Pebble Motion Problems and Multi-robot Path Planning on Graphs

Jingjin Yu

Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology

July 30, 2014
Computational Geometry Seminar
Tel Aviv University
A connected graph: $G = (V, E)$
A connected graph: \( G = (V, E) \)

Configurations: \( S = \langle s_1, \ldots, s_p \rangle, \; D = \langle d_1, \ldots, d_p \rangle \)
A connected graph: $G = (V, E)$

Configurations: $S = \langle s_1, \ldots, s_p \rangle$, $D = \langle d_1, \ldots, d_p \rangle$

A pebble move:
A connected graph: $G = (V, E)$

Configurations: $S = \langle s_1, \ldots, s_p \rangle$, $D = \langle d_1, \ldots, d_p \rangle$

A pebble move: 1

**Pebble Motion on Graphs (PMG):** A single pebble move in a step

**Pebble Motion with Rotations (PMR):** Multiple pebbles may move in a step
**PMR** allows *synchronous rotations* of pebbles
A Unique Feature of PMR

**PMR** allows *synchronous rotations* of pebbles

**PMR** encompasses *highly constrained* path planning problems
15-Puzzle and \((N^2 - 1)\)-Puzzle

**15-puzzle**: A **PMG** instance \((G, S, D)\) with \(G\) a \(4 \times 4\) grid and \(p = 15\).

Generalization to the \((N^2 - 1)\)-**puzzle**
- An \(O(N^3)\) algorithm is easy: Move pebbles \(i\) to its goal takes \(O(N)\) moves
- Computing a solution with least number of moves is NP-hard (Goldreich 1984, Ratner and Warmuth 1990)
- An optimal solution for the 15-puzzle can be computed quickly. Optimally solving a 24-puzzle is still time-consuming
General Graphs

$G$ is 2-connected with $p = |V| - 1$
- Configurations $S_1$ and $S_2$ are *connected* if the PMG instance $(G, S_1, S_2)$ is feasible
- Configurations $S_1$ and $S_2$ are *adjacent* if they are connected via a single move
- $puz(G)$: The graph formed with configurations as vertices and adjacencies as edges
- Besides a few special cases, if $G$ is non-bipartite, then $puz(G)$ has a single component, if $G$ is bipartite, then $puz(G)$ has two components (Wilson 1974)

General graphs with $p \leq |V| - 1$ (PMG)
- Solvable in $O(|V|^3)$ time – asymptotically tight (Kornhauser et al. 1984)
- Feasibility test for trees can be done in linear time (Auletta et al. 1996)
- Feasibility test for graphs can also be done in linear time (Goraly and Hassin 2010, Yu 2013)
Applications

[Image of a busy roundabout with heavy traffic]
Applications
Applications
Applications
Applications
Applications
Outline

- Pebble Motion Problems

- Feasibility Test and Planning Algorithms for PMR
  - The Special Case of $p = |V|$
  - Linear Time Feasibility Test

- Optimal Pebble Motion
  - Computational Complexity (NP-Hardness)
  - Multiflow-Based Integer Linear Programming (ILP) Algorithm

- The Case of Indistinguishable Pebbles
✓ Pebble Motion Problems

☐ Feasibility Test and Planning Algorithms for PMR
  ☐ The Special Case of \( p = |V| \)
  ☐ Linear Time Feasibility Test

☐ Optimal Pebble Motion
  ☐ Computational Complexity (NP-Hardness)
  ☐ Multiflow-Based Integer Linear Programming (ILP) Algorithm

☐ The Case of Indistinguishable Pebbles

Dr. Steven LaValle
UIUC

Dr. Daniela Rus
MIT
\[ p = |V| = n \Rightarrow \text{only rotations are possible} \]
Rotation Induced Permutation Group

\[ p = |V| = n \Rightarrow \text{only rotations are possible} \]
\[ p = |V| = n \Rightarrow \text{only rotations are possible} \]
Rotation Induced Permutation Group

\[ p = |V| = n \Rightarrow \text{only rotations are possible} \]

