Undirected Graph Exploration with $\Theta(\log\log n)$ Pebbles

A Grimm idea

Christoph Welzel

May 13, 2016

Logik und Theorie diskreter Systeme, RWTH Aachen

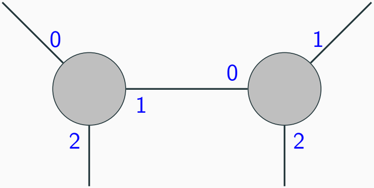
Introduction

- Exploration of huge finite graphs by agents
- Agents are located on a vertex and move along edges
- Agents can drop pebbles on vertices
- Agent *explores* a graph by systematically visiting all of its vertices

Preliminaries

Huge graphs

- Undirected
- Finite, but huge
- Indistinguishable vertices: 
- Bounded degree (Δ)
- Edges can locally be labeled with $0, \ldots, \Delta - 1$ (Port)



Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex

- As an agent moves onto a vertex it observes:

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
- As an agent moves onto a vertex it observes:
  - Port it entered through

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex

- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex
  - Degree of vertex

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
  - δ_{out}: leaving vertex

- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex
  - Degree of vertex

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
  - δ_{out}: leaving vertex

- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex
  - Degree of vertex
- Computation on a vertex determines:

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
  - δ_{out}: leaving vertex

- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex
  - Degree of vertex
- Computation on a vertex determines:
  - Port to leave

Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
  - δ_{out}: leaving vertex

- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex
  - Degree of vertex
- Computation on a vertex determines:
  - Port to leave
  - Pebbles to drop

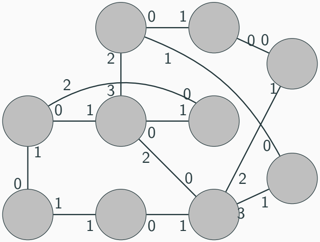
Agents

- Agent carries set of distinguishable pebbles
- Formalisation as pebble machines
  - $(Q, F, P, m, \delta, \delta_{in}, \delta_{out}, q_0)$
  - (s, p, m)-pebble machine
  - δ_{in}: moving onto vertex
  - δ_{out}: leaving vertex

- As an agent moves onto a vertex it observes:
  - Port it entered through
  - Pebbles it carries
  - Pebbles on the vertex
  - Degree of vertex
- Computation on a vertex determines:
  - Port to leave
  - Pebbles to drop
  - Pebbles to carry along

Exploration sequence

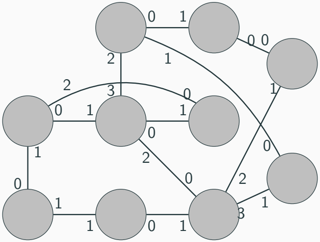
- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class



Exploration sequence

- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class

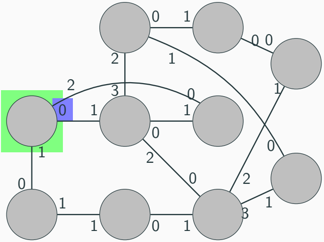
→ Example: 0 , 3 , 1 , 2



Exploration sequence

- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class

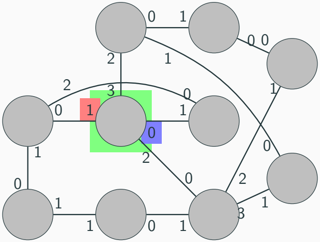
→ Example: 0, 3, 1, 2



Exploration sequence

- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class

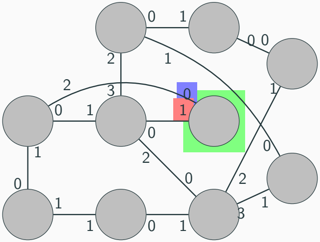
→ Example: 0 , 3 , 1 , 2



Exploration sequence

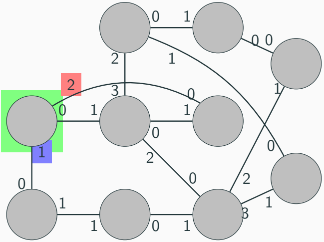
- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class

$\rightarrow$ Example: 0, 3, 1, 2



- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class

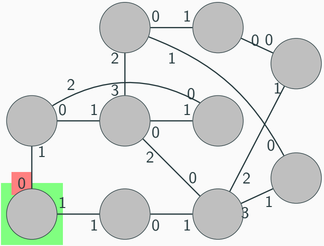
→ Example: 0 , 3 , 1 , 2



Exploration sequence

- Exploration sequence $e_1, \ldots, e_n$
- Relative movements of agent
- Leaving port: $\ell_i + e_i \mod d_v$
- *Universal* for a class of graphs if it explores any graph of that class

$\rightarrow$ Example: 0 , 3 , 1 , 2



Graph traversal

Exploring pebble machine

Theorem (Reingold)

There is an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence for any regular graph on n vertices.

Theorem

There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine that moves along a closed walk and either explores the graph or visits at least n distinct vertices, for any graph with bounded degree.

Exploring pebble machine

Theorem (Reingold)

There is an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence for any regular graph on n vertices.

$\boxed{\text{3-Regularity}}$

Theorem

There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine that moves along a closed walk and either explores the graph or visits at least n distinct vertices, for any graph with bounded degree.

Exploring pebble machine

Theorem (Reingold)

There is an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence for any regular graph on n vertices.

3-Regularity

Periodic Universal Lemma

Theorem

There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine that moves along a closed walk and either explores the graph or visits at least n distinct vertices, for any graph with bounded degree.

Exploring pebble machine

Theorem (Reingold)
There is an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence for any regular graph on n vertices.

3-Regularity

Periodic Universal Lemma

Closed Walk Lemma

Theorem
There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine that moves along a closed walk and either explores the graph or visits at least n distinct vertices, for any graph with bounded degree.

Closed walk

Lemma (Closed Walk Lemma)
An agent following an exploration sequence of the form $(e_0, \ldots, e_{k-1})^*$ *moves along a closed walk.*

Closed walk

Lemma (Closed Walk Lemma)

An agent following an exploration sequence of the form $(e_0, \ldots, e_{k-1})^*$ moves along a closed walk.

Lemma (Periodic Universal Lemma)

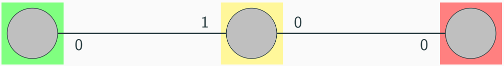
There exists an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence $(e_0, \ldots, e_{k-1})^*$ for any 3-regular graph with at most n vertices.

