical pebbling deletes for free; reversible computation cannot

Suppose we have $f(x) := g(x, h(x))$ for some function g, h.



Classically:

$$(x, y, 0) \xmapsto{\text{deletion is free}} (x, 0, f(x))$$

Quantum computation must be reversible:

$$|x, y, 0\rangle \longmapsto |x, y, f(x)\rangle \xmapsto{\text{uncompute}} |x, y \oplus y, f(x)\rangle = |x, 0, f(x)\rangle$$

Observation 1

If a value is used to (un)compute something this round, you can't delete it in the same round.

Observation 2

To uncompute y, we must still have its parents in memory (here: x).

We need a pebbling model that charges for cleanup

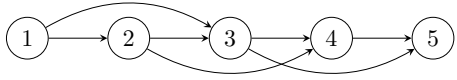
Observation 1

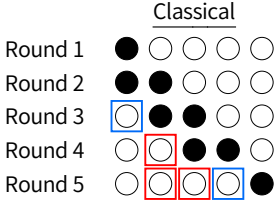
If a value is used to (un)compute something this round, you can't delete it in the same round.

Observation 2

To uncompute, we must still have its parents in memory.

Let's recall our previous example for a classical pebbling:





We need a pebbling model that charges for cleanup

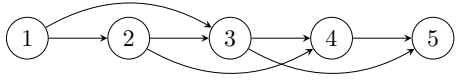
Observation 1

If a value is used to (un)compute something this round, you can't delete it in the same round.

Observation 2

To uncompute, we must still have its parents in memory.

Let's recall our previous example for a classical pebbling:



We need a pebbling model that charges for cleanup

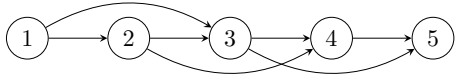
Observation 1

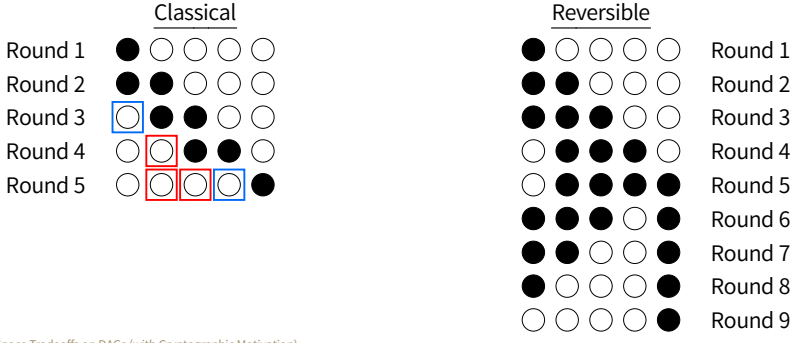
If a value is used to (un)compute something this round, you can't delete it in the same round.

Observation 2

To uncompute, we must still have its parents in memory.

Let's recall our previous example for a classical pebbling:





We need a pebbling model that charges for cleanup

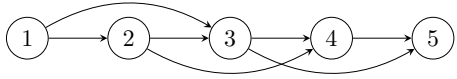
Observation 1

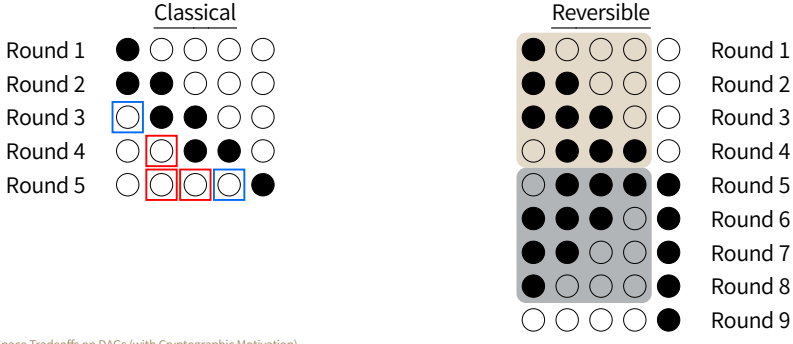
If a value is used to (un)compute something this round, you can't delete it in the same round.

Observation 2

To uncompute, we must still have its parents in memory.

Let's recall our previous example for a classical pebbling:





We add these constraints to the parallel reversible pebbling

Observation 1

If a value is used to (un)compute something this round, you can't delete it in the same round.

Observation 2

To uncompute, we must still have its parents in memory.

Parallel Reversible Pebbling Game Rules

Existing pebbling rules:

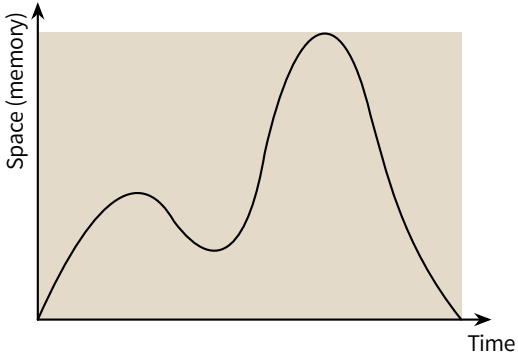
- Start with no pebbles on the graph and end with the target nodes
- All the parents need to be previously pebbled to place a new pebble

New reversible constraints:

- We must keep the pebbles if they are required to place/delete pebbles
- A pebble can be deleted only if all of its parents were previously pebbled

Pebbling formalizes space-time and cumulative cost

[Space-Time Cost]

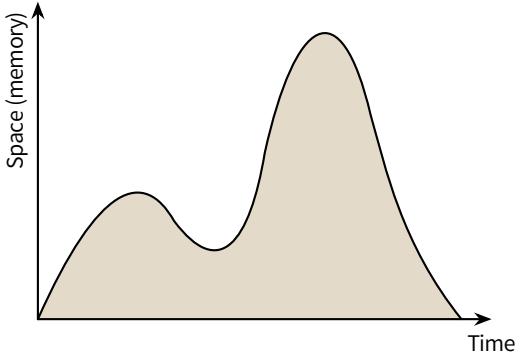


Given a pebbling $P = (P_1, \ldots, P_t)$,

$$\mathsf{ST}(P) := \left(\max_i |P_i|\right) \cdot t,$$

and for a DAG G, $\mathsf{ST}(G) := \min_{P \text{ legal for } G} \mathsf{ST}(P)$.

[Cumulative Cost]



Given a pebbling $P = (P_1, \ldots, P_t)$,

$$\mathsf{CC}(P) := \sum_{i=1}^{t} |P_i|,$$

and for a DAG G, $\mathsf{CC}(G) := \min_{P \text{ legal for } G} \mathsf{CC}(P)$.

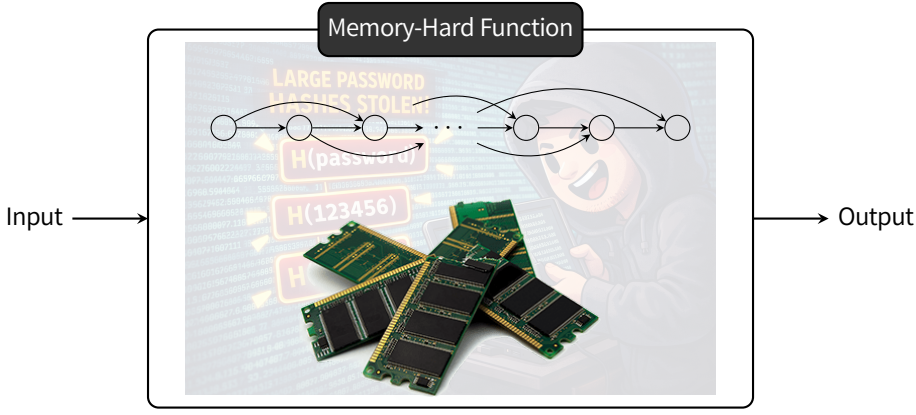
Memory-hardness fights parallel guessing

Cryptographic Motivation

- Offline attacks are cheap to parallelize (ASIC/GPUs)

Memory-hardness fights parallel guessing

Cryptographic Motivation

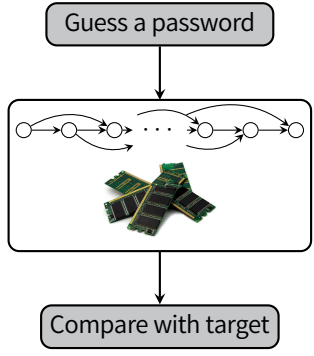


- Offline attacks are cheap to parallelize (ASIC/GPUs)
- Memory-Hard Functions force large memory (or large time) per evaluation
- Memory-Hard Functions can be defined by **structured DAGs**

Quantum search pays per reversible evaluation

Why space-time cost?

- A quantum search attack repeatedly evaluates the function
- Grover reduces the number of evaluations
- But each evaluation must still be implemented reversibly and coherently inside the quantum algorithm



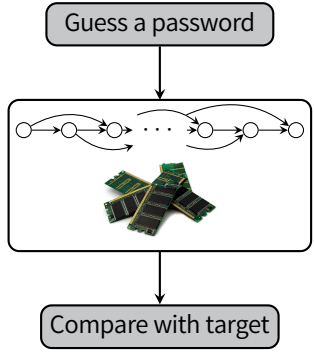
Quantum search pays per reversible evaluation

Why space-time cost?

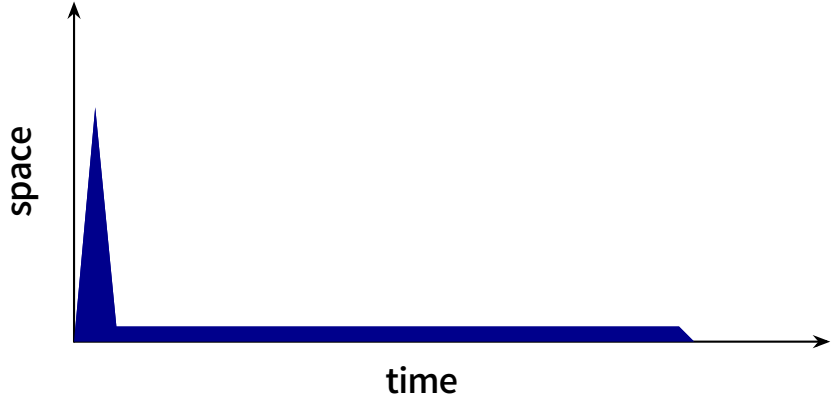
- A quantum search attack repeatedly evaluates the function
- Grover reduces the number of evaluations
- But each evaluation must still be implemented reversibly and coherently inside the quantum algorithm

$$\Downarrow$$

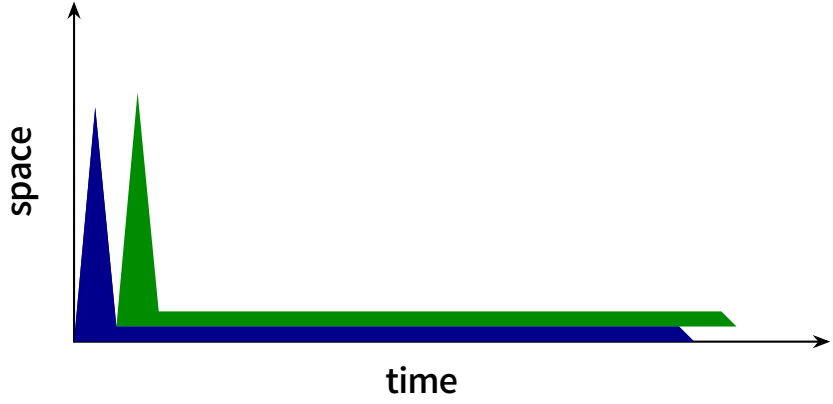
- So space–time cost measures the **full resource cost** of one reversible evaluation



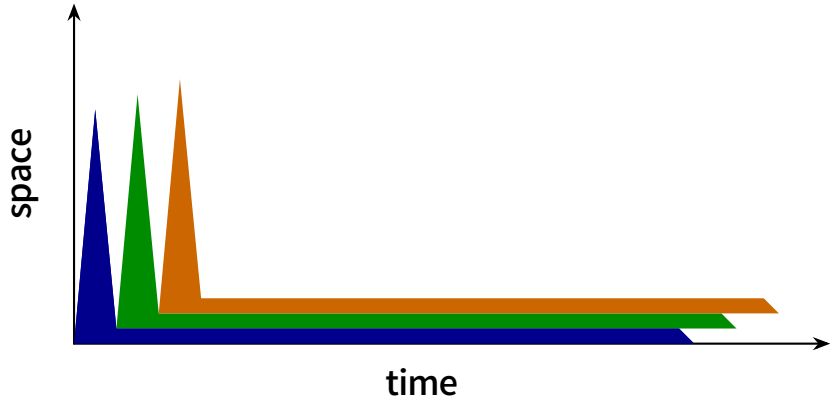
Cumulative cost captures amortization across many evaluations



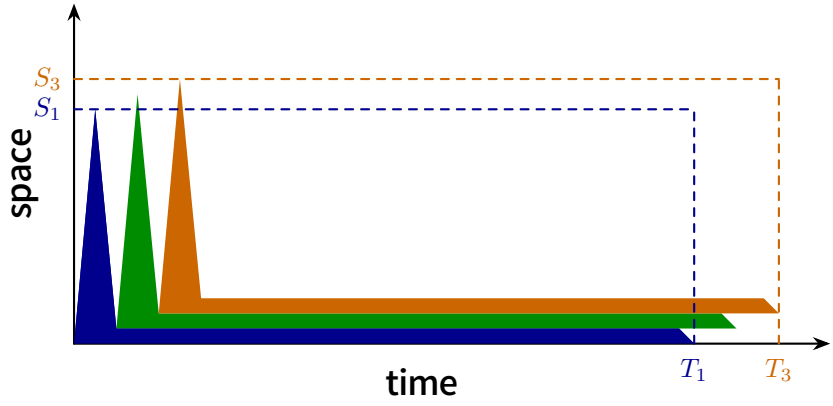
Cumulative cost captures amortization across many evaluations



Cumulative cost captures amortization across many evaluations



Cumulative cost captures amortization across many evaluations



- Space-time cost can stay roughly the same under pipelining ($S_1 \times T_1 \approx S_3 \times T_3$)
- Cumulative cost scales with the total memory used over time ($\mathsf{CC}(G^{\times 3}) = 3 \times \mathsf{CC}(G)$)

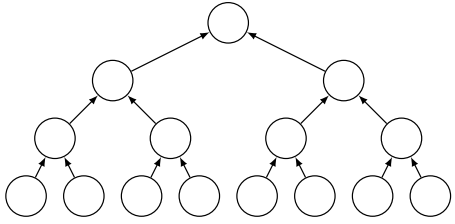
Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

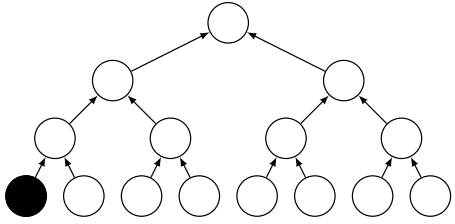
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