A connected graph \( G \) induces a group \( G \) generated by \textit{cyclic rotations}.
Rotation Induced Permutation Group

\[ p = |V| = n \implies \text{only rotations are possible} \]

A connected graph \( G \) induces a group \( G \) generated by cyclic rotations
Rotation Induced Permutation Group

\[ p = |V| = n \Rightarrow \text{only rotations are possible} \]

A connected graph \( G \) induces a group \( G \) generated by \textit{cyclic rotations}

\[ G \cong \mathbb{Z}/5 \]
The *diameter* of a group $G$, $\text{diam}(G)$, is the length of the longest irreducible generator product needed to reach any group element from identity.
The diameter of a group $G$, $diam(G)$, is the length of the longest irreducible generator product needed to reach any group element from identity.
The *diameter* of a group $\mathbf{G}$, $diam(\mathbf{G})$, is the length of the longest irreducible generator product needed to reach any group element from identity.
The *diameter* of a group $\mathbf{G}$, $\text{diam}(\mathbf{G})$, is the length of the longest irreducible generator product needed to reach any group element from identity $1$

$$
\text{diam}(\mathbf{G}) = 2
$$
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $diam(G) = O(n^2)$. 
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $diam(G) = O(n^2)$.

$G \cong \{1\}$

$diam(G) = 1$
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $\text{diam}(G) = O(n^2)$.

- $G \cong \{1\}$
  
  $\text{diam}(G) = 1$

- $G \cong \mathbb{Z}/n$
  
  $\text{diam}(G) = O(n)$
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $diam(G) = O(n^2)$.

- $G \cong \{1\}$
  - $diam(G) = 1$
- $G \cong \mathbb{Z}/n$
  - $diam(G) = O(n)$
- $G \cong A_n$ or $S_n$
  - $diam(G) = O(n^2)$
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $diam(G) = O(n^2)$.

- $G \cong \{1\}$
  - $diam(G) = 1$

- $G \cong A_n$ or $S_n$
  - $diam(G) = O(n^2)$

- $G \cong \mathbb{Z}/n$
  - $diam(G) = O(n)$

- $G \cong S_n$
  - $diam(G) = O(n^2)$
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $\text{diam}(G) = O(n^2)$.

- $G \cong \{1\}$
  \[ \text{diam}(G) = 1 \]

- $G \cong \mathbb{Z}/n$
  \[ \text{diam}(G) = O(n) \]

- $G \cong A_n$ or $S_n$
  \[ \text{diam}(G) = O(n^2) \]

- $G \cong S_n$
  \[ \text{diam}(G) = O(n^2) \]

$G$ for a general graph $G$ is the direct product of these groups.
Theorem (Diameter of a Graph Induced Permutation Group). Let $G$ be the group for a graph $G = (V, E)$ with $|V| = n$. Then $diam(G) = O(n^2)$.

$G \cong \{1\}$

$diam(G) = 1$

$G \cong \mathbb{Z}/n$

$diam(G) = O(n)$

$G \cong A_n$ or $S_n$

$diam(G) = O(n^2)$

$G \cong S_n$

$diam(G) = O(n^2)$

$G$ for a general graph $G$ is the direct product of these groups.
Definition (Cycle Similarity). Two configurations are \textit{cycle similar} if the same pebble in the two configurations shares one common cycle.
Definition (Cycle Similarity). Two configurations are **cycle similar** if the same pebble in the two configurations shares one common cycle.

Lemma (Arrangement). Given a separable G with each edge on exactly one cycle and configurations $S$ and $D$, in $O(n^2)$ moves, a configuration that is cycle similar to $D$ is reachable from $S$. 
Definition (Cycle Similarity). Two configurations are cycle similar if the same pebble in the two configurations shares one common cycle.