3-Regularity

Establish 3-Regularity

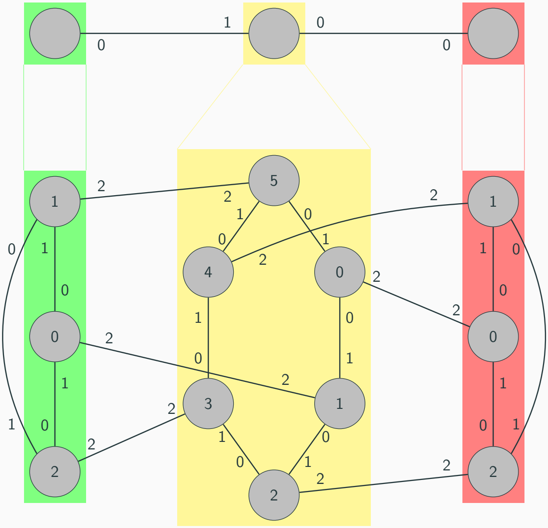
3-Regularity

Establish 3-Regularity



$\color{blue}{3}$-Regularity

Establish $\color{blue}{3}$-Regularity



Exploring pebble machine

Theorem (Reingold)

There is an $\mathcal{O}(\log n)$-space algorithm producing a universal exploration sequence for any regular graph on n vertices.

3-Regularity

Periodic Universal Lemma

Closed Walk Lemma

Theorem

There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$-pebble machine that moves along a closed walk and either explores the graph or visits at least n distinct vertices, for any graph with bounded degree.

Pebble simulation

Theorem

There is a constant $c \in \mathbb{N}$*, such that for any graph* G *with bounded degree and any* $(s, p, 2m)$*-pebble machine* $\mathcal{M}$*, there exists a* $(cs, p + c, m)$*-pebble machine* $\mathcal{M}'$ *that simulates the walk of* $\mathcal{M}$ *or explores* G*.*

Tape simulation via pebbles

- Simulate m bits by pebbles:

Tape simulation via pebbles

- Simulate m bits by pebbles:

$$\overbrace{\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline\end{array}}^{m = 12}$$

Tape simulation via pebbles

- Simulate m bits by pebbles:
    - Separate into m_1 sized blocks

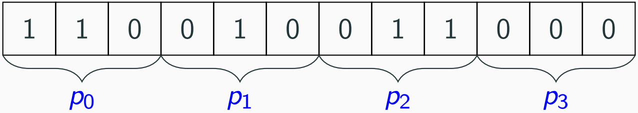
$$\overbrace{\underbrace{1\;1\;0}_{m_1 = 3}\;\underbrace{0\;1\;0}_{m_1 = 3}\;\underbrace{0\;1\;1}_{m_1 = 3}\;\underbrace{0\;0\;0}_{m_1 = 3}}^{m = 12}$$

Tape simulation via pebbles

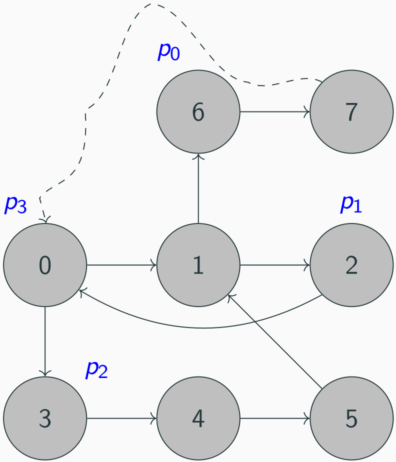
- Simulate m bits by pebbles:
  - Separate into m_1 sized blocks
  - Represent blocks by pebbles $p_0, \ldots, p_{\frac{m}{m_1}-1}$

$$\overbrace{\underbrace{\boxed{1}\boxed{1}\boxed{0}}_{p_0}\underbrace{\boxed{0}\boxed{1}\boxed{0}}_{p_1}\underbrace{\boxed{0}\boxed{1}\boxed{1}}_{p_2}\underbrace{\boxed{0}\boxed{0}\boxed{0}}_{p_3}}^{m=12}$$

Tape simulation via pebbles



- Pebble needs 2^{m_1} distinct "states"
- State as index of closed walk



Mechanisms of simulation

1. Simulation of M_{walk} providing walk with 2^{m_1} distinct vertices
2. Identify *distinct* vertices of walk
3. Read from and write to simulated tape
4. Preserve tape content through steps of simulated agent

Mechanisms of simulation

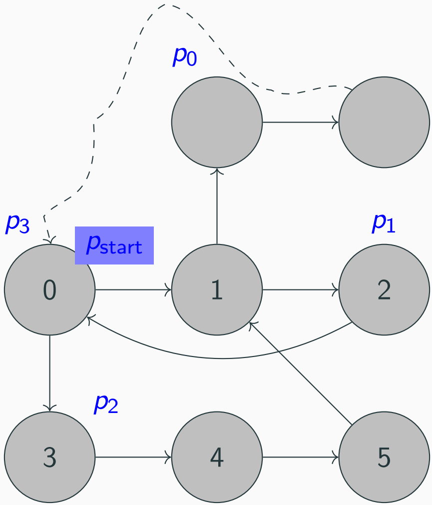
1. Simulation of M_{walk} providing walk with 2^{m_1} distinct vertices
2. Identify *distinct* vertices of walk
3. Read from and write to simulated tape
4. Preserve tape content through steps of simulated agent

2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

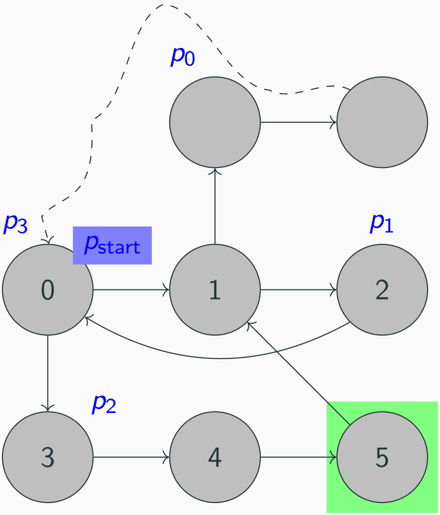


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

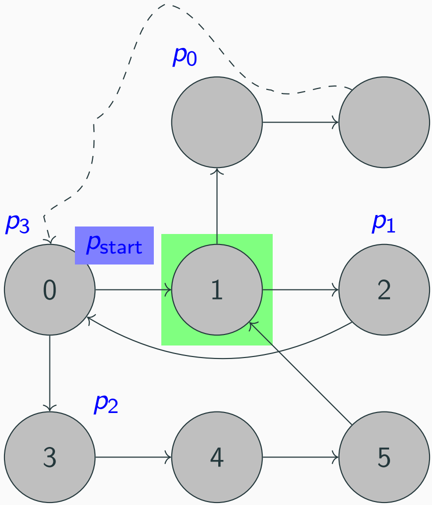


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

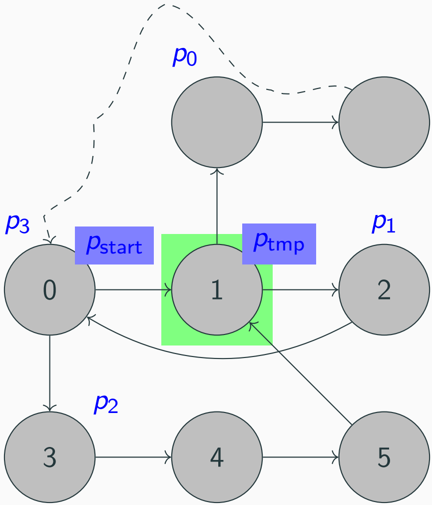
1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over



Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex “new”?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

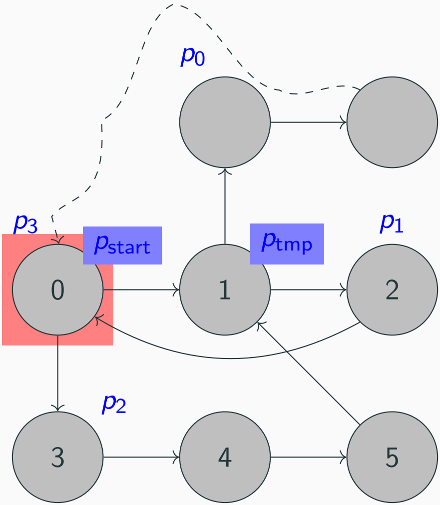


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

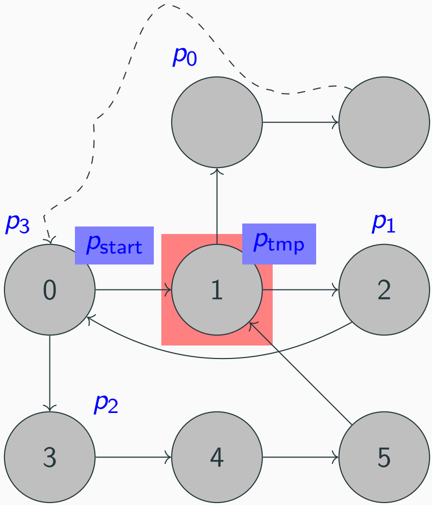


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

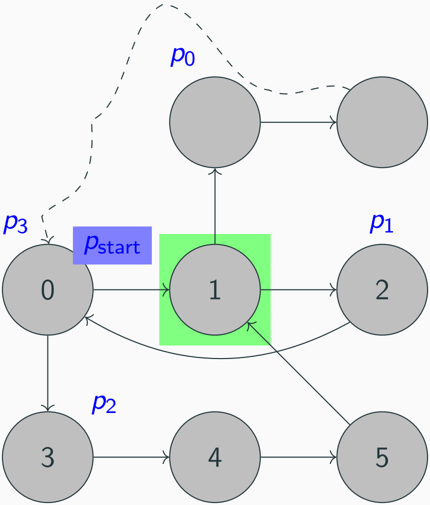


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find $p_{\text{tmp}} \rightarrow$ new vertex
7. Otherwise start over

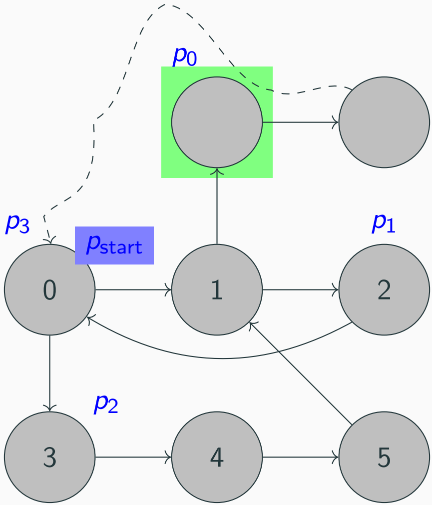


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

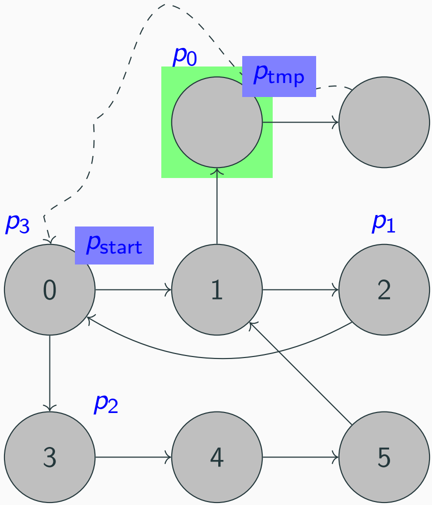


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex “new”?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

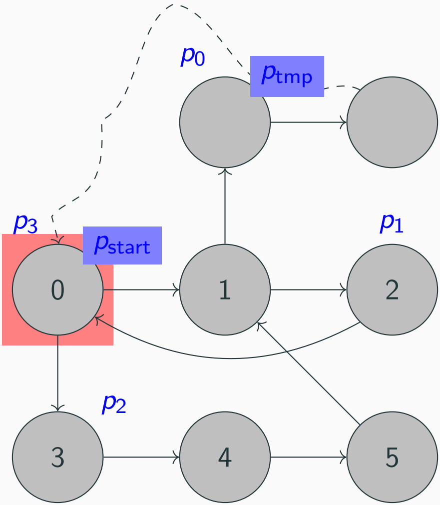


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

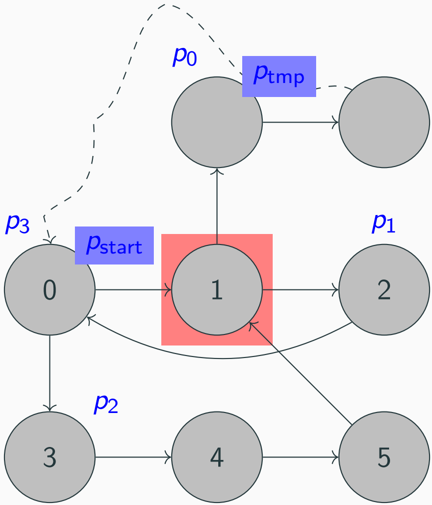


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex “new”?

Solution:

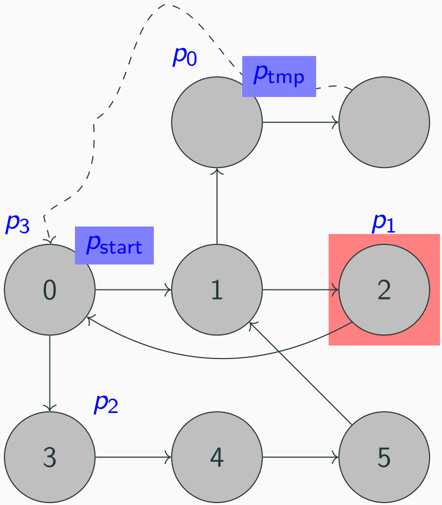
1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over



Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

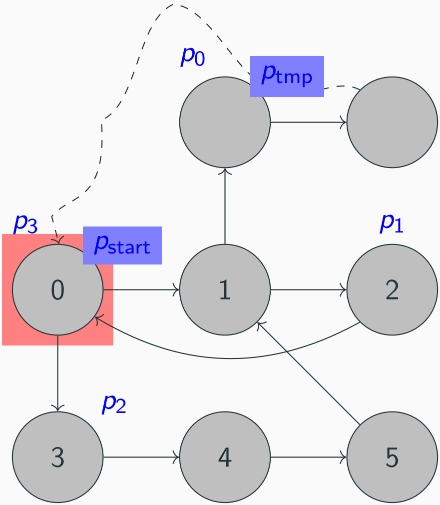


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

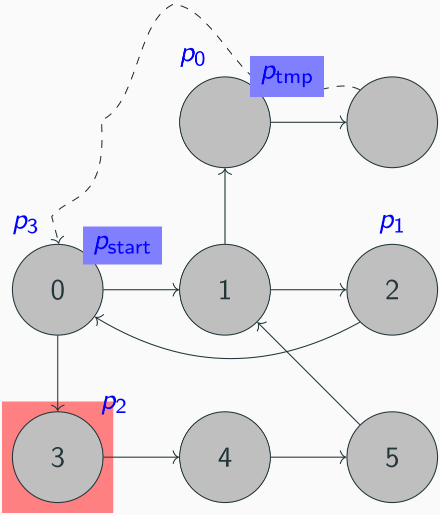


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

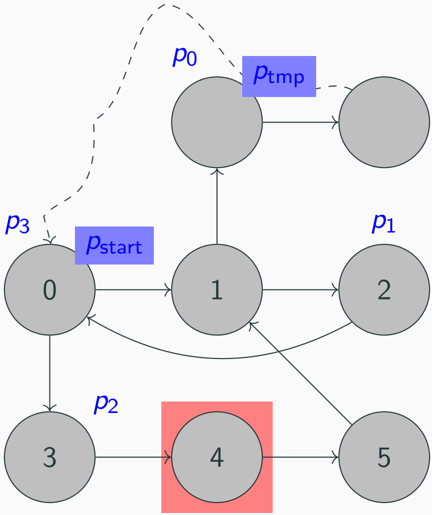


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find $p_{\text{tmp}} \rightarrow$ new vertex
7. Otherwise start over

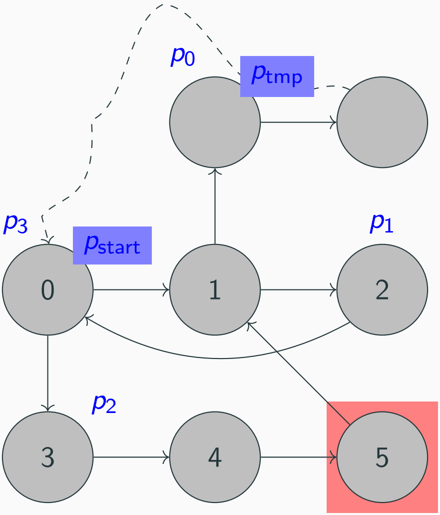


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

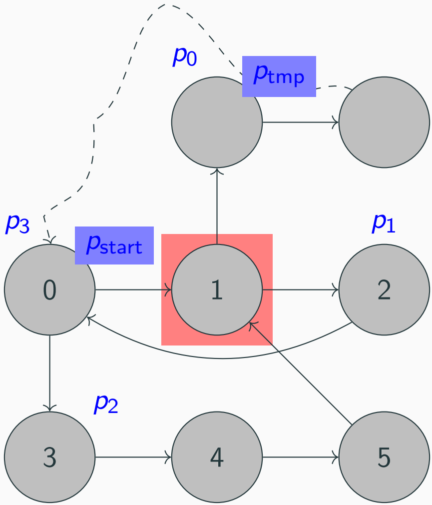


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

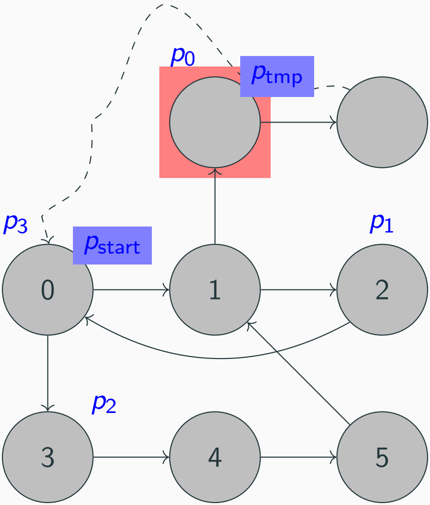


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

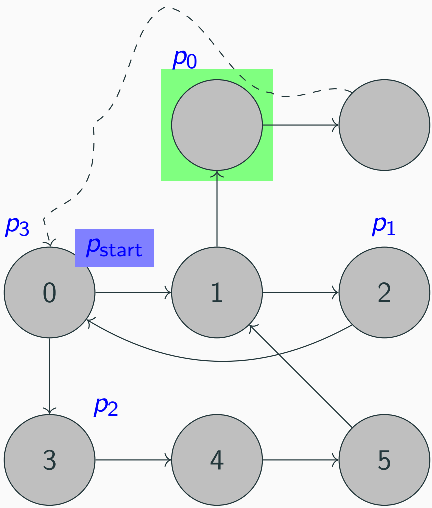


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over

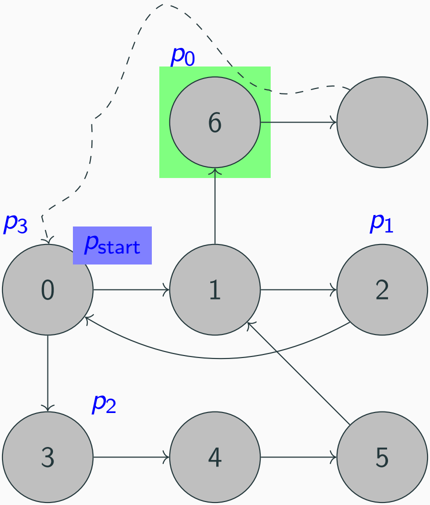


2. Finding distinct vertices

Problem: Indistinguishable vertices $\rightsquigarrow$ next vertex "new"?

Solution:

1. Execute step of M_{walk}
2. Drop pebble p_{tmp}
3. Store number of steps M_{walk} executed
4. Restart M_{walk}
5. Execute steps of M_{walk} until p_{tmp} is observed
6. Same amount of steps for M_{walk} to find p_{tmp} $\rightarrow$ new vertex
7. Otherwise start over



Simulation of a pebble machine

- Simulation of M_{Walk} which explores at least 2^{m_1} distinct vertices
- Identify *distinct* vertices of the walk
- Read from and write to simulated tape
- Preserve tape content through steps of simulated agent

Simulation of a pebble machine

- Simulation of M_{Walk} which explores at least 2^{m_1} distinct vertices $\checkmark$
- Identify *distinct* vertices of the walk
- Read from and write to simulated tape
- Preserve tape content through steps of simulated agent

Simulation of a pebble machine

- Simulation of M_{Walk} which explores at least 2^{m_1} distinct vertices ✓
- Identify *distinct* vertices of the walk ✓
- Read from and write to simulated tape
- Preserve tape content through steps of simulated agent

Simulation of a pebble machine

- Simulation of M_{Walk} which explores at least 2^{m_1} distinct vertices $\checkmark$
- Identify *distinct* vertices of the walk $\checkmark$
- Read from and write to simulated tape $\checkmark$
- Preserve tape content through steps of simulated agent

Simulation of a pebble machine

- Simulation of M_{Walk} which explores at least 2^{m_1} distinct vertices ✓
- Identify *distinct* vertices of the walk ✓
- Read from and write to simulated tape ✓
- Preserve tape content through steps of simulated agent ✓

Simulation of a pebble machine

- Simulation of M_{Walk} which explores at least 2^{m_1} distinct vertices ✓
- Identify *distinct* vertices of the walk ✓
- Read from and write to simulated tape ✓
- Preserve tape content through steps of simulated agent ✓

Theorem

There is a constant $c \in \mathbb{N}$, such that for any graph G with bounded degree and any $(s, p, 2m)$-pebble machine $\mathcal{M}$, there exists a $(cs, p + c, m)$-pebble machine $\mathcal{M}'$ that simulates the walk of $\mathcal{M}$ or explores G.