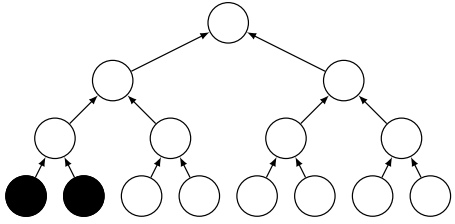
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

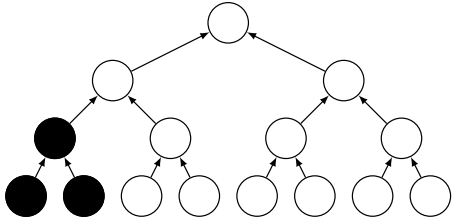
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

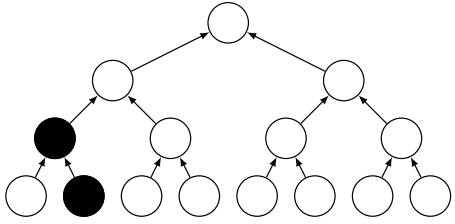
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

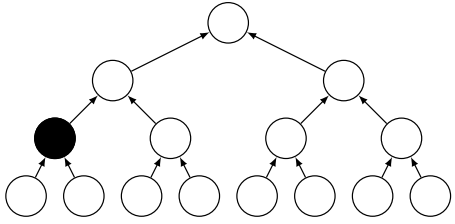
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

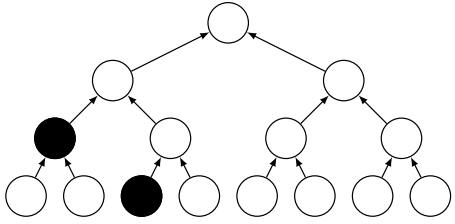
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

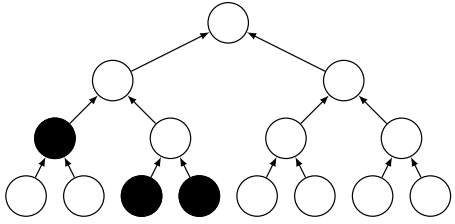
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

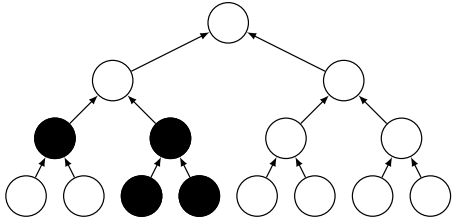
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

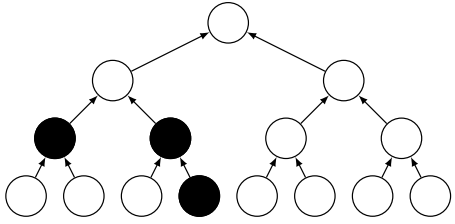
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

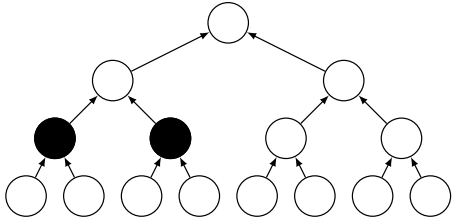
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

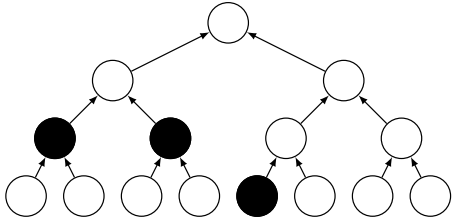
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

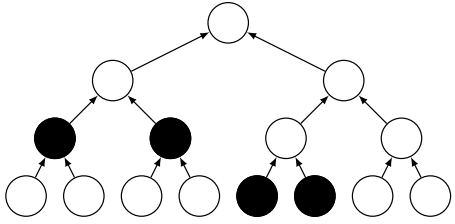
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

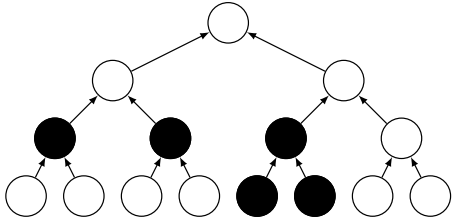
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

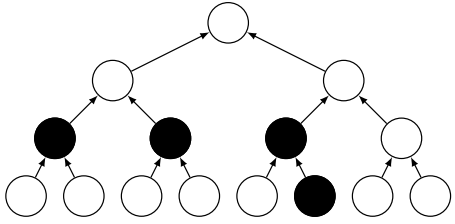
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

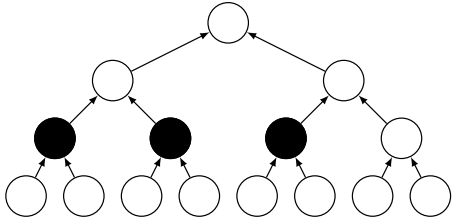
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

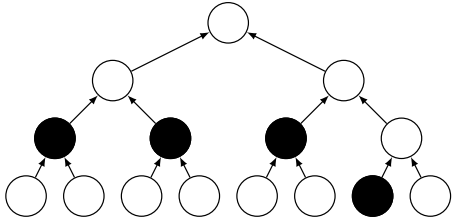
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

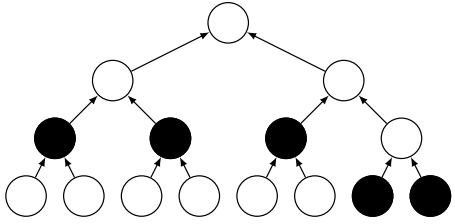
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

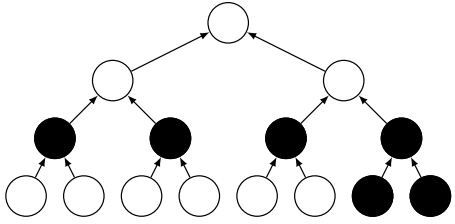
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

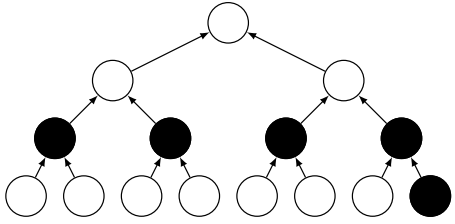
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

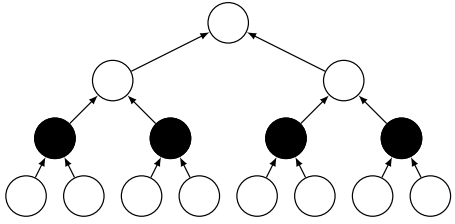
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

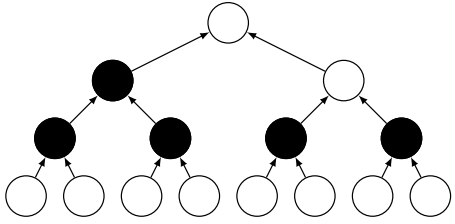
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

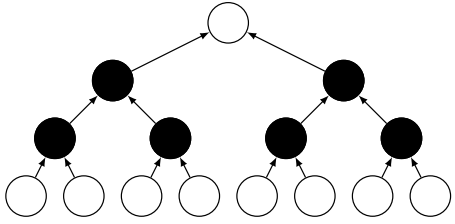
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

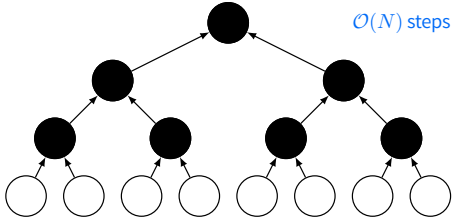
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

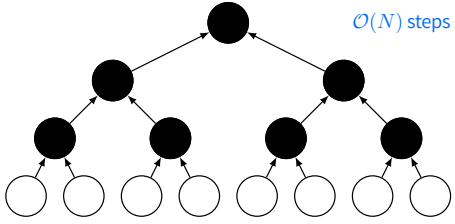
Sequential



Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Sequential



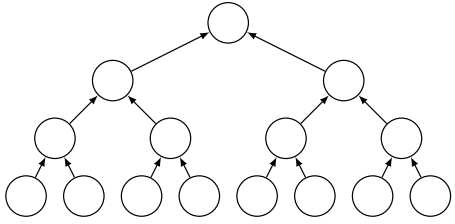
Sequential model:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(N)$

Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Parallel



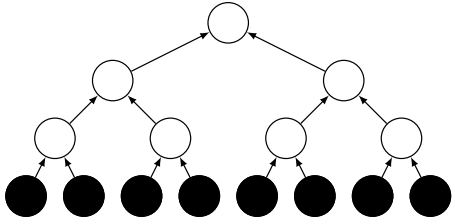
Sequential model:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(N)$

Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Parallel



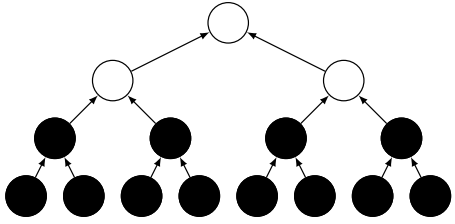
Sequential model:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(N)$

Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Parallel



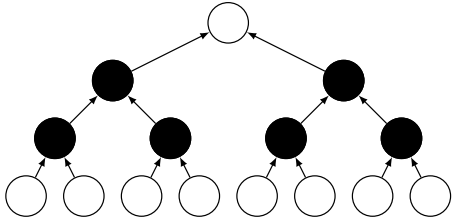
Sequential model:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(N)$

Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Parallel



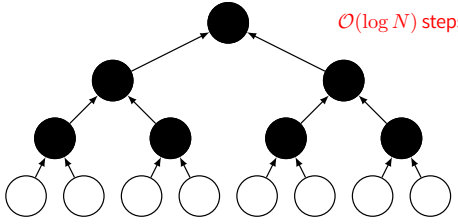
Sequential model:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(N)$

Sequential reversible pebbling misses parallel speedups

- Prior work [Ben89, LV96, Krá01, MSR+19] models reversible computation **sequentially**
- It correctly models cleanup, but it can substantially overestimate time when the DAG has a parallel structure

Parallel



Sequential model:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(N)$

Parallel evaluation:

$\Rightarrow$ (ST-Cost) $\approx$ (space) $\times \mathcal{O}(\log N)$

Three takeaways

Parallel Reversible Pebbling

LANGUAGE — **a clean combinatorial way** to reason about time-space tradeoffs **on DAGs (directed acyclic graphs)**

Upper and Lower Bounds

CONCRETE BOUNDS — **upper/lower bounds** for reversible pebbling on simple and structured DAGs

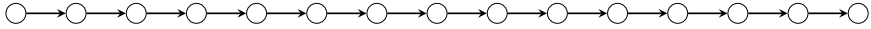
Generic Transformations

REUSABLE TOOL — a **generic transformation** from classical to reversible pebbling strategies

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

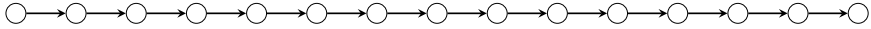


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

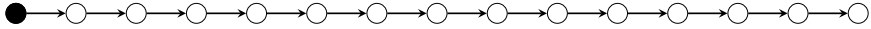


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

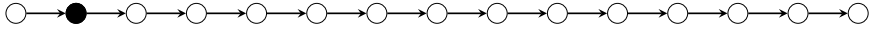


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

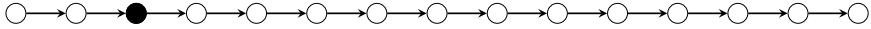


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

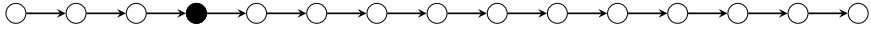


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

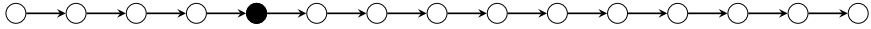


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

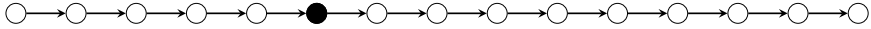


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

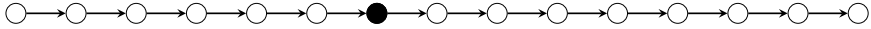


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling



We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

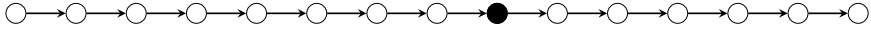
Parallel Classical Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

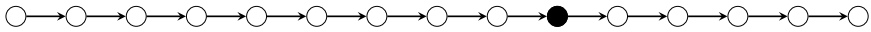


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

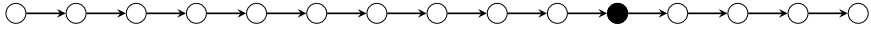


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

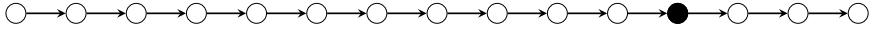


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

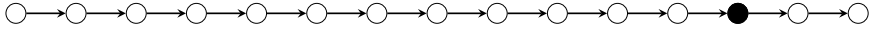


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

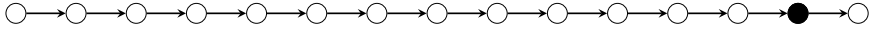


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling

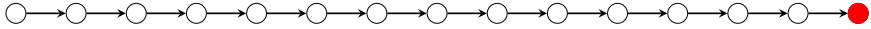


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Classical Pebbling



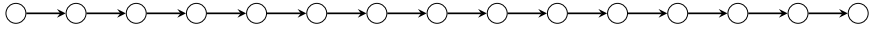
$(\text{ST-Cost}) = N$

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

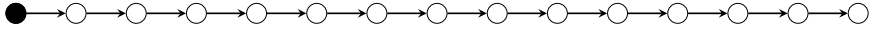


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

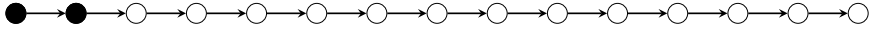


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

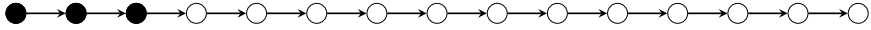


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

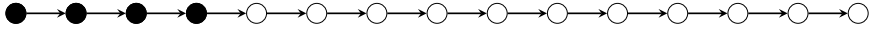


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

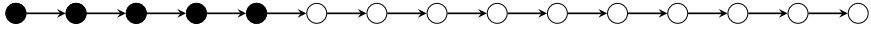


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

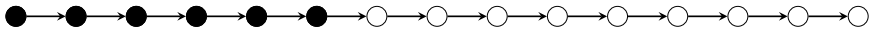


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

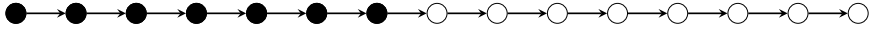


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling



We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Naïve Reversible Pebbling

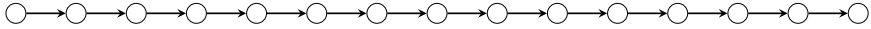
$(\text{ST-Cost}) = N^2$

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

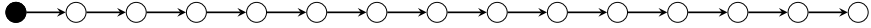


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

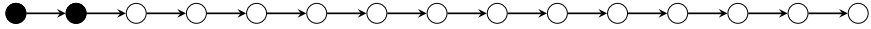


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

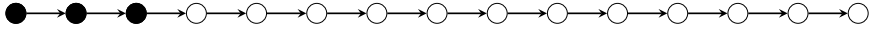


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

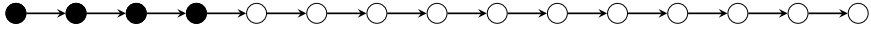


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

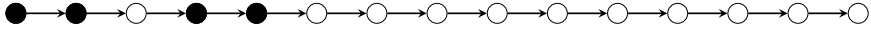


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

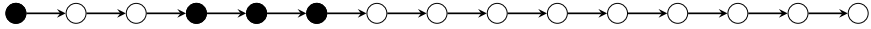


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

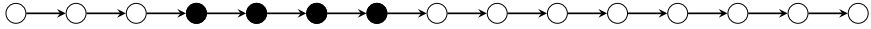


We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]



