Lecture 15
Multi-Robot Path Planning (II)

Instructor: Jingjin Yu
Outline

NP-Hardness of MPP_r
  ⇒ NP and NP-hardness
  ⇒ Reduction from 3-SAT

Algorithms for multi-robot path and motion planning
  ⇒ Graph search based algorithm for MPP_p
  ⇒ Integer linear programming models for MPP_r
Optimal Formulations

Given \((G, X_I, X_G)\), want collision free \(P = \{p_1, \ldots, p_n\}\)

Optimality objectives (minimization):

\(\Rightarrow\) **Max time (makespan):** \(\min_{P \in \mathcal{P}} \max_{p_i \in P} \text{time}(p_i)\)

\(\Rightarrow\) **Total time:** \(\min_{P \in \mathcal{P}} \sum_{p_i \in P} \text{time}(p_i)\)

\(\Rightarrow\) **Max distance:** \(\min_{P \in \mathcal{P}} \max_{p_i \in P} \text{length}(p_i)\)

\(\Rightarrow\) **Total distance:** \(\min_{P \in \mathcal{P}} \sum_{p_i \in P} \text{length}(p_i)\)
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>

<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.
NP and NP-Hardness

Note that we are classifying **problems** here! In particular, we are NOT classifying **algorithms**.

An **problem** is in the complexity class nondeterministic polynomial time (NP) if

⇒ It can be solved by a **non-deterministic Turing machine** in polynomial time
⇒ Equivalently, a given solution can be verified in polynomial time
⇒ E.g., graph search
⇒ E.g., given a graph $G$, find a Hamiltonian cycle
  ⇒ To see that it is in NP, given a cycle, verifying it is part of $G$ is doable in polynomial time

A problem $P_1$ is NP-hard if

⇒ Solving it is harder than solving any other problems in NP
⇒ I.e., any problem $P_2 \in \text{NP}$ can be solved in polynomial time via solving $P_1$
⇒ Note that a problem is NP-hard does not require it is in NP
⇒ A problem that is NP-hard and also in NP is NP-complete
⇒ This implies that all NP-complete problems are in a sense equal in hardness
Some Classical NP-Complete Problems

**Boolean satisfiability (SAT):** first problem proven to be NP-hard
- $n$ binary variables $x_1, ..., x_n$
- A literal $y$ of a variable $x$ is $x$ or $\neg x$, total $2n$ of these
- $m$ disjunctive clauses of literals, i.e. $c_j = y_{j_1} \lor \cdots \lor y_{j_k}$
- Question: are there values for $x_i, ..., x_n$ so that $c_1 \land \cdots \land c_m = 1$?
- Shown to be NP-complete (Cook-Levin theorem)
  - SAT is in NP because checking an answer is doable in polynomial time
  - NP-hard via direct reduction from a generic nondeterministic Turing machine

**3SAT:** SAT with each clause containing up to 3 literals
- NP-hard via the reduction from SAT
- This is how NP-hardness is proven in general

**Vertex cover:** $G = (V, E)$, is there a set of $K$ vertices that covers $V$?
- Reduction from 3SAT

Numerous others: Hamiltonian cycle, Traveling Salesperson Problem (TSP), Set Cover, Knapsack, ...
NP-Hardness of Makespan Optimal $\text{MPP}_r$

Min Makespan $\text{MPP}_r$ is NP-hard

$3\text{SAT}$ $(x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor \neg x_4) \land (\neg x_2 \lor x_3 \lor x_4)$

$n = 4$ variables
$m = 3$ clauses

Min Makespan MPP
NP-Hardness of Distance Optimal $\text{MPP}_r$

NP-hardness of distance optimal MPP is slightly more tricky...

**Theorem.** MPP is NP-hard when optimizing min makespan, min total time, min max distance, and min total distance.
Implications of MPP Intractability

A problem being NP-hard means likely no polynomial time algorithm exists for solving it exactly

⇒ More precisely, unless P = NP (a million dollar question)

Implications:

⇒ Practitioners should not try to find polynomial time algorithm
⇒ Aiming for solving the problem approximately
⇒ Or aiming for solving easier cases of the problem quickly
⇒ This is what we will try with MPP

Algorithmic solutions for MPP fall into two flavors

⇒ Discrete search based algorithm for
⇒ Integer programming solvers
Discrete Search Algorithms (I)