Exploring graphs with $\log \log n$ pebbles

Theorem

There exists a $(\mathcal{O}(1), 0, \mathcal{O}(\log n))$*-pebble machine that moves along a closed walk and either explores the graph or visits at least* n *distinct vertices, for any graph with bounded degree.*

$\boxed{\text{Apply simulation Theorem } \log \log n \text{ times}}$

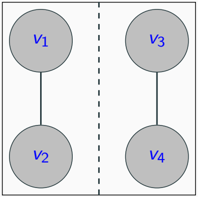
Theorem

Any bounded-degree graph on at most n *vertices can be explored using* $\mathcal{O}(\log \log n)$ *pebbles and memory.*

Limitations of pebble machines

Limitations of pebbles

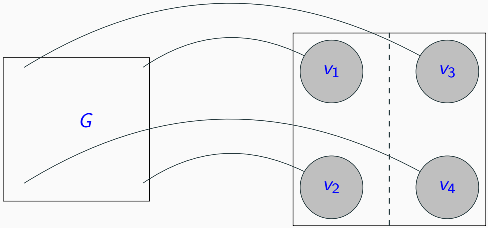
- Construct 3-regular graphs which an agent $\mathcal{A}$ with p pebbles cannot traverse
- Called p-barriers



- For every pair (a, b) in $\{v_1, v_2\} \times \{v_3, v_4\}$ $\mathcal{A}$ cannot traverse a p-barrier from a to b or vice versa using at most p pebbles
- Inductive construction of barriers

Limitations of pebbles

- Construct 3-regular graphs which an agent $\mathcal{A}$ with p pebbles cannot traverse
- Called p-barriers



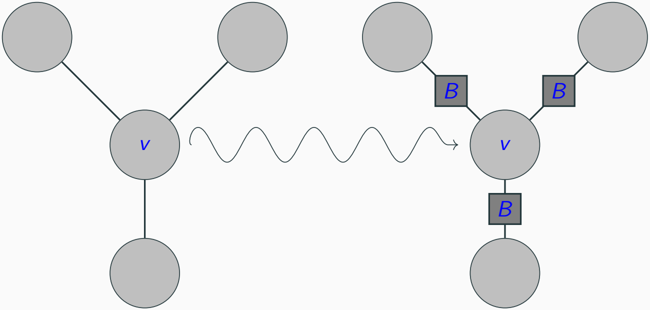
- For every pair (a, b) in $\{v_1, v_2\} \times \{v_3, v_4\}$ $\mathcal{A}$ cannot traverse a p-barrier from a to b or vice versa using at most p pebbles
- Inductive construction of barriers

Locality gadget

- Replace edge by $(r-1)$-barrier
- Enforce "locality" of pebbles

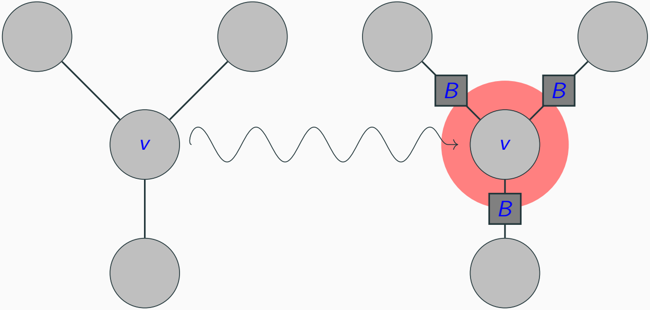
Locality gadget

- Replace edge by $(r-1)$-barrier
- Enforce "locality" of pebbles



Locality gadget

- Replace edge by $(r-1)$-barrier
- Enforce "locality" of pebbles



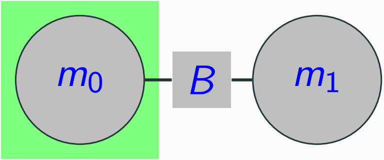
Theorem

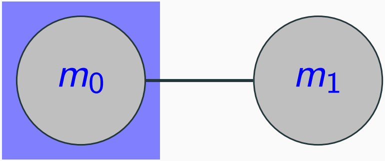
Given an $(r-1)$-barrier B with m vertices for an agent $\mathcal{A}$ with $p \geq r$ pebbles, we can construct an r-barrier B' with $\mathcal{O}(\binom{p}{r} \cdot m \cdot \alpha_B^2)$ vertices for $\mathcal{A}$.

where α_B is the amount of "local" configurations for $\mathcal{A}$ with r pebbles

Barrier construction: Idea

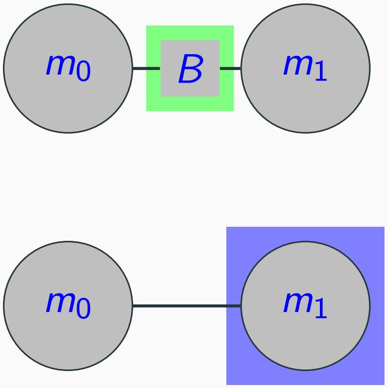
- Subset of pebbles of cardinality r
- Locality gadget as edges
- Pebbles are local: configurations in macro vertex $C_1, \ldots, C_\alpha$
- Project A to agent without pebbles B
- B has α states and traverses macro vertices





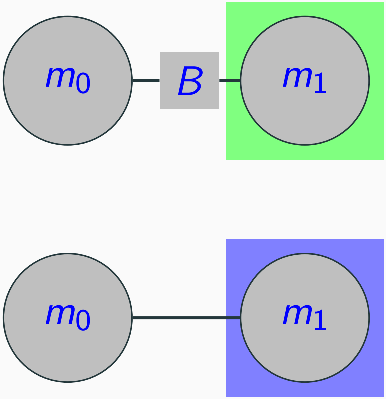
Barrier construction: Idea

- Subset of pebbles of cardinality r
- Locality gadget as edges
- Pebbles are local: configurations in macro vertex $C_1, \ldots, C_\alpha$
- Project A to agent without pebbles B
- B has α states and traverses macro vertices