We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

We begin with the simplest DAG: a line graph

Space-time upper bound

Note. A reversible pebbling strategy gives an upper bound

Parallel Reversible Pebbling [BHL22]

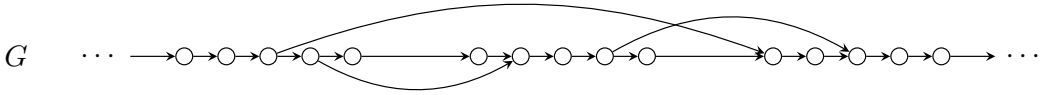
Our Result [BHL22]

$\|$: parallel, $\Longleftrightarrow$: reversible

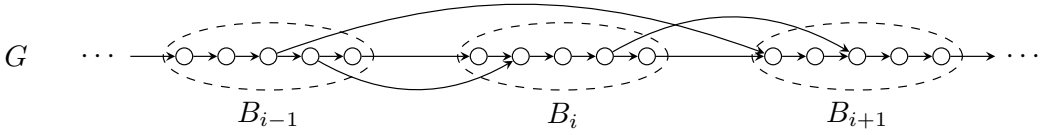
For a line graph L_N with N nodes, we have $\mathsf{ST}^{\|,\Longleftrightarrow}(L_N) = \mathcal{O}\left(N^{1+\frac{2}{\sqrt{\log N}}}\right)$.

- We modified Bennett and Li & Vitányi's strategy [Ben89, LV96] which used recursion
- This is efficient: for some N, this is always **tighter than** $\mathcal{O}\left(N^{1+\varepsilon}\right)$ for any constant $\varepsilon > 0$

A structured DAG can inherit a line-graph pebbling strategy

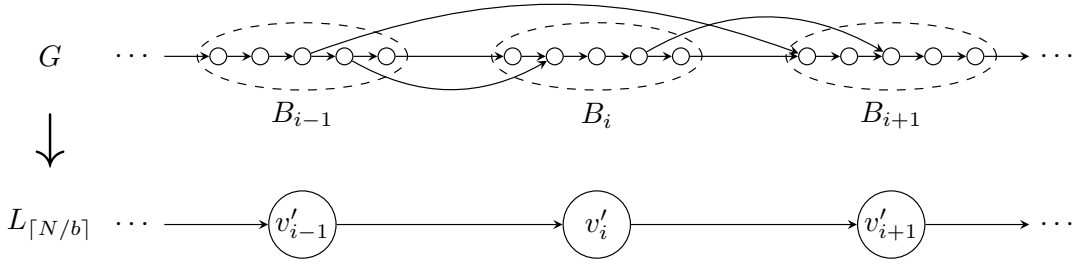


A structured DAG can inherit a line-graph pebbling strategy



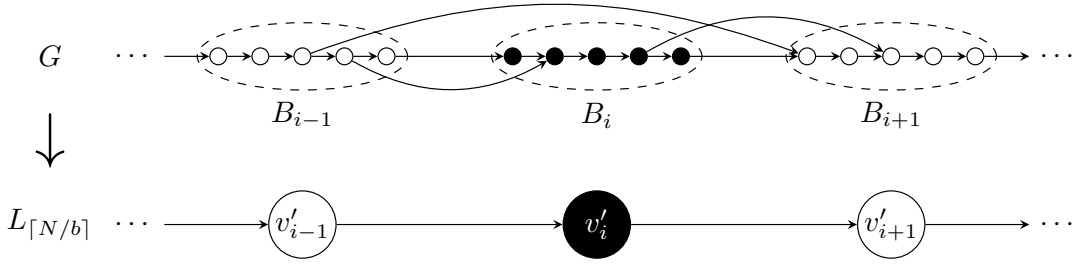
- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$

A structured DAG can inherit a line-graph pebbling strategy



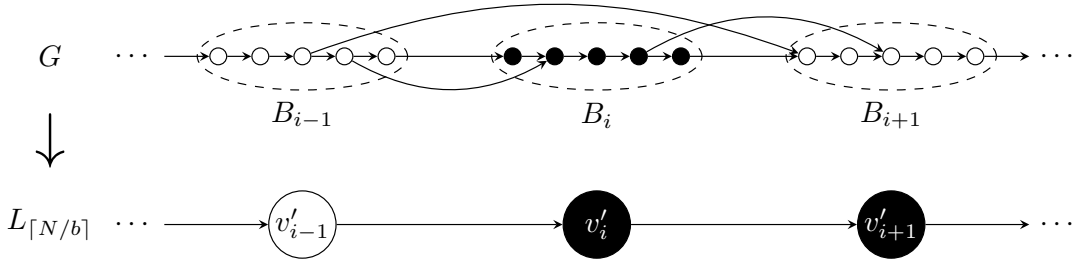
- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$

A structured DAG can inherit a line-graph pebbling strategy



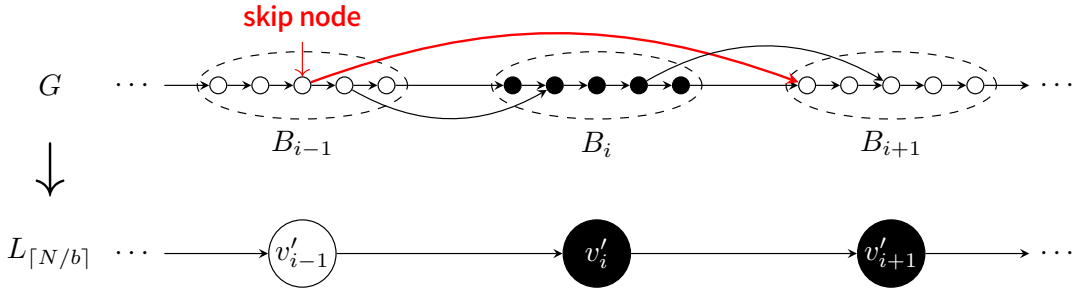
- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

A structured DAG can inherit a line-graph pebbling strategy



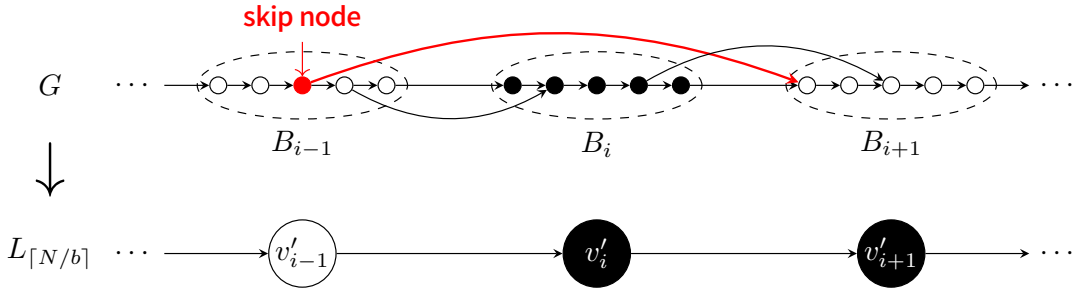
- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