There are many variations of search based solutions for solving MPP

⇒ One can solve the problem as coupled problem
⇒ For $n$ robots, this means state space has size $|G^n|$ - very impractical!
⇒ Generally, decoupled methods are used
⇒ That is, treat robots individually as much as possible
⇒ Recall the basic structure of search algorithms (still applies!)

```plaintext
input: $G = (V, E), x_I, x_G$
AddToQueue($x_I, Queue$);  // Add $x_I$ to a queue of nodes to be expanded
while (!IsEmpty(Queue))
    $x \leftarrow$ Front(Queue);  // Retrieve the front of the queue
    if ($x.expanded == true$) continue;  // Do not expand a node twice
    $x.expanded = true$;  // Mark $x$ as expanded
    if ($x == x_G$) return solution;  // Return if goal is reached
    for each neighbor $n_i$ of $x$  // Add all neighbors of to the queue
        if ($n_i.expanded == false$) AddToQueue($n_i, Queue$)
return failure;
```
Discrete Search Algorithms (II)

So, we play with the priority queue

⇒ To start, plan paths for each robot individually, ignoring possible collision
⇒ “Execute” the individual paths until collision occurs
⇒ At collision, the algorithm branches
⇒ As this happens, all branched nodes are added to the queue
⇒ E.g., two robots passing through a “corridor”
  ⇒ Assume we are minimizing the total time
  ⇒ First, two individual paths are computed
  ⇒ This is the initial node in the queue
  ⇒ After popping the node and executing it for one step, a conflict arises
  ⇒ This cause two new nodes to be generated
    ⇒ One node letting 1 move to the conflicting position, 2 stays
    ⇒ One node letting 2 move to the conflicting position, 1 stays
    ⇒ Both increases the initial cost by 1
⇒ These new nodes are then processed until a solution is found
Different objectives cause the queue to be sorted differently

\[ \Rightarrow \text{Total distance} \]

\[ \Rightarrow \text{Initially all choose left path} \]
\[ \Rightarrow \text{Then 1-4 have conflict at } t = 1, \text{ generating 4 new nodes (robot } i \text{ goes first)} \]
\[ \Rightarrow \text{Then at } t = 2, \text{ suppose we pick the node letting 1 go first, three new nodes are created} \]
\[ \Rightarrow \text{These three new nodes can be inserted into the front of the queue using a secondary heuristic} \]
\[ \Rightarrow \text{After one more iteration, 2 new nodes are generated} \]
\[ \Rightarrow \text{Then one last iteration resolves all conflicts} \]
\[ \Rightarrow \text{The total distance remains the same for all nodes, which is } 4x + 8 \]

\[ \Rightarrow \text{Total time} \]

\[ \Rightarrow \text{Initial node cost is } 4x + 8 \]
\[ \Rightarrow \text{Here, at } t = 1, \text{ 4 new nodes, cost is now } 4x + 11 \text{ for all} \]
\[ \Rightarrow \text{At } t = 2, \text{ if 4 goes through the right, cost is } 4x + 13, \text{ otherwise, } 4x + 14 \]
Strengths and Weakness of Discrete Search

Strengths of discrete search solutions

⇒ When it works, the algorithm runs rather fast
  ⇒ Because the overall algorithm is relatively simple due to its discrete nature
⇒ Capable of solving large problems
⇒ Generally straightforward to implement and tweak

Weaknesses

⇒ As the interactions among the robots grow, performance degrades quickly
⇒ As such, not suitable for solving very dense problems
⇒ Not suitable for handling MPPr as the number of possible rotations can be very large; huge branching factor
  ⇒ For 16-puzzle, >1000 possible cycles
  ⇒ Each cycle has two directions
  ⇒ Enumerating becomes impossible
An Integer Programming Based Solver

Key idea: time expansion

**Theorem.** Fixing a natural number $T$, an MPP 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.
The Constraints

Meet-on-vertex

\[ u \quad v \quad w \]

\[ \sum_{1 \leq i \leq n} (x_{uv,t,t+1}^i + x_{vv,t,t+1}^i + x_{wv,t,t+1}^i) \leq 1 \]

Meet-on-edge

\[ u \quad v \]

\[ \sum_{1 \leq i \leq n} (x_{uv,t,t+1}^i + x_{vu,t,t+1}^i) \leq 1 \]
Algorithm for Min Makespan

Pick an initial $T$

Build the time-expanded network

Set up an ILP model

No $T = T + 1$

Feasible?

Yes Return the path set

Additional heuristics

$\Rightarrow$ Reachability analysis

$\Rightarrow$ Divide and conquer

Other objectives (total time, max distance, total distance)

\[
\begin{align*}
\max \sum_{i=1}^{n} x_{i,i}, \text{subject to} & \quad \forall e_j, \sum_{i=1}^{n} x_{i,j} \leq 1 \\
& \forall v, \sum_{e_j \in \delta^+(v)} x_{i,j} = \sum_{e_j \in \delta^-(v)} x_{i,j}
\end{align*}
\]
The Approach Solves Some Tough Problems...

A 7-step min makespan plan
Generalization to Continuous Domain

Lattice overlay

Restore connectivity

Snapping

Trajectory planning

Path smoothing