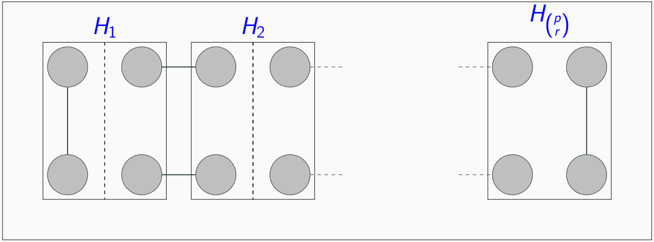
Barrier construction: Idea

- Subset of pebbles of cardinality r
- Locality gadget as edges
- Pebbles are local: configurations in macro vertex $C_1, \ldots, C_\alpha$
- Project A to agent without pebbles B
- B has α states and traverses macro vertices



Barrier construction

- B has no pebbles and cannot traverse 0-barrier
- r-barrier for chosen subset of pebbles
- Connect barriers for all possible subsets of pebbles with cardinality r:



Barrier base case

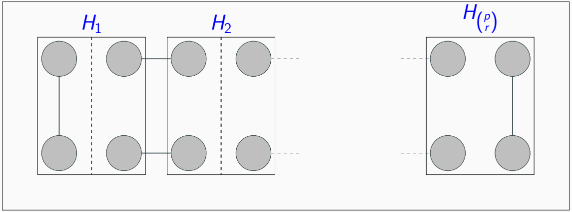
Theorem (Fraigniaud et al.)

For any q non cooperative s-state agents without pebbles, there exists a 3-regular graph G on $\mathcal{O}(qs)$ vertices with the following property: There are two edges $\{v_1, v_2\}$ and $\{v_3, v_4\}$ in G, the first labeled 0, such that any of the q agents starting in v_1 or v_2 does not traverse the edge $\{v_3, v_4\}$.

- For agent $\mathcal{A}$ consider $S = \{\mathcal{A}_q \mid q \in Q\}$
- Since $|S| = |Q| = q$ there is a 0-barrier with $\mathcal{O}(s^2)$ vertices

Theorem

Given an $(r-1)$*-barrier* B *with* m *vertices for an agent* $\mathcal{A}$ *with* $p \geq r$ *pebbles, we can construct an* r*-barrier* B' *with* $\mathcal{O}(\binom{p}{r} \cdot m \cdot \alpha_B^2)$ *vertices for* $\mathcal{A}$*.*



Lower pebble bound

Theorem

For $r \leq p$ and $s \geq 2^p$, the number of vertices of an r-barrier B_r for the s-state agent $\mathcal{A}$ with p pebbles is bounded by $\mathcal{O}(s^{8^{r+1}})$.

Theorem

For any constant $\epsilon > 0$, an agent with at most $\mathcal{O}((\log n)^{1-\epsilon})$ bits of memory needs at least $\Omega(\log \log n)$ distinguishable pebbles for exploring all graphs on at most n vertices.

Conclusion

Conclusion

- Exploring graphs with $\mathcal{O}(\log\log n)$ pebbles and memory
- Simulation via pebbles
- Limitations of pebbles: Barrier construction
- Every agent with sub-logarithmic memory needs $\Omega(\log\log n)$ pebbles to explore n vertices

Thank you for your attention.

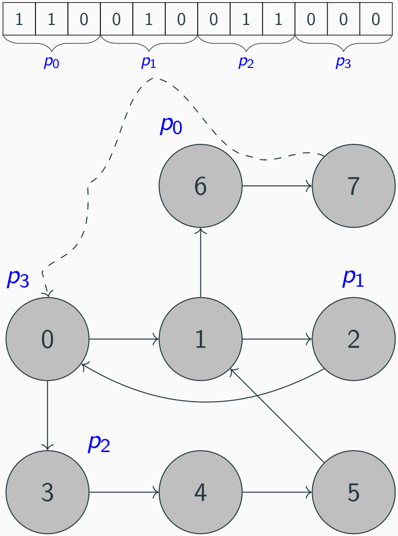
Slides and report on
https://github.com/weltoph/sem-pebbles.

1. Simulation of M_{walk}

- M_{Walk} implemented as $(\mathcal{O}(1), 0, \mathcal{O}(m_1))$-pebble machine
- Used variables/pebbles:
  - $\mathsf{T}_{\mathsf{walk}}$: stores M_{walk}'s tape
  - $\mathsf{T}_{\mathsf{steps}}$: stores number of taken steps
  - T_{id}: stores number of visited distinct vertices
  - p_{start}: marks beginning of walk

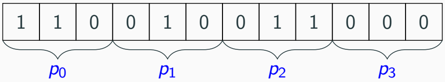
3. Memory management

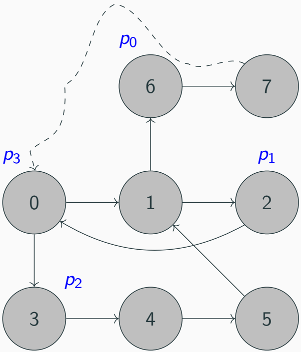
- $\mathsf{T}_{\mathsf{head}}$: head position
- $p_{\left\lfloor \frac{\mathsf{T}_{\mathsf{head}}}{m_1} \right\rfloor}$: pebble of memory block
- $\mathit{off} = \mathsf{T}_{\mathsf{head}} \mod m_1$: offset in memory block
- Reading:
  - Retrieve encoding pebble
  - Return off-th bit
- Writing:
  - Read bit
  - On bit-flip move encoding pebble 2^{off} bits up or down