A structured DAG can inherit a line-graph pebbling strategy



- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

A structured DAG can inherit a line-graph pebbling strategy

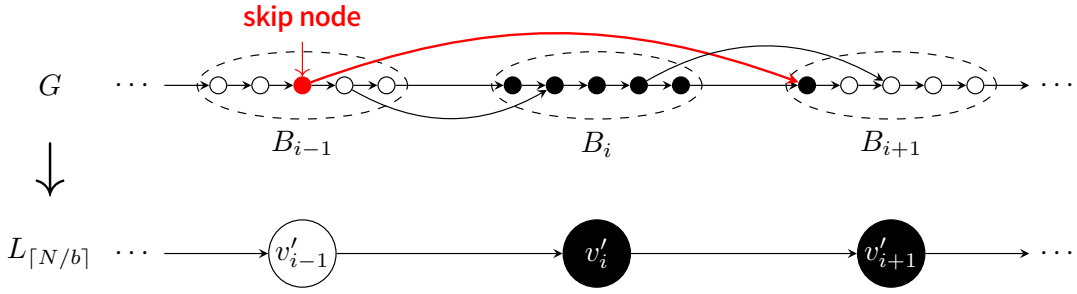


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

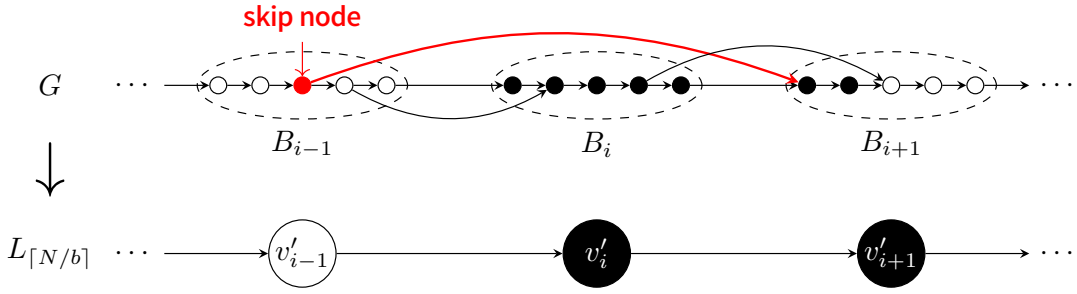


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

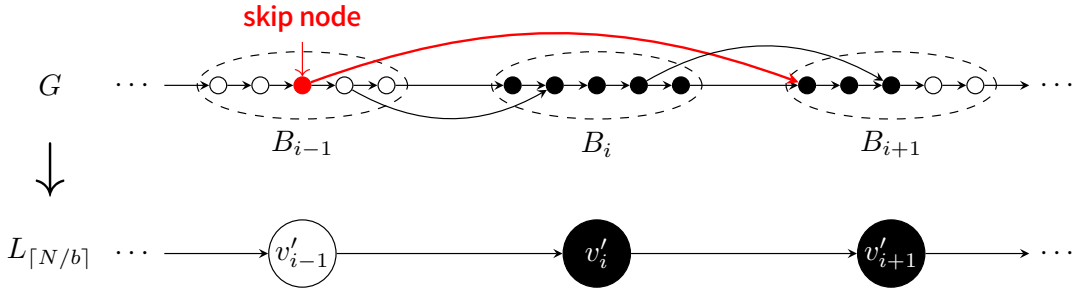


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

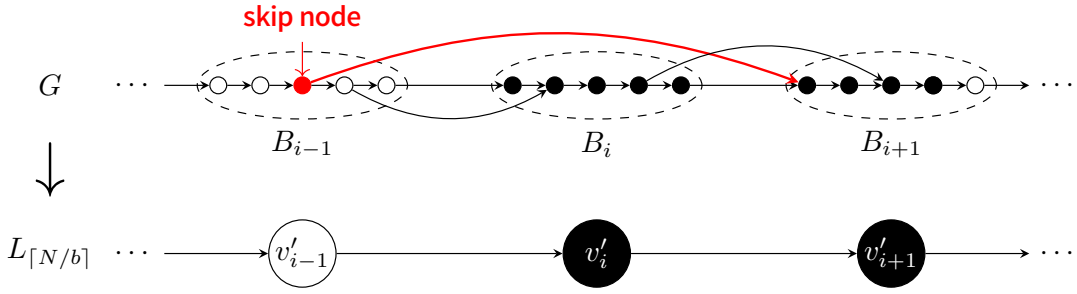


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

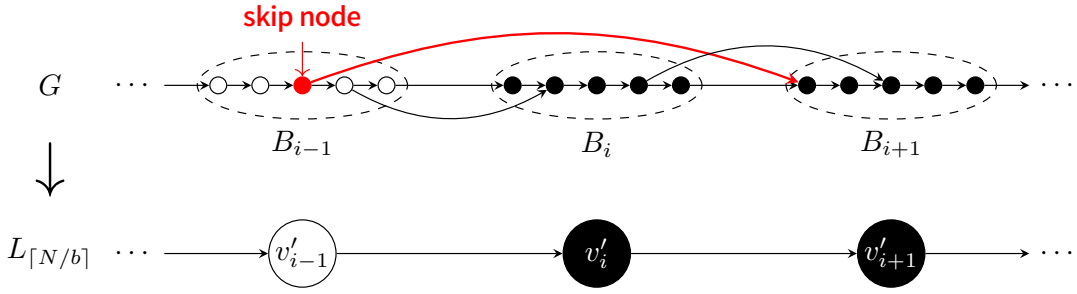


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

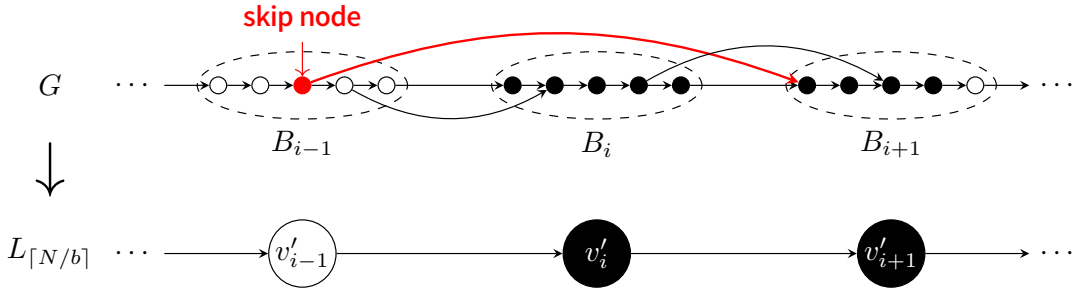


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

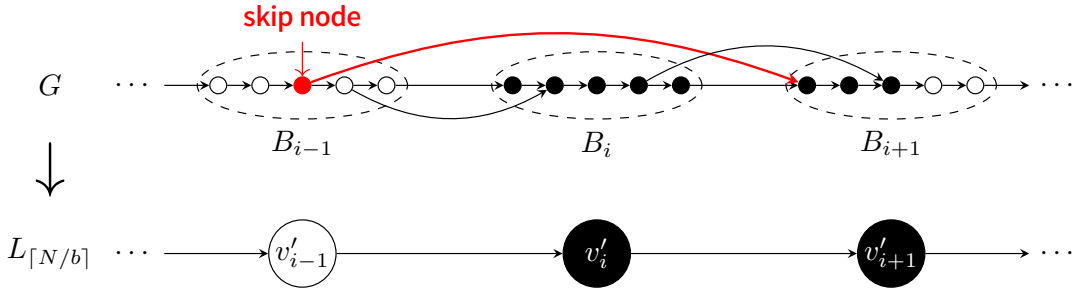


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

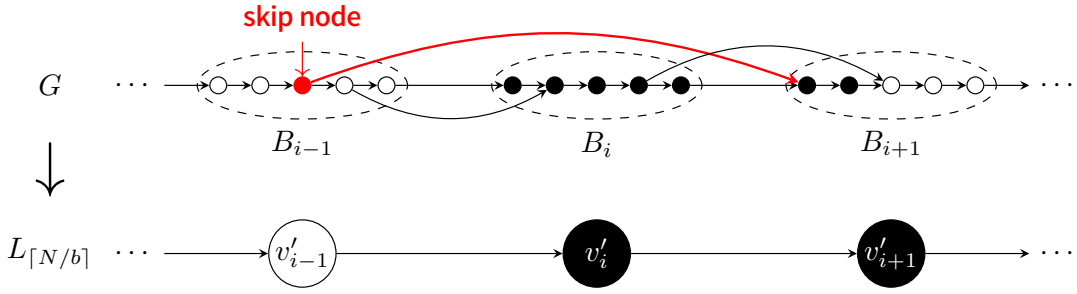


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

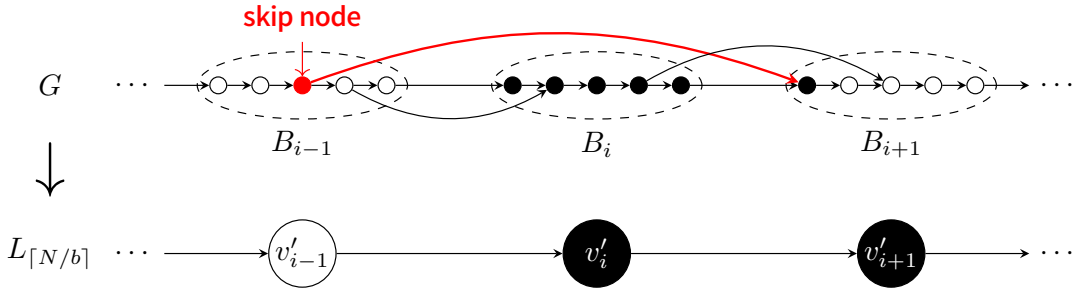


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

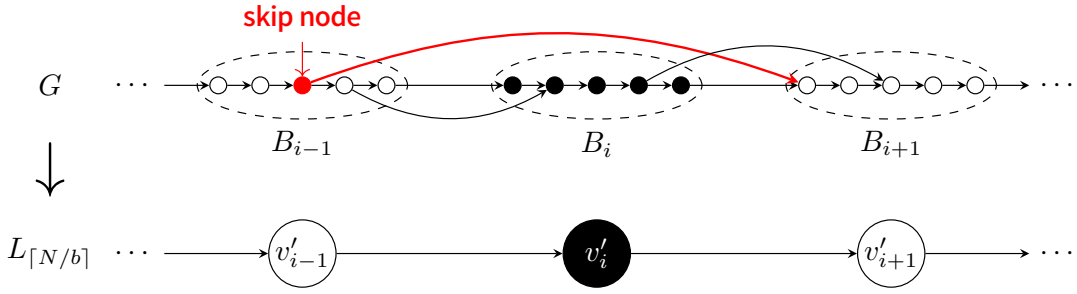


- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

A structured DAG can inherit a line-graph pebbling strategy

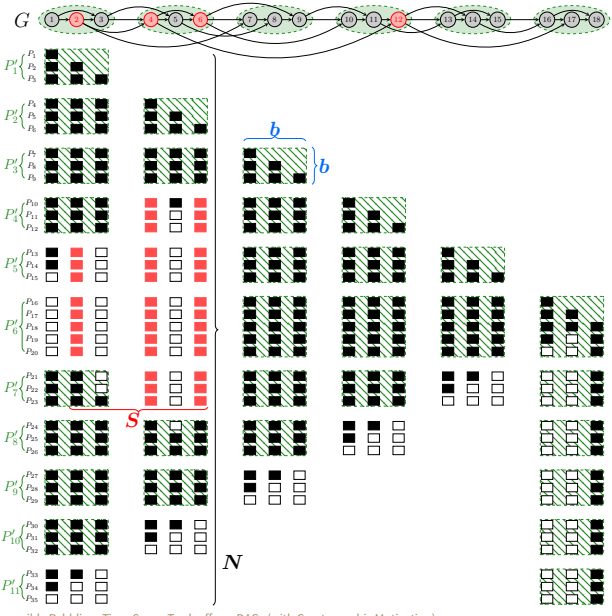


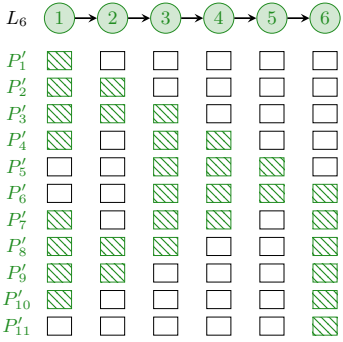
- Given a graph G, split into blocks of size b and create a line graph $L_{\lceil N/b \rceil}$ of size $\lceil N/b \rceil$
- Transform the reversible pebbling of $L_{\lceil N/b \rceil}$ to the original graph G

Key Intuition

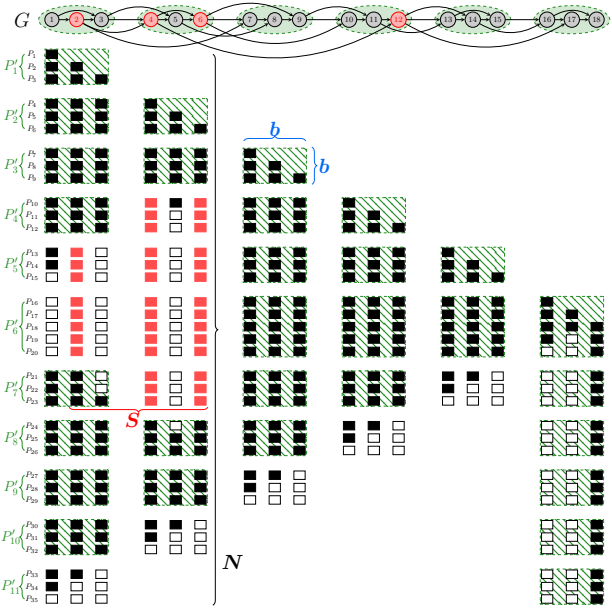
If we keep pebbles on the skip nodes, then we can effectively transform a reversible pebbling of $L_{\lceil N/b \rceil}$ to a reversible pebbling of G

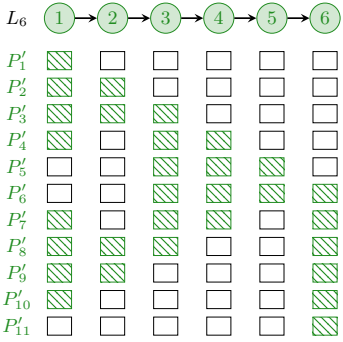
The transfer yields an ST upper bound for the original DAG





The transfer yields an ST upper bound for the original DAG





Our Result [BHL22]

$\mathsf{ST}^{\|,\leftrightarrows}(G) =$
$\mathcal{O}\Big(SN + b^2 \cdot \mathsf{ST}^{\|,\leftrightarrows}(L_{\lceil N/b \rceil})\Big)$,
where $S =$ (# skip nodes) and
$b > 0$: block size

DRSample is one structured DAG where this technique applies

Our Result [BHL22]

$\mathsf{ST}^{\|,\leftrightarrow}(G) = \mathcal{O}\Big(SN + b^2 \cdot \mathsf{ST}^{\|,\leftrightarrow}(L_{\lceil N/b \rceil})\Big)$, where $S = (\#\text{ skip nodes})$ and $b > 0$: block size

DRSample [ABH17]: a Memory-Hard Function candidate with strong classical memory-hardness

- A line graph + random additional edge for each node (not uniformly random)

DRSample is one structured DAG where this technique applies

Our Result [BHL22]

$\mathsf{ST}^{\|,\leftrightarrow}(G) = \mathcal{O}\Big(SN + b^2 \cdot \mathsf{ST}^{\|,\leftrightarrow}(L_{\lceil N/b \rceil})\Big)$, where $S = (\#\text{ skip nodes})$ and $b > 0$: block size

DRSample [ABH17]: a Memory-Hard Function candidate with strong classical memory-hardness

- A line graph + random additional edge for each node (not uniformly random)
- We show that if $b = \mathcal{O}\left(\frac{N}{\log N}\right)$, then $S = \mathcal{O}\left(\frac{N \log\log N}{\log N}\right)$

DRSample is one structured DAG where this technique applies

Our Result [BHL22]

$\mathsf{ST}^{\|,\leftrightarrow}(G) = \mathcal{O}\Big(SN + b^2 \cdot \mathsf{ST}^{\|,\leftrightarrow}(L_{\lceil N/b \rceil})\Big)$, where $S = (\#\text{ skip nodes})$ and $b > 0$: block size

DRSample [ABH17]: a Memory-Hard Function candidate with strong classical memory-hardness

- A line graph + random additional edge for each node (not uniformly random)
- We show that if $b = \mathcal{O}\left(\frac{N}{\log N}\right)$, then $S = \mathcal{O}\left(\frac{N \log\log N}{\log N}\right)$

$$\Rightarrow \mathsf{ST}^{\|,\leftrightarrow}(\text{DRSample}) = \mathcal{O}\left(\frac{N^2 \log\log N}{\log N}\right)$$

$\therefore$ A line-graph upper bound can become a concrete bound for a cryptographic DAG

DRSample is one structured DAG where this technique applies

Our Result [BHL22]

$\mathsf{ST}^{\|,\leftrightarrow}(G) = \mathcal{O}\left(SN + b^2 \cdot \mathsf{ST}^{\|,\leftrightarrow}(L_{\lceil N/b \rceil})\right)$, where $S = (\#\text{ skip nodes})$ and $b > 0$: block size

DRSample [ABH17]: a Memory-Hard Function candidate with strong classical memory-hardness

- A line graph + random additional edge for each node (not uniformly random)
- We show that if $b = \mathcal{O}\left(\frac{N}{\log N}\right)$, then $S = \mathcal{O}\left(\frac{N \log\log N}{\log N}\right)$

$\Rightarrow \mathsf{ST}^{\|,\leftrightarrow}(\text{DRSample}) = \mathcal{O}\left(\frac{N^2 \log\log N}{\log N}\right)$ **Note.** $\mathsf{ST}^{\|,\leftrightarrow}(\text{DRSample}) = \Omega\left(\frac{N^2}{\log N}\right)$

$\therefore$ A line-graph upper bound can become a concrete bound for a cryptographic DAG

DRSample is one structured DAG where this technique applies

Our Result [BHL22]

$\mathsf{ST}^{\|,\leftrightarrow}(G) = \mathcal{O}\left(SN + b^2 \cdot \mathsf{ST}^{\|,\leftrightarrow}(L_{\lceil N/b \rceil})\right)$, where $S = (\#\text{ skip nodes})$ and $b > 0$: block size

DRSample [ABH17]: a Memory-Hard Function candidate with strong classical memory-hardness

- A line graph + random additional edge for each node (not uniformly random)
- We show that if $b = \mathcal{O}\left(\frac{N}{\log N}\right)$, then $S = \mathcal{O}\left(\frac{N \log\log N}{\log N}\right)$

$\Rightarrow \mathsf{ST}^{\|,\leftrightarrow}(\text{DRSample}) = \mathcal{O}\left(\frac{N^2 \log\log N}{\log N}\right)$ **Note.** $\mathsf{ST}^{\|,\leftrightarrow}(\text{DRSample}) = \Omega\left(\frac{N^2}{\log N}\right)$

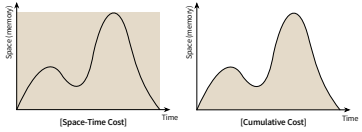
$\therefore$ A line-graph upper bound can become a concrete bound for a cryptographic DAG

Question

How close is the line-graph upper bound to optimal?

A line graph already reveals the cost of reversibility

Observation. For any DAG G, $\mathsf{CC}^{\parallel,\leftrightarrow}(G) \leq \mathsf{ST}^{\parallel,\leftrightarrow}(G)$

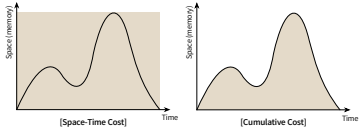


Our Result [BHL25]

For a line graph L_N with N nodes, $\mathsf{CC}^{\parallel,\leftrightarrow}(L_N) = \Omega\left(N^{1+\frac{\sqrt{2}-o(1)}{\sqrt{\log N}}}\right)$.

A line graph already reveals the cost of reversibility

Observation. For any DAG G, $\mathsf{CC}^{\parallel,\leftrightarrow}(G) \leq \mathsf{ST}^{\parallel,\leftrightarrow}(G)$



Our Result [BHL25]

For a line graph L_N with N nodes, $\mathsf{CC}^{\parallel,\leftrightarrow}(L_N) = \Omega\left(N^{1+\frac{\sqrt{2}-o(1)}{\sqrt{\log N}}}\right)$.