Lemma (Arrangement). Given a separable G with each edge on exactly one cycle and configurations $S$ and $D$, in $O(n^2)$ moves, a configuration that is cycle similar to $D$ is reachable from $S$. 
Lemma (Rearrangement). The pebbles arranged according to the Arrangement Lemma can be rearranged such that the resulting configuration is the same as $D$ or differ from $D$ by a fixed transposition of two neighboring pebbles in $D$. Rearrangement requires $O(n^2)$ moves.
Lemma (Rearrangement). The pebbles arranged according to the Arrangement Lemma can be rearranged such that the resulting configuration is the same as $D$ or differ from $D$ by a fixed transposition of two neighboring pebbles in $D$. Rearrangement requires $O(n^2)$ moves.
Constructive proof of $G \geq A_n$ and $diam(G) = O(n^2)$ for a cactus $G$

- Build a cycle tree of the cycles on the cactus
- Arrange $S$ to $D'$ that it is cycle similar to $D$ in $O(n^2)$ steps
- Rearrange $D'$ to $D''$ that is identical to $D$ or differ from $D$ only at two adjacent pebbles, again in $O(n^2)$ steps
Algorithm for Feasibility and Path Planning
Move $S$ and $D$ so that the pebbles occupy the same set of vertices and concentrate the pebbles to 2-edge-connected components.
Move $S$ and $D$ so that the pebbles occupy the same set of vertices and concentrate the pebbles to 2-edge-connected components

$G$ is contracted into a skeleton tree by shrinking 2-edge-connected components
Move $S$ and $D$ so that the pebbles occupy the same set of vertices and concentrate the pebbles to 2-edge-connected components.

- $G$ is *contracted* into a *skeleton tree* by shrinking 2-edge-connected components.

- Apply the PMT (PMG with $G$ being a tree) feasibility test (Auletta et al. 1996).
Move $S$ and $D$ so that the pebbles occupy the same set of vertices and concentrate the pebbles to 2-edge-connected components

$G$ is expressed in a skeleton tree by shrinking 2-edge-connected components

Apply the PMT (PMG with $G$ being a tree) feasibility test (Auletta et al. 1996)

This yields in linear time the feasibility and the set of pebble exchanges needed to take $S$ to $D$
Move $S$ and $D$ so that the pebbles occupy the same set of vertices and concentrate the pebbles to 2-edge-connected components.

$G$ is contracted into a skeleton tree by shrinking 2-edge-connected components.

Apply the PMT (PMG with $G$ being a tree) feasibility test (Auletta et al. 1996).

This yields in linear time the feasibility and the set of pebble exchanges needed to take $S$ to $D$.

Each pebble exchange is doable in $O(n)$ steps and $O(n^2)$ total pebble moves, yielding an $O(n^3)$ planning algorithm.
Outline

- Pebble Motion Problems

- Feasibility Test and Planning Algorithms for PMR
  - The Special Case of $p = |V|$
  - Linear Time Feasibility Test

- Optimal Pebble Motion
  - Computational Complexity (NP-Hardness)
  - Multiflow-Based Integer Linear Programming (ILP) Algorithm

- The Case of Indistinguishable Pebbles
Optimal Pebble Motion Problems

Let $t_i$ be the arrival time of pebble $i$ at its goal (the pebble is then fixed)

- **Minimum makespan**: Find a plan that minimizes $\max t_i$
- **Minimum total arrival time**: Find a plan that minimizes $\sum t_i$
- **Minimum total distance**: Find a plan that minimizes the total number of edges travelled by all pebbles
Proposition (Incompatibility of Objectives). In general, any two of the minimum makespan, minimum total arrival time, and minimum total distance objectives cannot be simultaneous satisfied.
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

3SAT Variables: $x_1, ..., x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4$
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

3SAT Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4$

Minimum makespan PMR
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

3SAT Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4$, $\neg x_1 \lor x_2 \lor \neg x_4$, $\neg x_2 \lor x_3 \lor x_4$

Minimum makespan PMR
**Theorem (NP-Hardness).** Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

