Progress on Multi-Agent Path Finding in Real-World Scenarios

Cainiao Network
December 20, 2017

N. Ayanian, L. Cohen, W. Hoenig, S. Koenig, S. Kumar, J. Li, Hang Ma, T. Uras, H. Xu
University of Southern California
C. Tovey
Georgia Institute of Technology
G. Sharon
University of Texas at Austin

Multi-Agent Path Finding (MAPF)

• Multi-agent path finding (MAPF)
  • Given: a number of agents (each with a start and goal location) and a known environment
  • Task: find collision-free paths for the agents from their start to their goal locations that minimize some objective
• Objectives
  • Makespan: latest arrival time of an agent at its goal location
  • Flowtime: sum of the arrival times of all agents at their goal locations

Multi-Agent Path Finding (MAPF)
Multi-Agent Path Finding (MAPF)

4-neighbor grid
Multi-Agent Path Finding (MAPF)

- Application: Amazon fulfillment centers

Hang Ma (hangma@usc.edu)

Multi-Agent Path Finding (MAPF)
- Application: autonomous tug robots (joint work with NASA Ames)
- Reduce pollution
- Reduce energy consumption
- Reduce human danger
- Reduce human workload
- Reduce airport size

Hang Ma (hangma@usc.edu)

Multi-Agent Path Finding (MAPF)
- Application: automated ports

Hang Ma (hangma@usc.edu)

Multi-Agent Path Finding (MAPF)
- Each agent moves N, E, S or W into an adjacent unblocked cell
- Not allowed ("vertex collision")
  - Agent 1 moves from X to Y
  - Agent 2 moves from Z to Y
- Not allowed ("edge collision")
  - Agent 1 moves from X to Y
  - Agent 2 moves from Y to X
- Allowed

Hang Ma (hangma@usc.edu)
Multi-Agent Path Finding (MAPF)

- Optimal MAPF algorithms
  - Theorem [Yu and LaValle]: MAPF is NP-hard to solve optimally for makespan or flowtime minimization.
- Bounded-suboptimal MAPF algorithms
  - Theorem: MAPF is NP-hard to approximate within any factor less than 4/3 for makespan minimization on graphs in general.

Multi-Agent Path Finding (MAPF)

- Reduction from $(\leq 3, =3)$-SAT: It is NP-complete to determine whether a given $(\leq 3, =3)$-SAT instance is satisfiable.
- Each clause contains at most 3 literals.
- Each variable appears exactly 3 clauses.
- Each variable appears uncomplemented at least once.
- Each variable appears complemented at least once.
- Example: $((x_1 \lor x_2 \lor \neg x_4) \land (x_1' \lor x_2' \lor \neg x_3') \land (x_1' \lor x_2' \lor \neg x_4'))$

Conflict-Based Search with Highways

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Bounded-suboptimal MAPF solver that plans for each agent independently.
- Experience graphs [Phillips, Cohen, Chitta and Likhachev]: Bounded-suboptimal single-agent path planner so that the resulting path uses edges in a given subgraph (the experience graph) as much as possible.
Conflict-Based Search with Highways

- Graph for an A* search
  - Regular (no highways)
  - Highways K1 (experience graphs)
- Graph relaxation for calculating the heuristics of an A* search
  - All costs are 1 except for the ones shown

Suboptimality bound 4

All costs are 1

Optimality 1

Features:
- Collisions?
- Direction of direction vector (N, E, S, W)
- Magnitude of direction vector
- > 0.5?

Hang Ma (hangma@usc.edu)
Conflict-Based Search with Highways

- Graphical models basically encode probabilistic knowledge
  - If agents collide in a cell, make it more likely that there is a highway in that cell
  - If most agents move northward in a cell, make it more likely that a highway in that cell, if any, is a northward one
  - If a northward highway is in a cell, make it more likely that highways in its northern and southern neighbors, if any, are also northward ones (to form a longer lane)
  - If a northward highway is in a cell, make it more likely that highways in its western and eastern neighbors, if any, are southward ones (to form adjacent lanes in opposite directions)

- Rapid random restarts help to solve more multi-agent path finding problems within a given runtime limit.
  - Here: We randomize the ordering in which the agents plan their paths in the high-level root node.