- **Separation:**
  - For a line graph, classical pebbling costs only $\Theta(N)$, while reversible pebbling is already superlinear
- **Near-optimality:**
  - Our upper bound $\mathcal{O}\left(N^{1+\frac{2}{\sqrt{\log N}}}\right)$ is within a subpolynomial factor of the lower bound

Three takeaways

Parallel Reversible Pebbling

LANGUAGE — **a clean combinatorial way** to reason about time-space tradeoffs **on DAGs (directed acyclic graphs)**

Upper and Lower Bounds

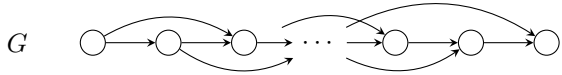
CONCRETE BOUNDS — **upper/lower bounds** for reversible pebbling on simple and structured DAGs

Generic Transformations

REUSABLE TOOL — a **generic transformation** from classical to reversible pebbling strategies

A classical pebbling can be transformed into a reversible one

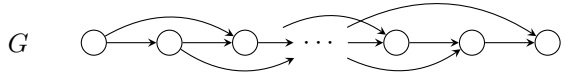
Pebbling Composition



Classical pebbling $P = \{P_1, \ldots, P_t\}$ of G

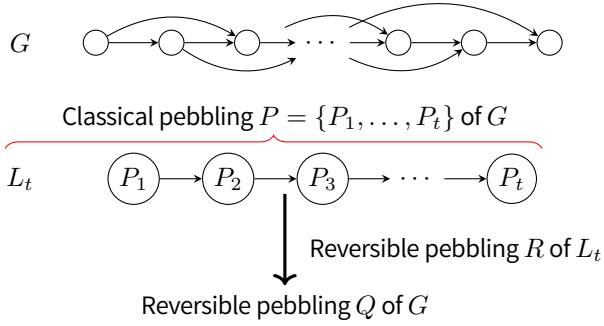
A classical pebbling can be transformed into a reversible one

Pebbling Composition



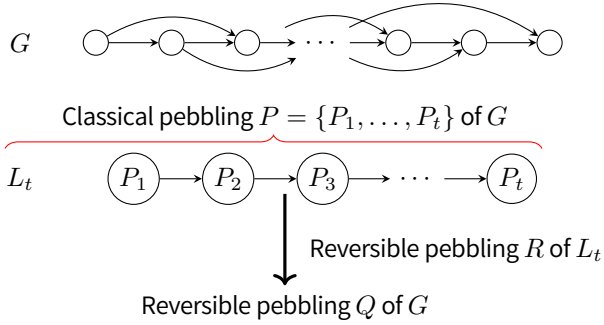
A classical pebbling can be transformed into a reversible one

Pebbling Composition



A classical pebbling can be transformed into a reversible one

Pebbling Composition



Our Result [BHL25]

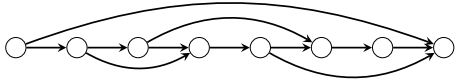
For a DAG G with N nodes, $\mathsf{ST}^{\parallel,\leftrightarrow}(G) = \mathcal{O}\left(N^{\frac{2\sqrt{2}}{\sqrt{\log N}}}\right) \cdot \mathsf{ST}^{\parallel}(G)$.

- The reversible overhead is only **subpolynomial**: $\mathcal{O}\left(N^{o(1)}\right)$

A reversible pebbling determines which classical states to keep

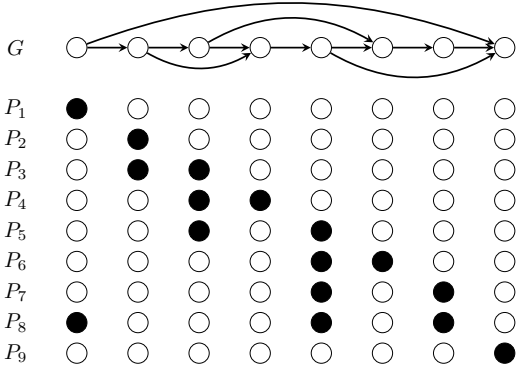
Pebbling composition example

G



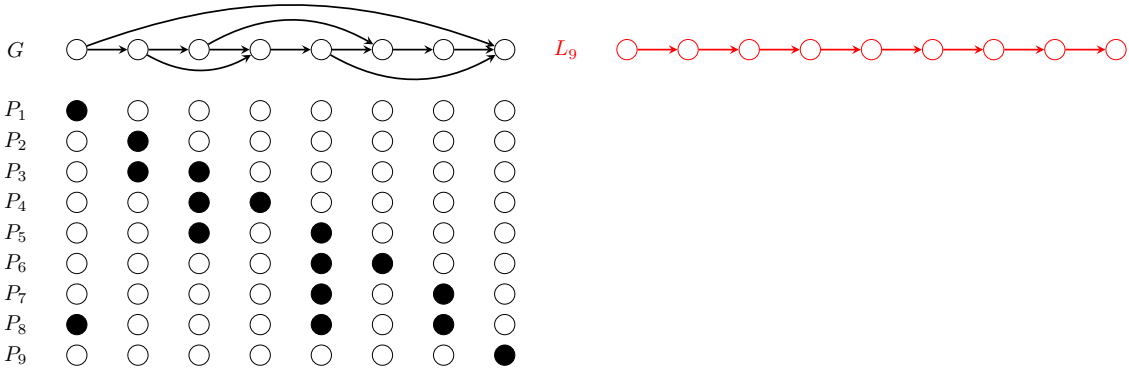
A reversible pebbling determines which classical states to keep

Pebbling composition example



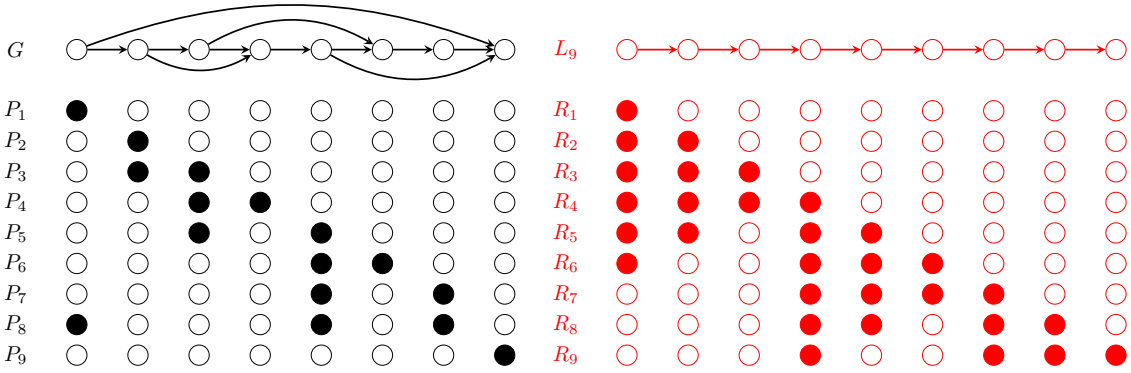
A reversible pebbling determines which classical states to keep

Pebbling composition example



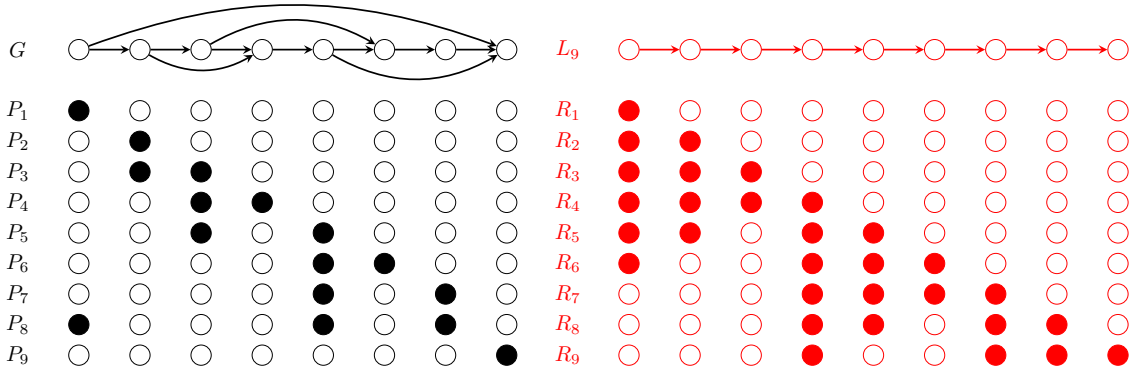
A reversible pebbling determines which classical states to keep