**3SAT**  Variables: $x_1, \ldots, x_4$, Clauses:  
\[
    x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4
\]

**Minimum makespan PMR**
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all \( \text{NP-hard} \).

**3SAT** Variables: \( x_1, \ldots, x_4 \), Clauses: \( x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4 \)

Minimum makespan PMR
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

3SAT Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4$, $\neg x_1 \lor x_2 \lor \neg x_4$, $\neg x_2 \lor x_3 \lor x_4$
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

3SAT  Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4$

Minimum makespan PMR
**Theorem (NP-Hardness).** Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

**3SAT** Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4$

Minimum makespan PMR
**Theorem (NP-Hardness).** Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

**3SAT** Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4$

**Minimum makespan PMR**

A possible assignment: $x_1 = x_2 = x_3 = x_4 = 1$
Theorem (NP-Hardness). Planning paths for PMR to minimize makespan, total arrival time, and total distance objectives are all NP-hard.

3SAT Variables: $x_1, \ldots, x_4$, Clauses: $x_1 \lor \neg x_3 \lor x_4, \neg x_1 \lor x_2 \lor \neg x_4, \neg x_2 \lor x_3 \lor x_4$

Minimum makespan PMR

A possible assignment:
$x_1 = x_2 = x_3 = x_4 = 1$
Corollary. Planning for PMR to minimize makespan remains NP-Hard for two types of pebbles.
Algorithmic Solution for Optimal PMR

Key idea: time expansion
Algorithmic Solution for Optimal PMR

Key idea: time expansion

\[ a_1, a_2, a'_2, a'_1 \]
Algorithmic Solution for Optimal PMR

Key idea: time expansion

\[ a_1 \quad a_2' \]
\[ a_1' \quad a_2 \]

\[ t = 0 \quad 1 \quad 1' \quad 2 \quad 2' \quad 3 \quad 3' \quad T=4 \quad 4' \]
Algorithmic Solution for Optimal PMR

Key idea: time expansion
Algorithmic Solution for Optimal PMR

Key idea: time expansion
Algorithmic Solution for Optimal PMR

Key idea: time expansion

Theorem. Fixing a natural number $T$, a PMR instance admits a solution with at most $T$ time steps if and only if the corresponding time-expanded network with $T$ periods admits a solution consisting of disjoint paths.
Algorithm for Minimum Makespan PMR

1. Pick an initial $T$
2. Build the time-expanded network
3. Set up an ILP model
4. Run optimizer
   - Feasible? Yes → Return the path set
   - Feasible? No → $T \leftarrow T + 1$
Algorithm for Minimum Makespan PMR

1. Pick an initial $T$
2. Build the time-expanded network
3. Set up an ILP model
4. Run optimizer
   - Feasible? If yes, return the path set;
   - If no, $T = T + 1$ and repeat with the updated $T$.

Additional heuristics:
- Reachability analysis
- Divide and conquer
Computational Performance (MinTime MPPG)

16 × 16 grid, 60 pebbles

Longest path Length = 22
Computed min time = 22
Running time: < 1 second

Problems can be solved under 1 second: ~100 pebbles, ~1000 vertices
100+ times faster than the next best (Surynek 2012)
Outline

- Pebble Motion Problems
- Feasibility Test and Planning Algorithms for PMR
  - The Special Case of $p = |V|$  
  - Linear Time Feasibility Test
- Optimal Pebble Motion
  - Computational Complexity (NP-Hardness)  
  - Multiflow-Based Integer Linear Programming (ILP) Algorithm
- The Case of Indistinguishable Pebbles
Always feasible
In fact, many optimal versions (MinTime, MinTotalDist, ...) are efficiently solvable.
Always feasible

In fact, many optimal versions (MinTime, **MinTotalDist**, ...) are efficiently solvable.
Compute lengths of all shortest paths between starts and goals
Computing a Shortest Unscheduled Path Set

- Compute lengths of all shortest paths between starts and goals
- Compute lengths of all shortest paths between starts and goals

<table>
<thead>
<tr>
<th></th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>$v_2$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$v_3$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Computing a Shortest Unscheduled Path Set