3. Memory management: Writing

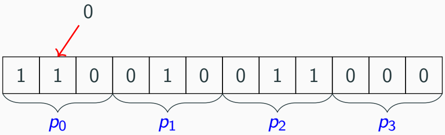
- Read bit
- On bit-flip: move pebble

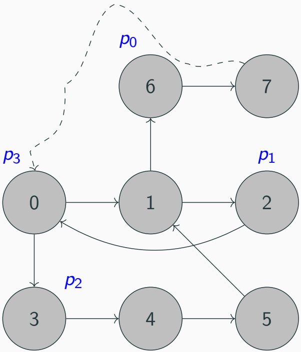




3. Memory management: Writing

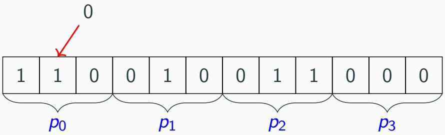
- Read bit
- On bit-flip: move pebble

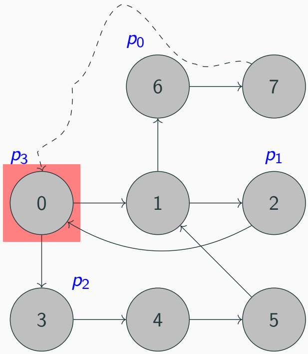




3. Memory management: Writing

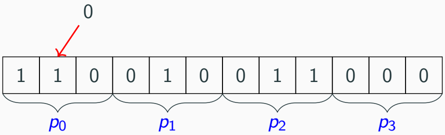
- Read bit
- On bit-flip: move pebble

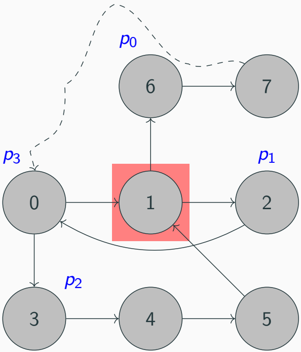




3. Memory management: Writing

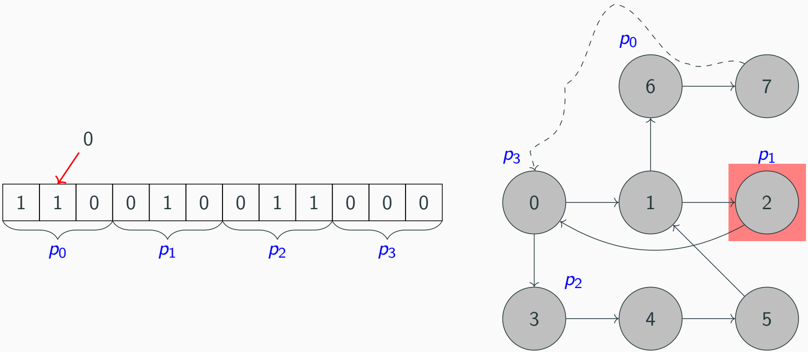
- Read bit
- On bit-flip: move pebble





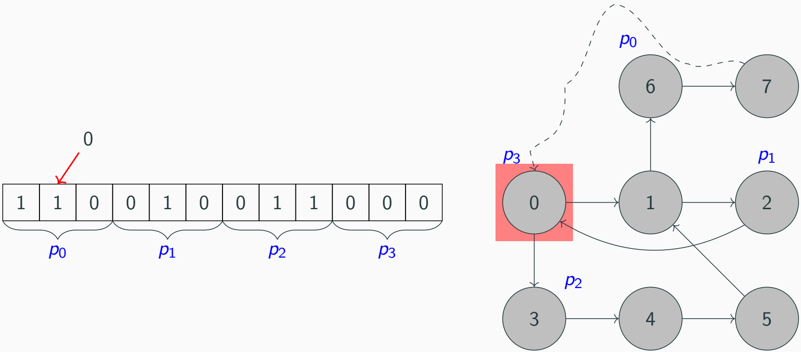
3. Memory management: Writing

- Read bit
- On bit-flip: move pebble



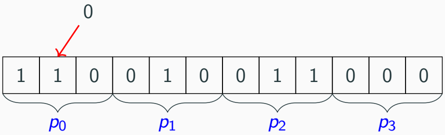
3. Memory management: Writing

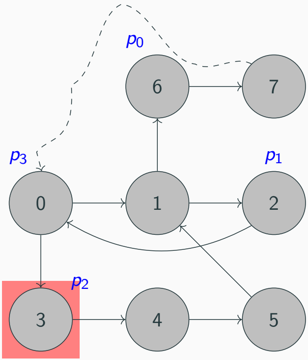
- Read bit
- On bit-flip: move pebble



3. Memory management: Writing

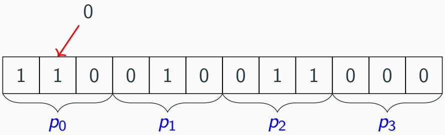
- Read bit
- On bit-flip: move pebble

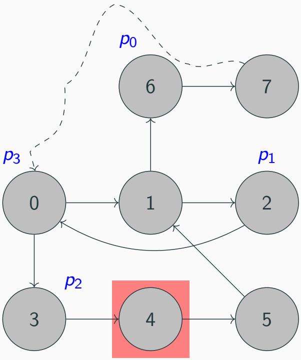




3. Memory management: Writing

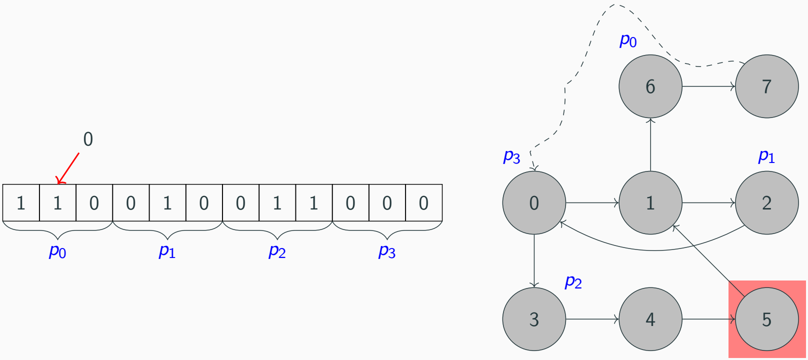
- Read bit
- On bit-flip: move pebble





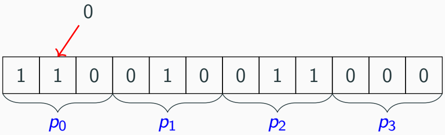
3. Memory management: Writing

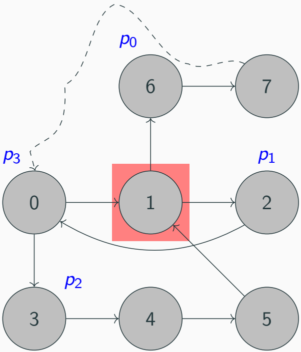
- Read bit
- On bit-flip: move pebble



3. Memory management: Writing

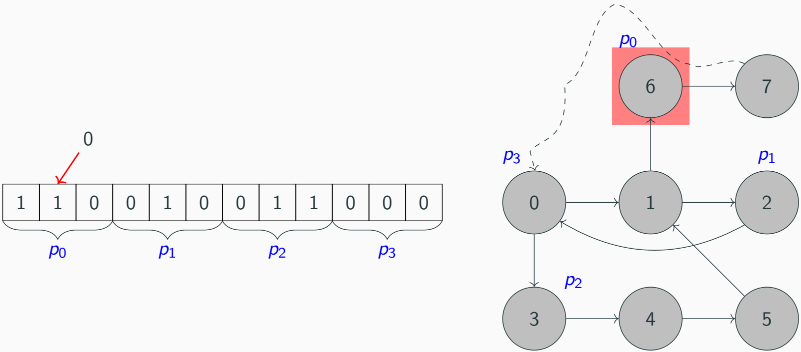
- Read bit
- On bit-flip: move pebble





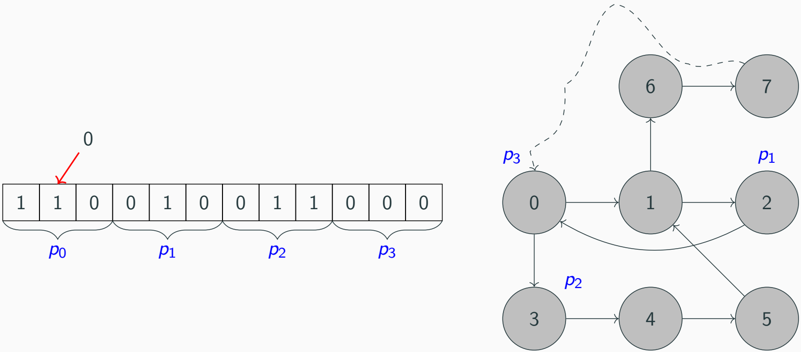
3. Memory management: Writing

- Read bit
- On bit-flip: move pebble



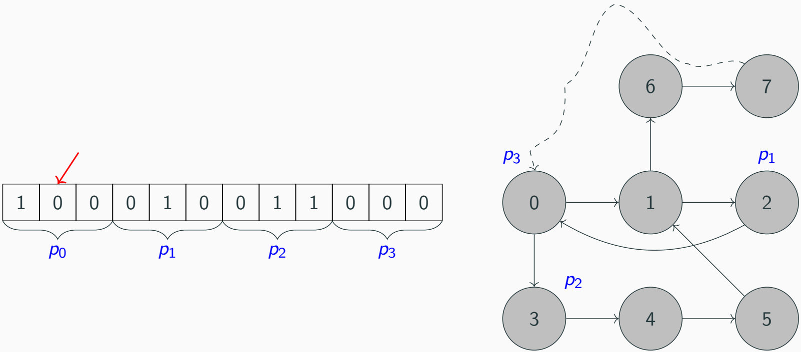