Pebbling composition example



A reversible pebbling determines which classical states to keep

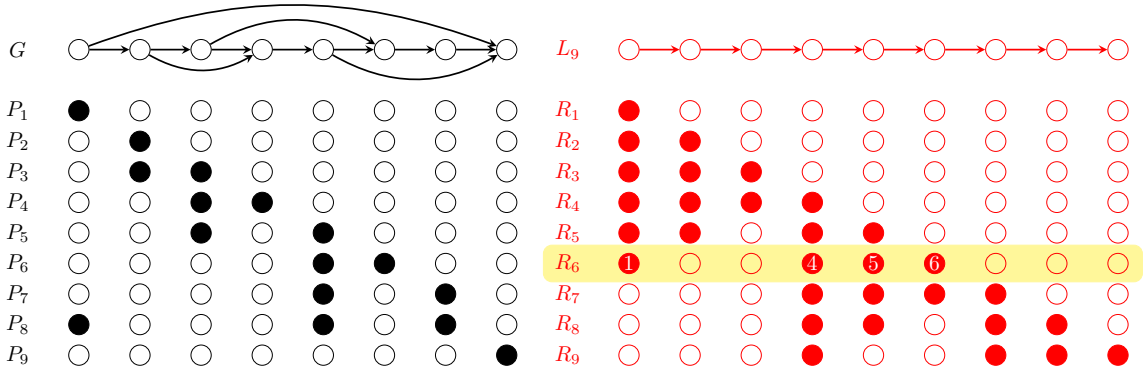
Pebbling composition example



For $i = 1, ..., 9$, set $Q_i := \bigcup_{j \in R_i} P_j$

A reversible pebbling determines which classical states to keep

Pebbling composition example

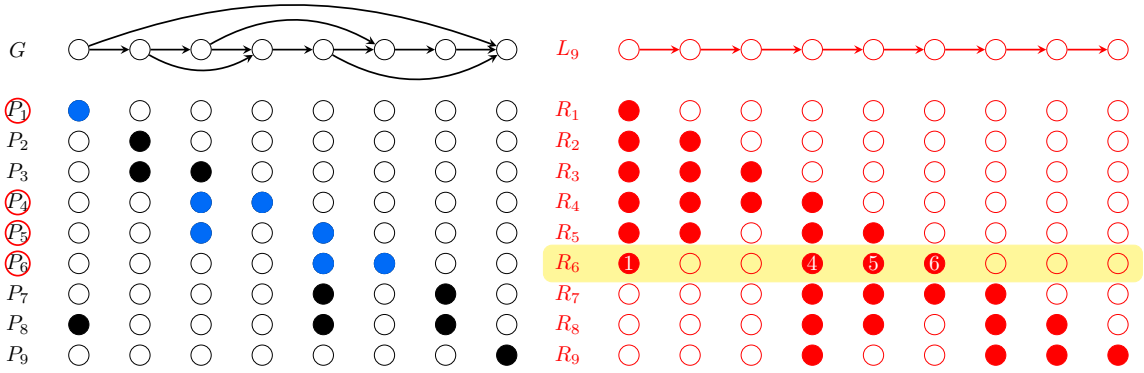


For $i = 1, ..., 9$, set $Q_i := \bigcup_{j \in R_i} P_j$

$$Q_6 = \bigcup_{j \in R_6} P_j = P_1 \cup P_4 \cup P_5 \cup P_6$$

A reversible pebbling determines which classical states to keep

Pebbling composition example

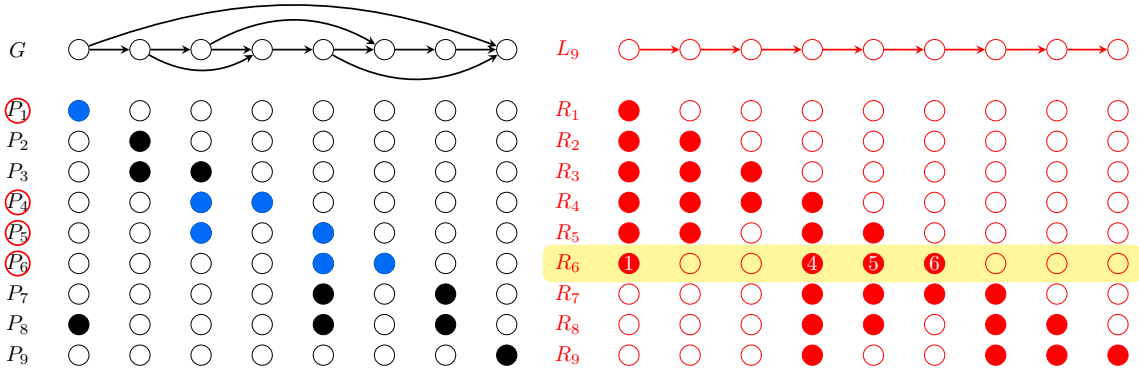


For $i = 1, ..., 9$, set $Q_i := \bigcup_{j \in R_i} P_j$

$$Q_6 = \bigcup_{j \in R_6} P_j = P_1 \cup P_4 \cup P_5 \cup P_6$$

A reversible pebbling determines which classical states to keep

Pebbling composition example

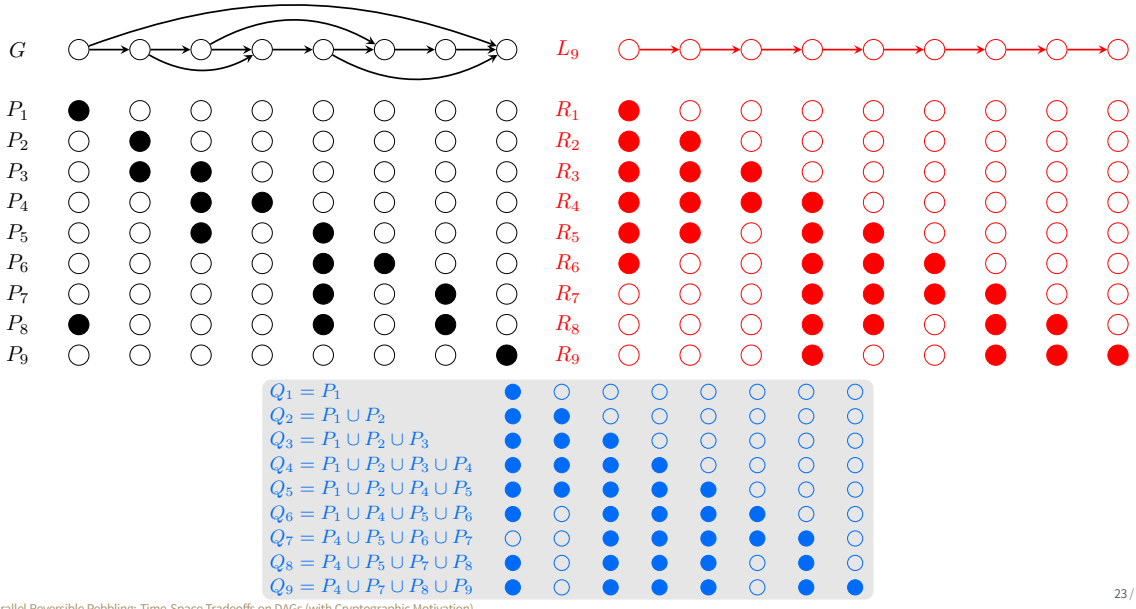


For $i = 1, ..., 9$, set $Q_i := \bigcup_{j \in R_i} P_j$

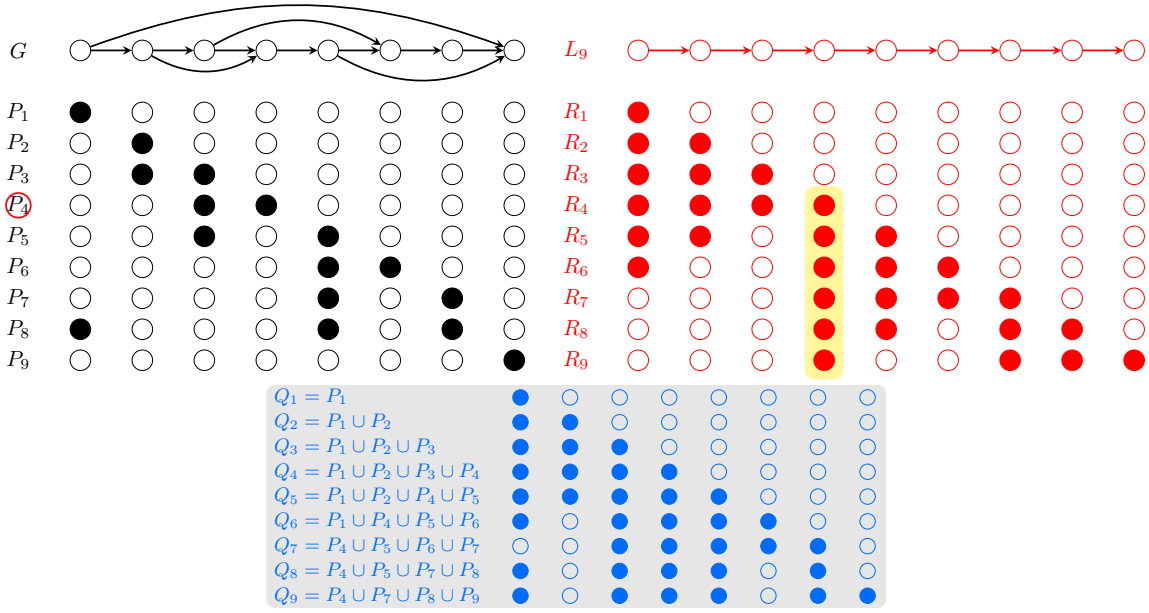
$$Q_6 = \bigcup_{j \in R_6} P_j = P_1 \cup P_4 \cup P_5 \cup P_6$$

A reversible pebbling determines which classical states to keep

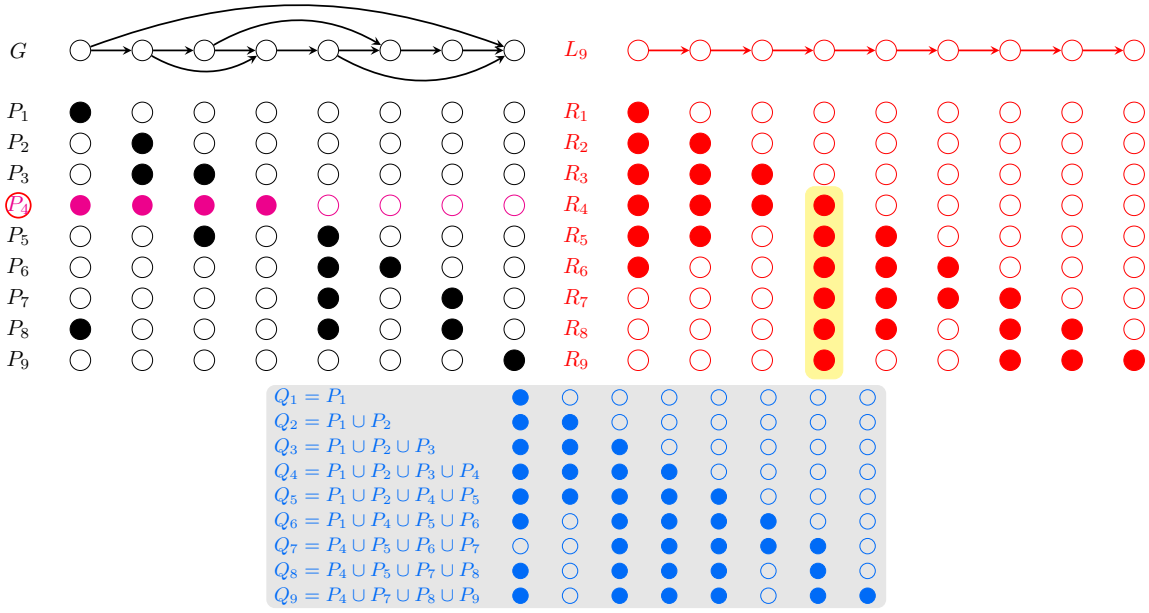
Pebbling composition example



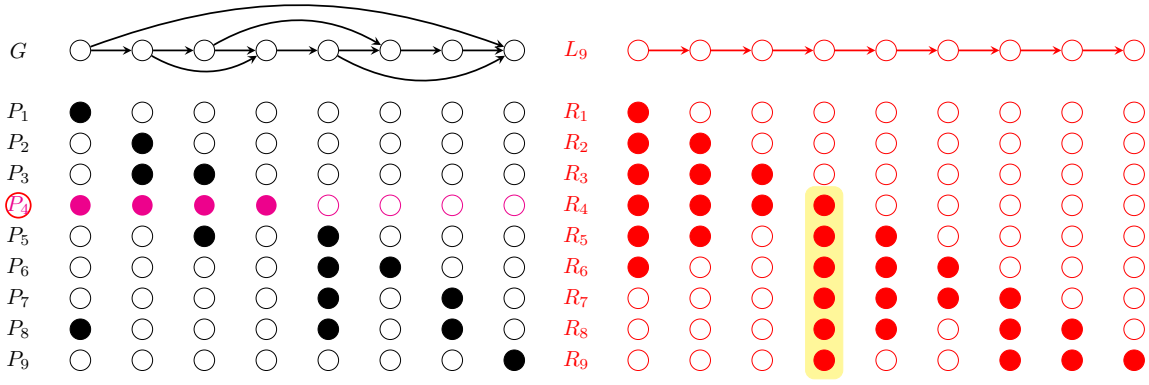
Naïve composition can blow up cumulative cost



Naïve composition can blow up cumulative cost



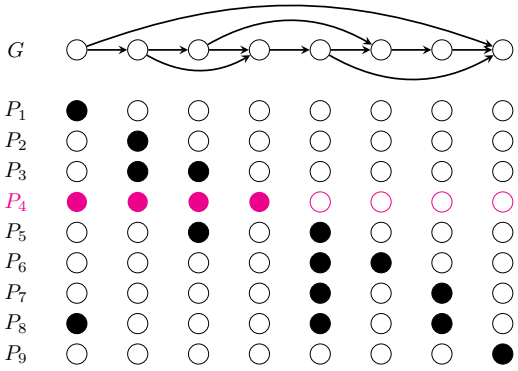
Naïve composition can blow up cumulative cost



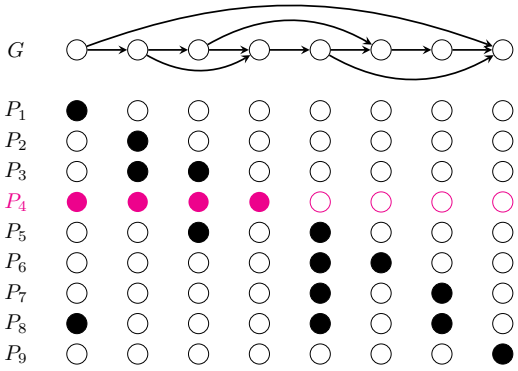
$Q_1 = P_1$
$Q_2 = P_1 \cup P_2$
$Q_3 = P_1 \cup P_2 \cup P_3$
$Q_4 = P_1 \cup P_2 \cup P_3 \cup P_4$
$Q_5 = P_1 \cup P_2 \cup P_4 \cup P_5$
$Q_6 = P_1 \cup P_4 \cup P_5 \cup P_6$
$Q_7 = P_4 \cup P_5 \cup P_6 \cup P_7$
$Q_8 = P_4 \cup P_5 \cup P_7 \cup P_8$
$Q_9 = P_4 \cup P_7 \cup P_8 \cup P_9$

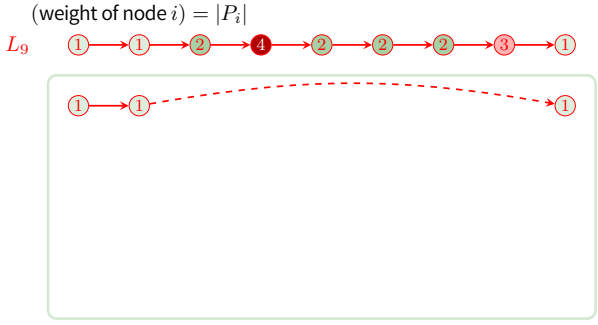
$\therefore \mathsf{CC}(P) \ll \mathsf{CC}(Q)$!

A weighted line graph fixes this problem



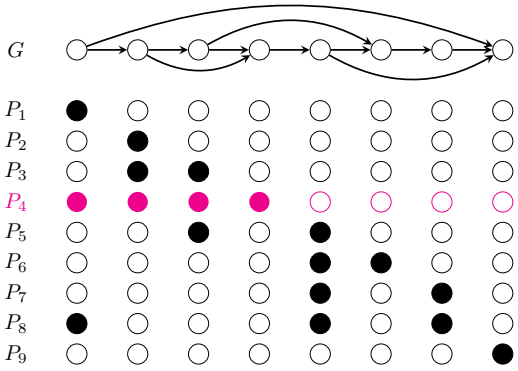
A weighted line graph fixes this problem

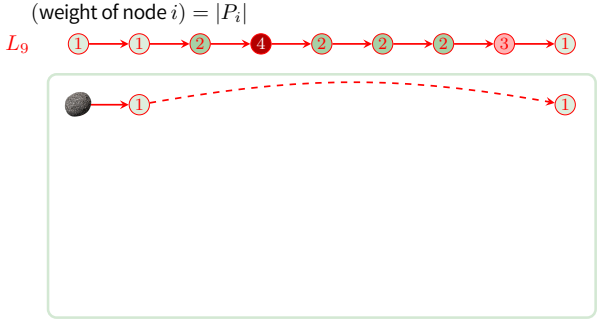




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

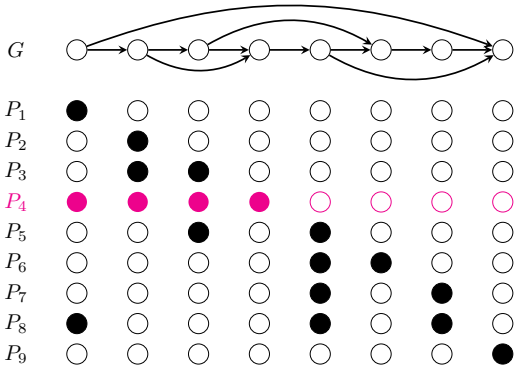
A weighted line graph fixes this problem

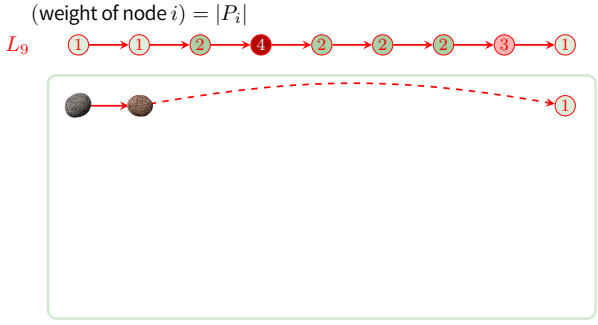




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

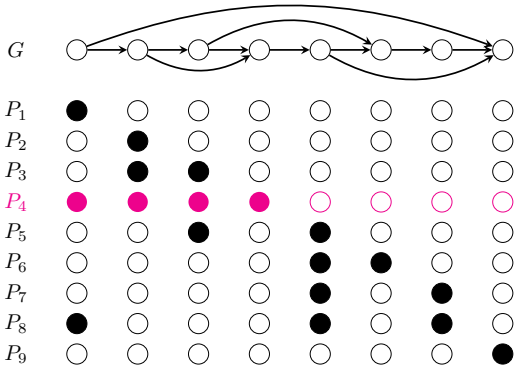
A weighted line graph fixes this problem

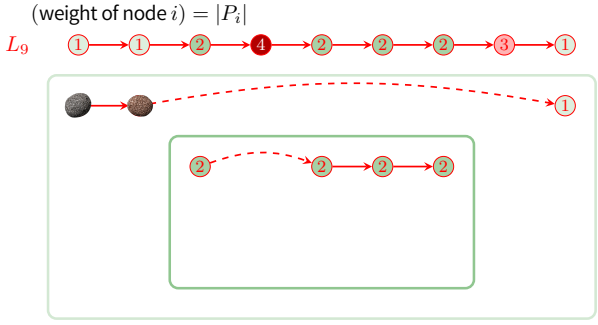




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

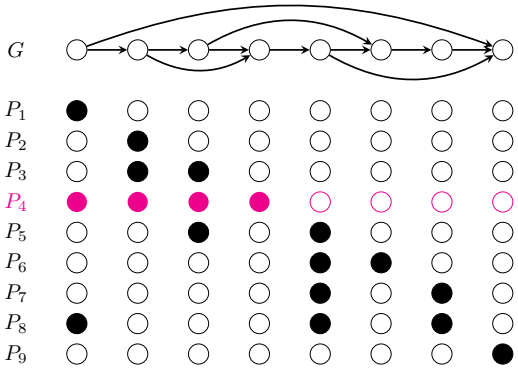
A weighted line graph fixes this problem

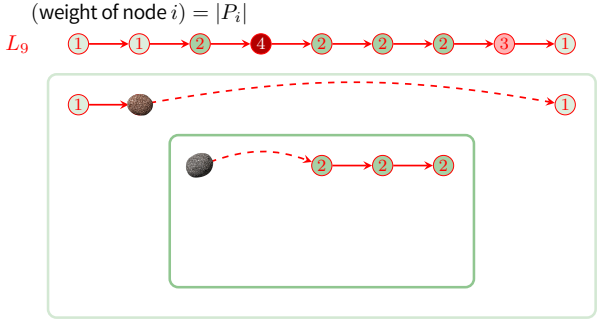




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

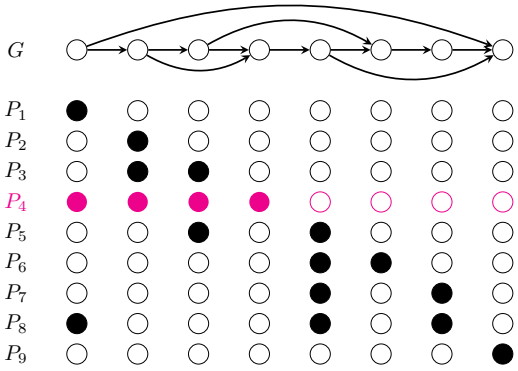
A weighted line graph fixes this problem