- Compute lengths of all shortest paths between starts and goals

- Run **Hungarian** algorithm to find an optimal pairing

```
<table>
<thead>
<tr>
<th></th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>v2</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>v3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
```
Computing a Shortest Unscheduled Path Set

- Compute lengths of all shortest paths between starts and goals

- Run **Hungarian** algorithm to find an optimal pairing

<table>
<thead>
<tr>
<th></th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>$v_2$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$v_3$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Computing a Shortest Unscheduled Path Set

- Compute lengths of all shortest paths between starts and goals

<table>
<thead>
<tr>
<th></th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>$v_2$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$v_3$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

- Run **Hungarian** algorithm to find an optimal pairing
Computing a Shortest Unscheduled Path Set

- Compute lengths of all shortest paths between starts and goals

- Run **Hungarian** algorithm to find an optimal pairing
Computing a Shortest Unscheduled Path Set

- Compute lengths of all shortest paths between starts and goals

<table>
<thead>
<tr>
<th></th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>$v_2$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$v_3$</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

- Run Hungarian algorithm to find an optimal pairing

optimal (shortest) but unscheduled path set $Q = \{ q_1, \ldots, q_n \}$
Lemma: A shortest path set $Q = \{ q_1, \ldots, q_p \}$ induces a directed acyclic graph structure.
Lemma: A shortest path set $Q = \{ q_1, \ldots, q_p \}$ induces a directed acyclic graph structure.
Lemma: A shortest path set $Q = \{ q_1, ..., q_p \}$ induces a directed acyclic graph structure.
Lemma: A shortest path set $Q = \{q_1, \ldots, q_p\}$ induces a directed acyclic graph structure.
Lemma: A shortest path set $Q = \{q_1, \ldots, q_p\}$ induces a directed acyclic graph structure.
Corollary: Fixing a shortest path set $Q$, there exists a goal vertex that does not lie on any other paths (i.e., it is not in the middle of another path of $Q$).
Corollary: Fixing a shortest path set $Q$, there exists a goal vertex that does not lie on any other paths (i.e., it is not in the middle of another path of $Q$).
**Corollary:** Fixing a shortest path set $Q$, there exists a goal vertex that does not lie on any other paths (i.e., it is not in the middle of another path of $Q$).
**Corollary:** Fixing a shortest path set $Q$, there exists a goal vertex that does not lie on any other paths (i.e., it is not in the middle of another path of $Q$).
Corollary: Fixing a shortest path set $Q$, there exists a goal vertex that does not lie on any other paths (i.e., it is not in the middle of another path of $Q$).
Scheduling the Paths (Coordination and Control)
Scheduling the Paths (Coordination and Control)
Scheduling the Paths (Coordination and Control)
Scheduling the Paths (Coordination and Control)
Scheduling the Paths (Coordination and Control)
Theorem (Bounded Completion Time for Distance Optimal Paths): Given a PMR instance with indistinguishable pebbles, there exists a minimum total distance plan with a completion time of no more than $p + \ell - 1$, in which $p$ is the number of pebbles and $\ell$ is the longest distance between any start and goal vertices. Moreover, the $p + \ell - 1$ time bound is tight.
Algorithm 1  Distance Optimal Planner for PMR with Indistinguishable Pebbles

Input: Graph $G$, start vertices $x_I$, goal vertices $x_G$.  \hspace{5mm} \textbf{Output:} path set $P = \{ p_1, ..., p_p \}$

1. Compute $dist(u, v)$ for all $u \in x_I, v \in x_G$
2. Run Hungarian method to pair up pebbles and goals, yielding \textit{unscheduled} paths $Q = \{ q_1, ..., q_p \}$
3. \textbf{for} each $t = 1: p$ standalone goal (a goal of a path $q_i$ that is not on other paths $q_j \neq q_i$)
4. Do a backward search (limited to edges of the path set) for a closest vehicle
5. Set the vehicle to start at time $t - 1$, yielding a \textit{scheduled} path $p_t$
6. \textbf{return} $P = \{ p_1, ..., p_p \}$
Algorithm 1 Distance Optimal Planner for PMR with Indistinguishable Pebbles