3. Memory management: Writing

- Read bit
- On bit-flip: move pebble



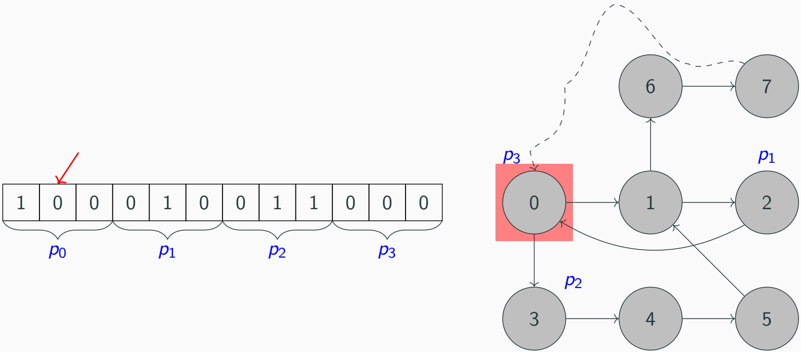
3. Memory management: Writing

- Read bit
- On bit-flip: move pebble



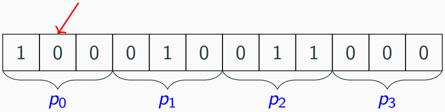
3. Memory management: Writing

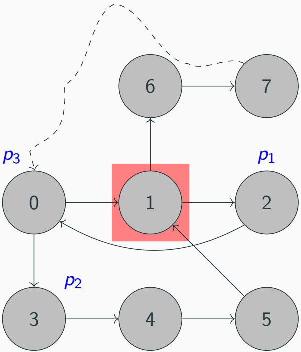
- Read bit
- On bit-flip: move pebble



3. Memory management: Writing

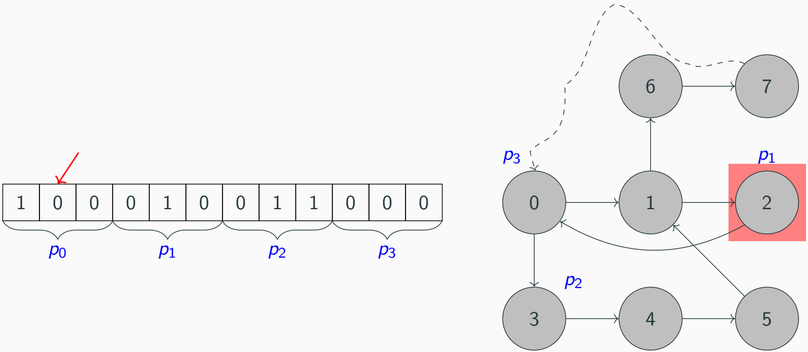
- Read bit
- On bit-flip: move pebble





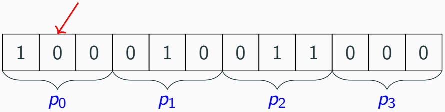
3. Memory management: Writing

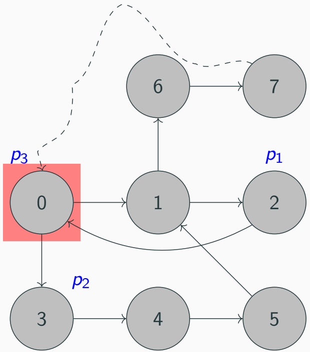
- Read bit
- On bit-flip: move pebble



3. Memory management: Writing

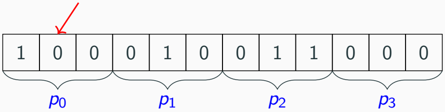
- Read bit
- On bit-flip: move pebble

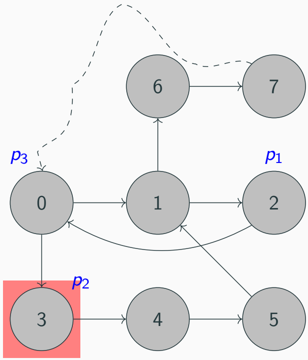




3. Memory management: Writing

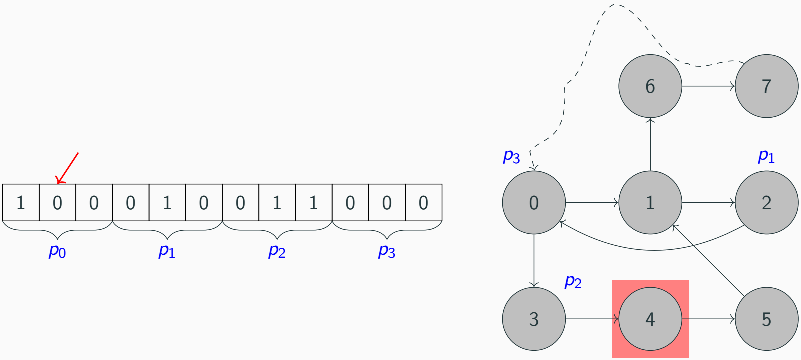
- Read bit
- On bit-flip: move pebble





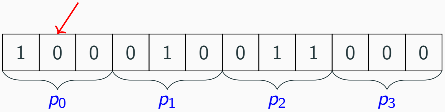
3. Memory management: Writing

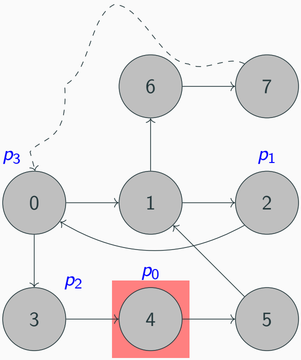
- Read bit
- On bit-flip: move pebble



3. Memory management: Writing

- Read bit
- On bit-flip: move pebble





4. Transfer tape-content through movements

- Simulated agent moves
- Preserve tape-content:
  - Drop p_{next} on next vertex v
  - ω' walk of M_{walk} starting from v
  - Transfer encoding pebbles onto ω'
  - Place pebbles at same index

Barrier size

Theorem

For $r \leq p$ and $s \geq 2^p$, the number of vertices of an r-barrier B_r for the s-state agent $\mathcal{A}$ with p pebbles is bounded by $\mathcal{O}(s^{8^{r+1}})$.