Lecture 14
Multi-Robot Path Planning (I)
Instructor: Jingjin Yu
Outline

Multi-robot path planning (MPP)
  ⇒ Applications
  ⇒ Formulations

Feasibility of MPP
  ⇒ The 15-puzzle variant
  ⇒ The synchronous rotation variant

Structure and complexity of optimal MPP
Some Applications

Applications

⇒ Container ports
⇒ Fulfillment centers
⇒ Delivery drones
⇒ Microfluidics (chemical/medical)
⇒ Future autonomous vehicles
⇒ Many others
Continuous Domain

Continuous formulation

⇒ Some \( n \) objects, usually close to each other (crowded)
⇒ Usually in bounded region (why?)
⇒ The objects must be moved from one configuration to another
⇒ No collision is allowed
⇒ This problem (finding any solution) is PSPACE-hard

A sequential solution
Discrete Domain: Single Move per Step

Origin: the 15-puzzle

- $4 \times 4$ game board, 15 square game pieces
- Only pieces adjacent to the empty space can move
- This is generalized first to $(N^2 - 1)$-puzzle
- Then generalized to **2-connected graph** with $n$ vertices and $n - 1$ robots
- Then generalized to general graph with $n$ vertices and up to $n - 1$ robots
- In these problems only one robot may move at a time
- Let's call these problems MPP$_S$
- Also known as the pebble motion problem ($PMG$)
For MPPs on a $n$-vertex graph, moves may be parallelized

⇒ Essentially, there are multiple “move sequences”
⇒ Each sequence requires one empty vertex in the head
⇒ It is possible that all robots move in the same step
⇒ Let’s call this MPP$_p$
Discrete Domain: Parallel Moves w/ Rotation

\( \text{MPP}_p \) still does not reflect full capabilities of modern robots

\( \Rightarrow \) Robots should not always require empty vertex to move, e.g.,

\( \Rightarrow \) We can further allow rotations in addition to parallel moves

\( \Rightarrow \) Let’s call this \( \text{MPP}_r \)

\( \Rightarrow \) Problems can be feasible even when there are no empty vertices!
Feasibility of the 15-Puzzle

The 15-puzzle (and \((N^2 - 1)\)-puzzle in general) is not always feasible.

⇒ In particular, if two robots are “swapped”, it defines an infeasible problem.

What happens if we legally swap 4 and 5?

⇒ It also (always) flip at least one other pair (in this case, 1 and 2).
For \((N^2 - 1)\)-puzzles

⇒ The only source of infeasibility comes from this
⇒ How can we detect this?
⇒ Through counting the number of moves for each robot

Now suppose we have a feasible problem, how do we solve it?

⇒ We will use the 8-puzzle as an example

⇒ Is the problem feasible?
⇒ How do we solve this problem?
Solving the 15-Puzzle (II)

First, move 1 to its goal

Then, move 2 to 3’s goal

Then, move 3 to 6’s goal, followed by moving 2,3 to their goals

First row is solved!
Solving the 15-Puzzle (III)

Then, solve the first column similarly. In this case, move 4 to 7’s goal

Then, move 7 to 8’s goal (already done), and solve 4 and 7

Then, the last piece should be readily solvable
Solving the 15-Puzzle (IV)

Procedure for solving the \((N^2 - 1)\)-puzzle

⇒ Check feasibility
⇒ Solve the first (top) row
⇒ Solve the first (left most) column, now we have a \(((N - 1)^2 - 1)\)-puzzle
⇒ Repeat the above steps until we get a 3-puzzle
⇒ The 3-puzzle should be readily solvable

The same procedure applies to \(N \times M\)-puzzles where \(N \neq M\)

How many moves are needed?

⇒ For each robot, may need to move \(O(N)\) robots
⇒ Each robot needs to be moved \(O(N)\) times
⇒ So \(O(N)\) total moves
⇒ For all \(N^2 - 1\) robots, \(O(N^4)\) total moves

The method generalizes to 2-connected graphs
How about general graphs?

We can break it into trees and 2-connected graphs.

We already can solve 2-connected graphs.

How about trees?
Partitioning Robots on Trees

We can solve the problem by partitioning robots into equivalence classes

- Two robots belong to the same equivalence classes if they can exchange locations
- It turns out this is easy to determine (well, intuitively)
- Two robots 1 and 2 that are “adjacent” (i.e., no other robots between them on the tree) cannot exchange (i.e., not equivalent) if they have the following configuration
  - If there is one more empty vertex, then 1 and 2 are equivalent
  - This can be checked (in linear time)
  - Combined with 2-connected graph solver, solves MPP$_S$ on general graph
Feasibility with Parallel Moves

\( \text{MPP}_p \) feasibility is the same as \( \text{MPP}_s \)

\( \Rightarrow \) Why?

\( \Rightarrow \) Every parallel move can be carried out as step-by-step single robot moves

\( \text{MPP}_r \), it’s slightly different

\( \Rightarrow \) We can decompose the problem into trees and 2-edge-connected components

\( \Rightarrow \) But, we need some methods for solving problems like this (\( N^2 \)-puzzle)
Solving $N^2$ puzzles

$N^2$-puzzle is similar to $(N^2 - 1)$-puzzle

⇒ We can first solve the top row (and left most column)

![Diagram showing solution steps]

⇒ This yields an $(N - 1)^2$-puzzle

⇒ We do this until we get to $3 \times 3$, and solve the top row

⇒ This leaves us with a $2 \times 3$ puzzle, which can be solved

⇒ E.g., exchange 8 and 9

![Diagram showing final solution]
MPP Problem: \((G, X_I, X_G)\), solution: collision free \(P = \{p_1, ..., p_n\}\)

Optimality objectives (minimization):

\[
\begin{align*}
\Rightarrow & \quad \text{Max time (makespan): } \min_{P \in \mathcal{P}} \max_{p_i \in P} \text{time}(p_i) \\
\Rightarrow & \quad \text{Total time: } \min_{P \in \mathcal{P}} \sum_{p_i \in P} \text{time}(p_i) \\
\Rightarrow & \quad \text{Max distance: } \min_{P \in \mathcal{P}} \max_{p_i \in P} \text{length}(p_i) \\
\Rightarrow & \quad \text{Total distance: } \min_{P \in \mathcal{P}} \sum_{p_i \in P} \text{length}(p_i)
\end{align*}
\]
### Incompatibility of the Formulations

<table>
<thead>
<tr>
<th></th>
<th>Makespan</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clockwise</td>
<td>$x + 1$</td>
<td>$2x + 3$</td>
</tr>
<tr>
<td>Counterclockwise</td>
<td>$x + 4$</td>
<td>$x + 12$</td>
</tr>
</tbody>
</table>

### Total distance

<table>
<thead>
<tr>
<th></th>
<th>Total distance</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left path only</td>
<td>$4x + 8$</td>
<td>$4x + 14$</td>
</tr>
<tr>
<td>Using right path</td>
<td>$4x + 10$</td>
<td>$4x + 13$</td>
</tr>
</tbody>
</table>

A pair of the four MPP objectives on makespan, total time, max distance, and total distance demonstrates a Pareto-optimal structure.