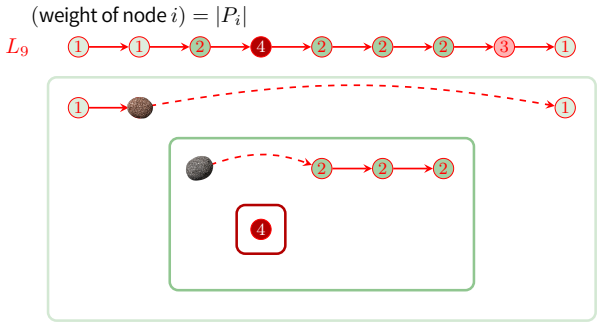




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

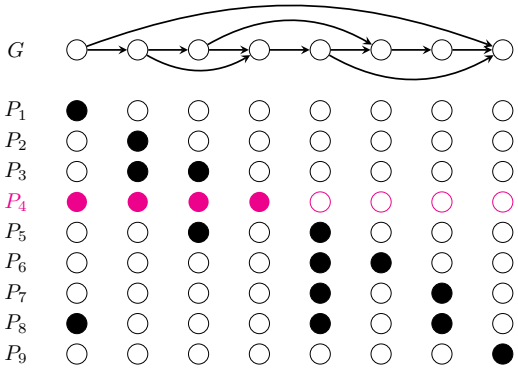
A weighted line graph fixes this problem

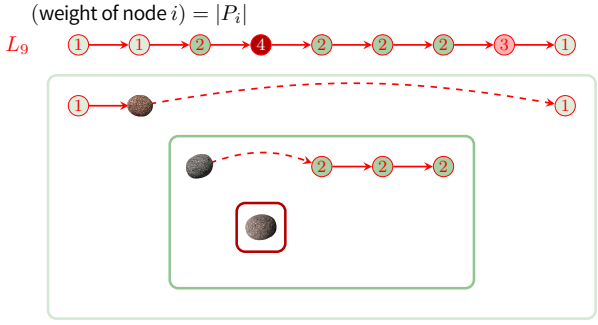




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

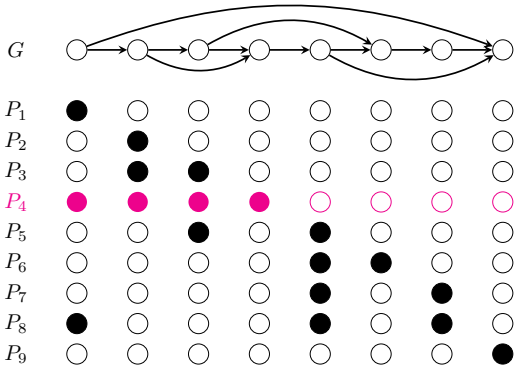
A weighted line graph fixes this problem

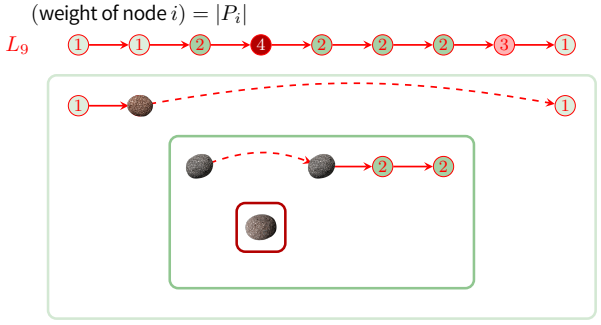




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

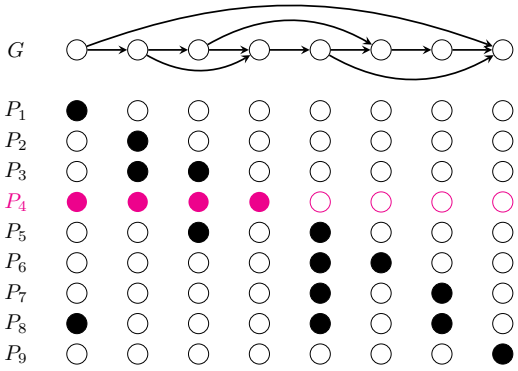
A weighted line graph fixes this problem

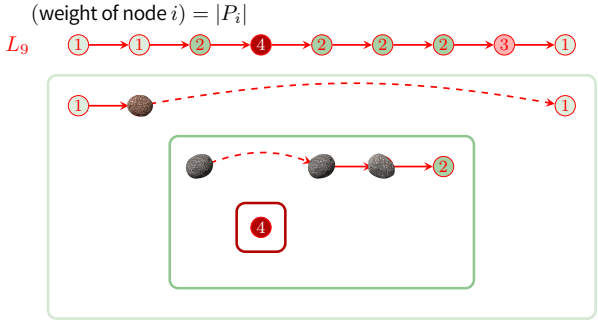




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

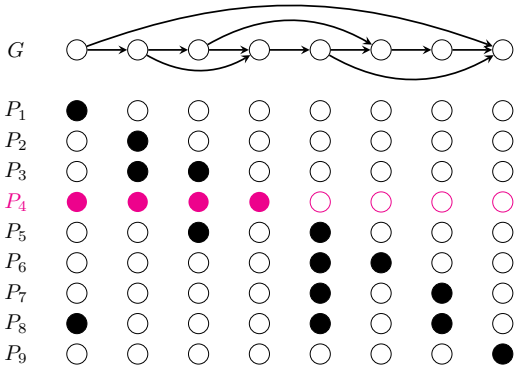
A weighted line graph fixes this problem

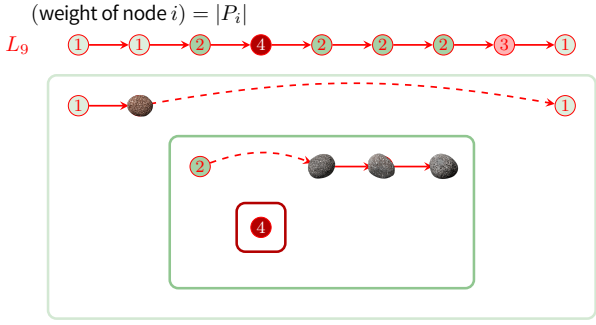




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

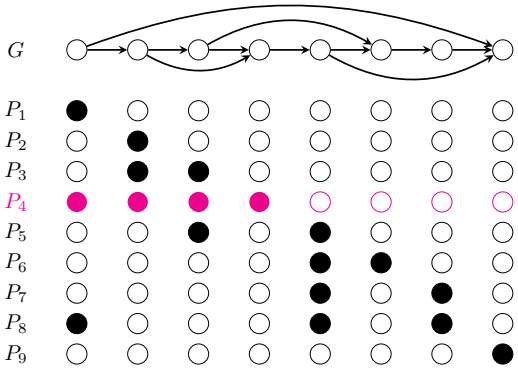
A weighted line graph fixes this problem



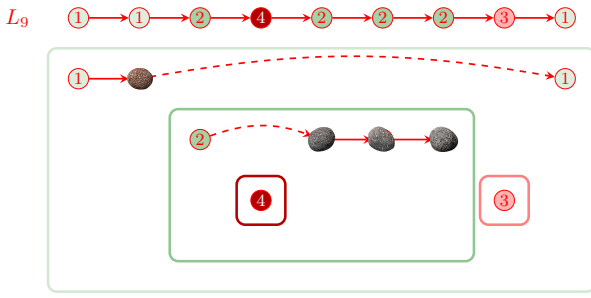


Idea: Recursively pebble/unpebble the **heavier** nodes as needed

A weighted line graph fixes this problem

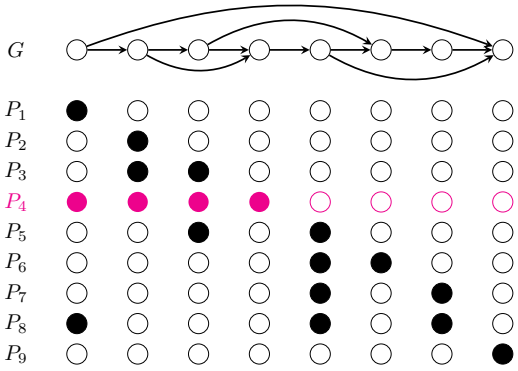


(weight of node i) $= |P_i|$

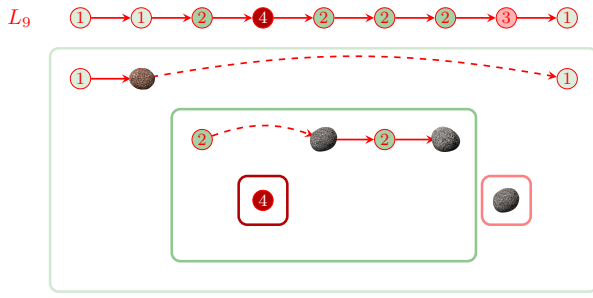


Idea: Recursively pebble/unpebble the **heavier** nodes as needed

A weighted line graph fixes this problem

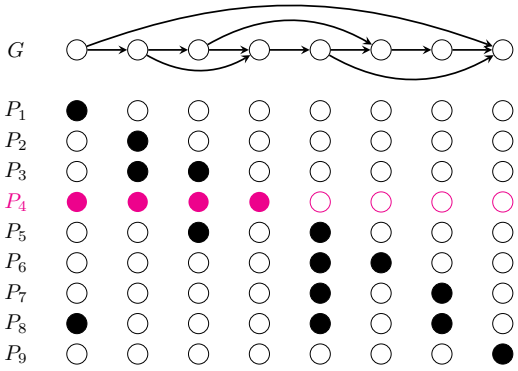


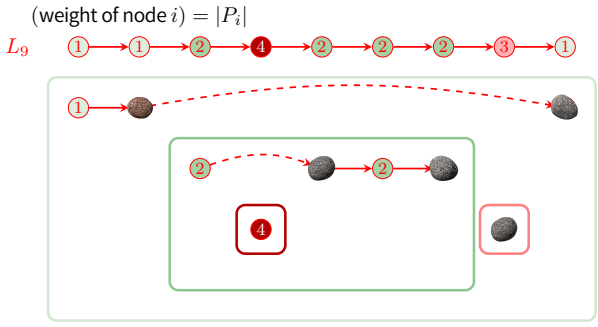
(weight of node i) $= |P_i|$



Idea: Recursively pebble/unpebble the **heavier** nodes as needed

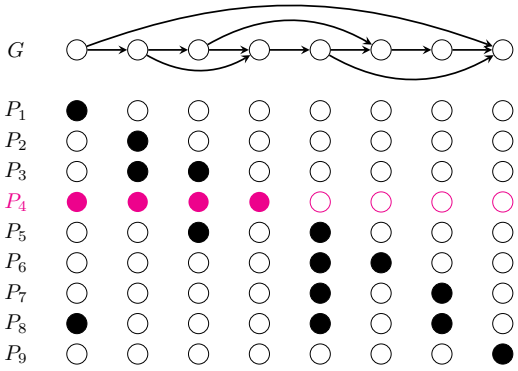
A weighted line graph fixes this problem



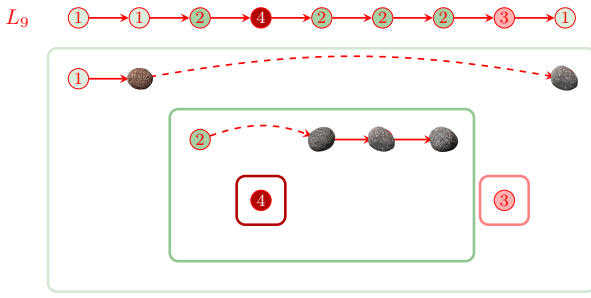


Idea: Recursively pebble/unpebble the **heavier** nodes as needed

A weighted line graph fixes this problem

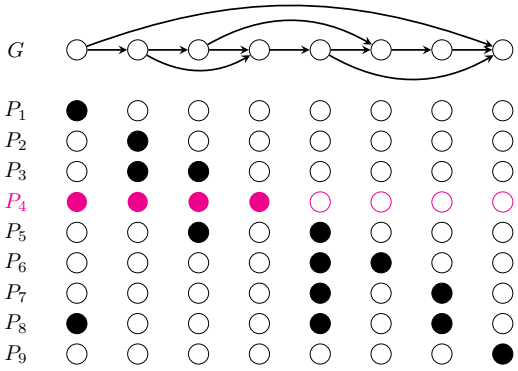


(weight of node i) $= |P_i|$

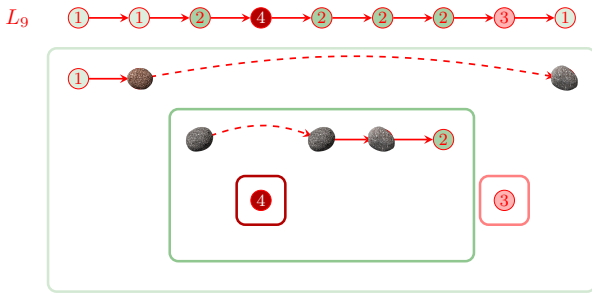


Idea: Recursively pebble/unpebble the **heavier** nodes as needed

A weighted line graph fixes this problem

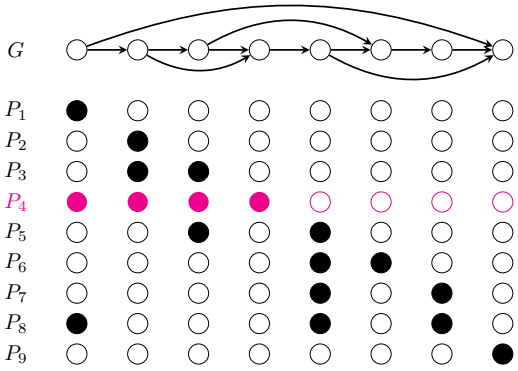


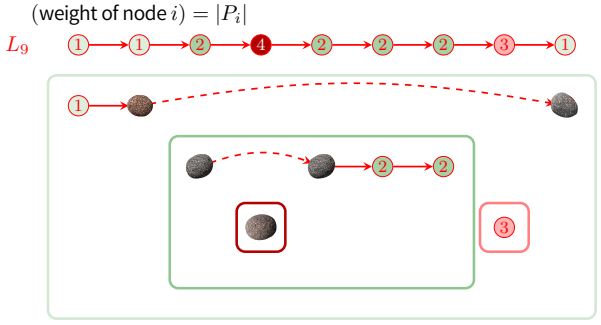
(weight of node i) $= |P_i|$



Idea: Recursively pebble/unpebble the **heavier** nodes as needed

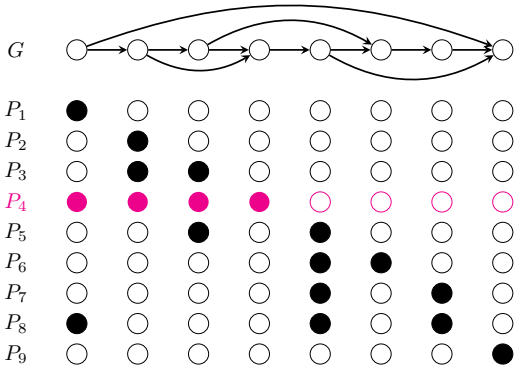
A weighted line graph fixes this problem