**Input:** Graph $G$, start vertices $x_I$, goal vertices $x_G$.  **Output:** path set $P = \{ p_1, \ldots, p_p \}$

1. Compute $dist(u, v)$ for all $u \in x_I, v \in x_G$
2. Run Hungarian method to pair up pebbles and goals, yielding unscheduled paths $Q = \{ q_1, \ldots, q_p \}$
3. for each $t = 1: p$ standalone goal (a goal of a path $q_i$ that is not on other paths $q_j \neq q_i$)
4. Do a backward search (limited to edges of the path set) for a closest vehicle
5. Set the vehicle to start at time $t - 1$, yielding a scheduled path $p_t$
6. **return** $P = \{ p_1, \ldots, p_p \}$

- **Running time:** $O(p^3)$ – determined by the Hungarian algorithm
**Algorithm 1** Distance Optimal Planner for PMR with Indistinguishable Pebbles

**Input:** Graph $G$, start vertices $x_I$, goal vertices $x_G$.  
**Output:** path set $P = \{p_1, ..., p_p\}$

1. Compute $\text{dist}(u, v)$ for all $u \in x_I, v \in x_G$
2. Run Hungarian method to pair up pebbles and goals, yielding *unscheduled* paths $Q = \{q_1, ..., q_p\}$
3. for each $t = 1: p$ standalone goal (a goal of a path $q_i$ that is not on other paths $q_j \neq q_i$)
4. Do a backward search (limited to edges of the path set) for a closest vehicle
5. Set the vehicle to start at time $t - 1$, yielding a *scheduled* path $p_t$
6. return $P = \{p_1, ..., p_p\}$

- **Running time:** $O(p^3)$ – determined by the Hungarian algorithm
- The scheduling/control portion can be fully decentralized
  - Requires only local (distance 2) communication
  - Retains the $p + \ell - 1$ completion time
  - Generally yields better than $p + \ell - 1$ completion time
### Algorithm 1 Distance Optimal Planner for PMR with Indistinguishable Pebbles

**Input:** Graph $G$, start vertices $x_I$, goal vertices $x_G$.  
**Output:** path set $P = \{p_1, ..., p_p\}$

1. Compute $\text{dist}(u, v)$ for all $u \in x_I, v \in x_G$
2. Run Hungarian method to pair up pebbles and goals, yielding unscheduled paths $Q = \{q_1, ..., q_p\}$

3. for each $t = 1: p$ standalone goal (a goal of a path $q_i$ that is not on other paths $q_j \neq q_i$)
   - Do a backward search (limited to edges of the path set) for a closest vehicle
   - Set the vehicle to start at time $t - 1$, yielding a scheduled path $p_t$
4. return $P = \{p_1, ..., p_p\}$

- **Running time:** $O(p^3)$ – determined by the Hungarian algorithm
- **The scheduling/control portion can be fully decentralized**
  - Requires only local (distance 2) communication
  - Retains the $p + \ell - 1$ completion time
  - Generally yields better than $p + \ell - 1$ completion time
- **Generalizes to arbitrary roadmaps**
Computational Performance

http://msl.cs.uiuc.edu/~jyu18/pe/formation.html - JAVA plugin required
Computational Performance

http://msl.cs.uiuc.edu/~jyu18/pe/formation.html - JAVA plugin required

~100 pebbles, ~1000 vertices, running time <0.01 second

Problems can be solved under 1 second: ~800 pebbles, ~10000 vertices
Summary

 ✓ Pebble Motion Problems

 ✓ Feasibility Test and Planning Algorithms for PMR
   ✓ The Special Case of $p = |V|$  
   ✓ Linear Time Feasibility Test

 ✓ Optimal Pebble Motion
   ✓ Computational Complexity (NP-Hardness)
   ✓ Multiflow-Based Integer Linear Programming (ILP) Algorithm

 ✓ The Case of Indistinguishable Pebbles