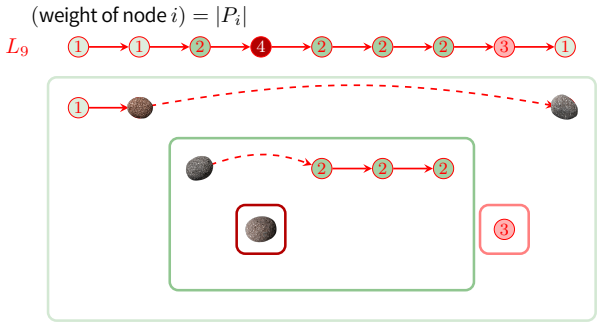




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

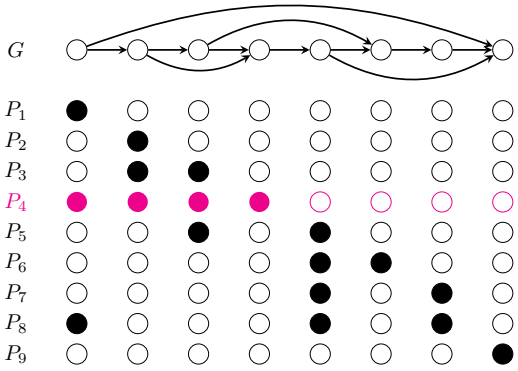
A weighted line graph fixes this problem

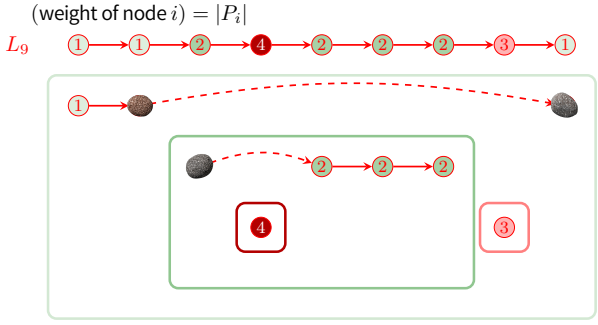




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

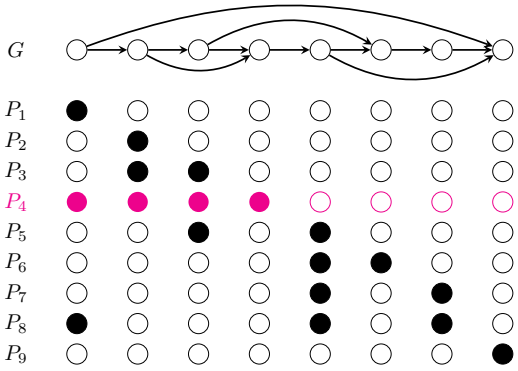
A weighted line graph fixes this problem

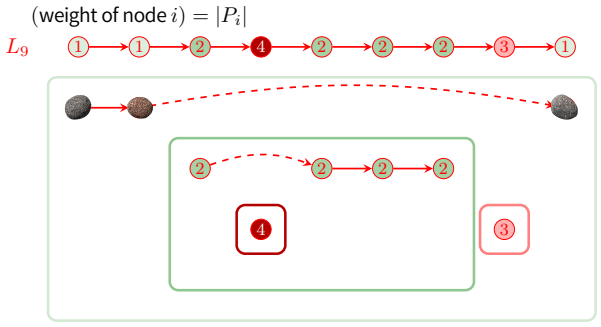




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

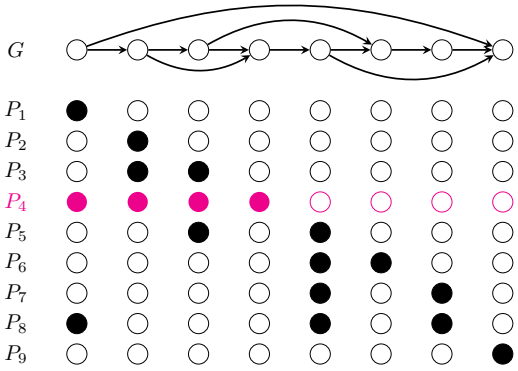
A weighted line graph fixes this problem

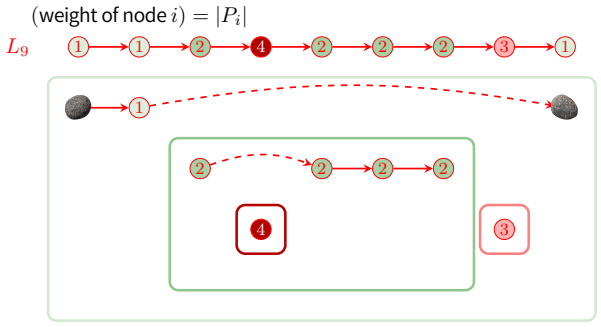




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

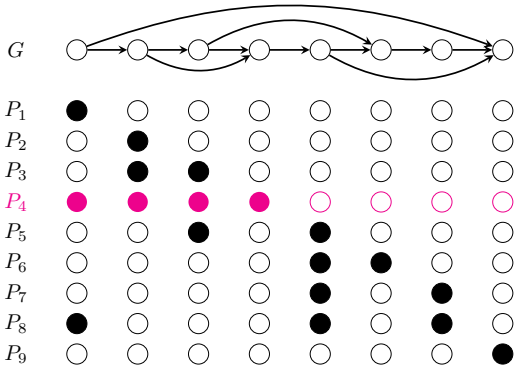
A weighted line graph fixes this problem

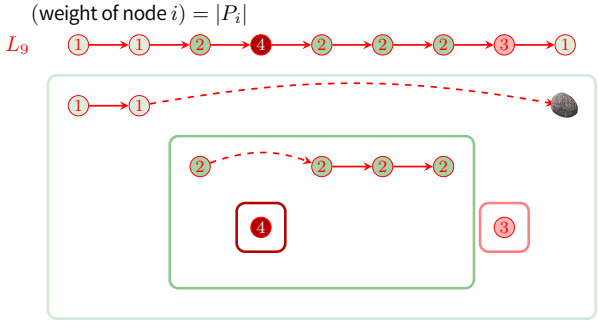




Idea: Recursively pebble/unpebble the **heavier** nodes as needed

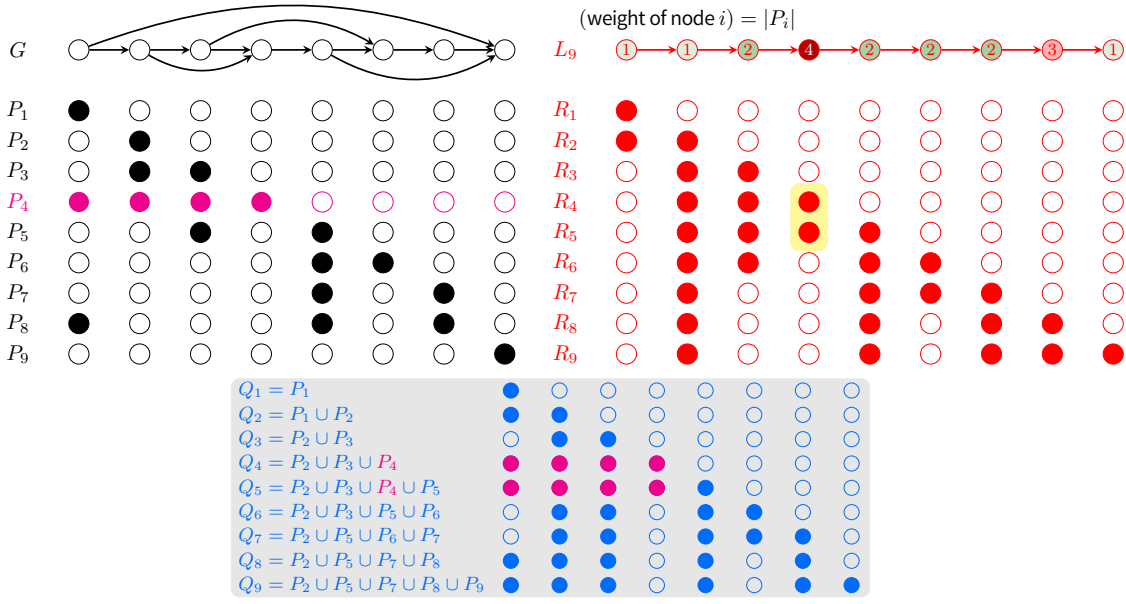
A weighted line graph fixes this problem



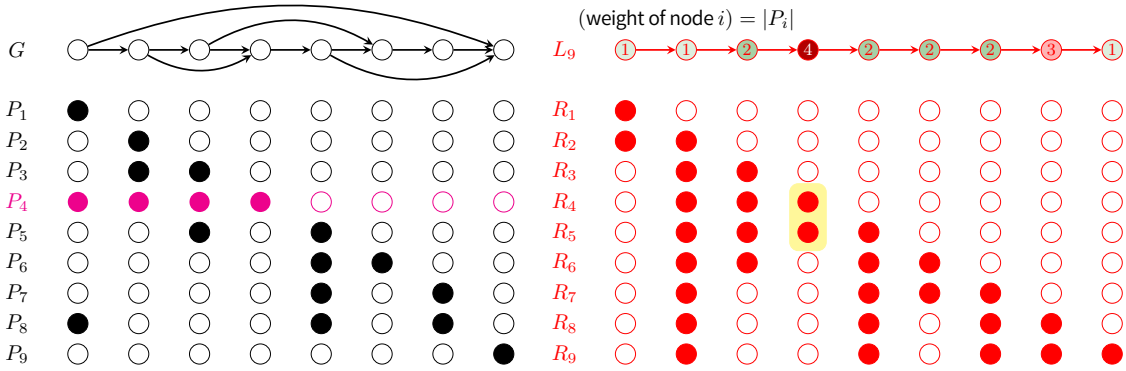


Idea: Recursively pebble/unpebble the **heavier** nodes as needed

A weighted line graph fixes this problem



A weighted line graph fixes this problem



Our Result [BHL25]

For a DAG G with N nodes, $\mathsf{CC}^{\parallel,\leftrightarrows}(G) = \mathcal{O}\left(N^{\frac{\mathcal{O}(1)}{\sqrt[4]{\log N}}}\right) \cdot \mathsf{CC}^{\parallel}(G)$.

Three takeaways and open problems

LANGUAGE The **parallel reversible pebbling game** captures both cleanup and parallelism

CONCRETE BOUNDS Reversibility already causes **superlinear cost** on a line graph

REUSABLE TOOL Classical pebbling strategies can be **transformed** into reversible ones with only **subpolynomial overhead**

Three takeaways and open problems

LANGUAGE The **parallel reversible pebbling game** captures both cleanup and parallelism

CONCRETE BOUNDS Reversibility already causes **superlinear cost** on a line graph

REUSABLE TOOL Classical pebbling strategies can be **transformed** into reversible ones with only **subpolynomial overhead**

Open Problems

- Can we tighten the gap between the upper and lower bounds for line graphs?

Three takeaways and open problems

LANGUAGE The **parallel reversible pebbling game** captures both cleanup and parallelism

CONCRETE BOUNDS Reversibility already causes **superlinear cost** on a line graph

REUSABLE TOOL Classical pebbling strategies can be **transformed** into reversible ones with only **subpolynomial overhead**

Open Problems

- Can we tighten the gap between the upper and lower bounds for line graphs?
- Can we sharpen the cumulative-cost overhead in the pebbling composition?

Three takeaways and open problems

LANGUAGE The **parallel reversible pebbling game** captures both cleanup and parallelism

CONCRETE BOUNDS Reversibility already causes **superlinear cost** on a line graph

REUSABLE TOOL Classical pebbling strategies can be **transformed** into reversible ones with only **subpolynomial overhead**

Open Problems

- Can we tighten the gap between the upper and lower bounds for line graphs?
- Can we sharpen the cumulative-cost overhead in the pebbling composition?
- Can parallel reversible pebbling be formulated and analyzed as an optimization problem whose relaxations yield useful bounds?

Three takeaways and open problems

LANGUAGE The **parallel reversible pebbling game** captures both cleanup and parallelism

CONCRETE BOUNDS Reversibility already causes **superlinear cost** on a line graph

REUSABLE TOOL Classical pebbling strategies can be **transformed** into reversible ones with only **subpolynomial overhead**

Open Problems

- Can we tighten the gap between the upper and lower bounds for line graphs?
- Can we sharpen the cumulative-cost overhead in the pebbling composition?
- Can parallel reversible pebbling be formulated and analyzed as an optimization problem whose relaxations yield useful bounds?
- Can similar techniques be extended to other pebbling settings, such as red-blue pebbling or spooky pebbling?

References I


Joël Alwen, Jeremiah Blocki, and Ben Harsha, **Practical graphs for optimal side-channel resistant memory-hard functions**, ACM CCS 2017 (Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, eds.), ACM Press, October / November 2017, pp. 1001--1017.


Charles H. Bennett, **Time/space trade-offs for reversible computation**, SIAM J. Comput. **18** (1989), no. 4, 766–776.


Jeremiah Blocki, Blake Holman, and Seunghoon Lee, **The parallel reversible pebbling game: Analyzing the post-quantum security of iMHFs**, TCC 2022, Part I (Eike Kiltz and Vinod Vaikuntanathan, eds.), LNCS, vol. 13747, Springer, Cham, November 2022, pp. 52--79.


Jeremiah Blocki, Blake Holman, and Seunghoon Lee, **The impact of reversibility on parallel pebbling**, EUROCRYPT 2025, Part I (Serge Fehr and Pierre-Alain Fouque, eds.), LNCS, vol. 15601, Springer, Cham, May 2025, pp. 124--154.


Richard Král'ovič, **Time and space complexity of reversible pebbling**, SOFSEM 2001: Theory and Practice of Informatics (Berlin, Heidelberg) (Leszek Pacholski and Peter Ružička, eds.), Springer Berlin Heidelberg, 2001, pp. 292--303.


Ming Li and Paul Vitányi, **Reversibility and adiabatic computation: Trading time and space for energy**, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452** (1996), no. 1947, 769–789.


Giulia Meuli, Mathias Soeken, Martin Roetteler, Nikolaj Bjorner, and Giovanni De Micheli, **Reversible pebbling game for quantum memory management**, 2019 Design, Automation Test in Europe Conference Exhibition (DATE), 2019, pp. 288--291.