<table>
<thead>
<tr>
<th>Runs</th>
<th>Time Limit</th>
<th>50 “easy”</th>
<th>50 “hard”</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>300 sec</td>
<td>100.00%</td>
<td>0.00%</td>
<td>76.00%</td>
</tr>
<tr>
<td>3</td>
<td>100 sec</td>
<td>97.65%</td>
<td>96.87%</td>
<td>97.60%</td>
</tr>
<tr>
<td>5</td>
<td>60 sec</td>
<td>98.57%</td>
<td>98.81%</td>
<td>98.70%</td>
</tr>
</tbody>
</table>
Target Assignment and Path Finding (TAPF)

- **Non-anonymous MAPF**
  - NP-hard
  - solved with A* approaches
  - e.g. conflict-based search or M*

- **Anonymous MAPF**
  - polynomial-time solvable for makespan minimization
  - solved with flow approaches
  - e.g. max-flow algorithm

(Non-)Anonymous MAPF

- **Anonymous MAPF**
  - Given: a number of agents (each with a start and goal location) and a known environment
  - Task: find collision-free paths for the agents from their start to their goal locations that minimize makespan or flowtime

- **(Non-anonymous) MAPF**
  - Given: a number of agents (each with a start location), an equal number of goal locations, and a known environment
  - Task: assign a different goal location to each agent and then find collision-free paths for the agents from their start to their goal locations that minimize makespan or flowtime

Anonymous MAPF

- Theorem [Yu and Lavalle]: An anonymous MAPF instance admits a MAPF plan with makespan at most $T$ if and only if the time-expanded network with $T$ periods admits a max flow of the number of agents.

Anonymous MAPF

- Each agent moves N, E, S or W into an adjacent unblocked cell
- Not allowed (“vertex collision”)
  - Agent 1 moves from X to Y
  - Agent 2 moves from Z to Y
- Not allowed (“edge collision”)
  - Agent 1 moves from X to Y
  - Agent 2 moves from Y to X

Anonymous MAPF

- Each agent moves N, E, S or W into an adjacent unblocked cell
- All edges have capacity one
- Not allowed (“vertex collision”)
  - Agent 1 moves from X to Y
  - Agent 2 moves from Z to Y
- Not allowed (“edge collision”)
  - Agent 1 moves from X to Y
  - Agent 2 moves from Y to X
Target Assignment and Path Finding (TAPF)

\[ \text{TAPF} = \text{mix of non-anonymous and anonymous MAPF} \]

TAPF with \( k \) teams (here: \( k = 3 \)), also called types or groups

• Theorem: TAPF (with \( k \geq 1 \) teams) is NP-hard to solve optimally for makespan or flowtime minimization

• Theorem: TAPF (with \( k \geq 1 \) teams) is NP-hard to approximate within any factor less than \( 4/3 \) for makespan minimization on graphs in general

Target Assignment and Path Finding (TAPF)

• Task: find the target assignments and collision-free paths that minimize the makespan.

• How to solve? Ideas from:
  • Conflict-based search for solving non-anonymous MAPF (NP-hard)
  • Max-flow algorithm for solving anonymous MAPF (P)

⇒ Our algorithm:
  • Conflict-Based Min-Cost Flow (CBM) = Conflict-Based Search (CBS) + (min-cost) max flow
Target Assignment and Path Finding (TAPF)

- CBS for Non-Anonymous MAPF

  - CBS:
    1. Find paths for each single agent separately
    2. Look for collisions in paths
    3. If there is a collision between \( a_i \) and \( a_j \):
       - Option 1 or Option 2 to avoid collision
         - Collision: \(<\text{agent}\_i,\text{agent}\_j,\text{location}\_x,\text{time}\_t>\>
         - Constraint: \(<\text{agent},\text{location},\text{time}>\>
       - Option 2: \( a_i \) cannot stay in \( x \) at time \( t \)

Target Assignment and Path Finding (TAPF)

- CBM: considers each team to be a meta-agent / a best-search on a search tree

  - For every tree node:
    1. Find paths for a single group separately
    2. Look for collisions in paths
    3. If there is a collision between team\(1\) and team\(2\):
       - Option 1 or Option 2 to avoid collision
         - Collision: \(<\text{team}\_\text{team1},\text{team}\_\text{team2},\text{location}\_x,\text{time}\_t>\>
         - Constraint: \(<\text{team},\text{location},\text{time}>\>
       - Option 1: agents in team\(1\) cannot stay in \( x \) at time \( t \)
       - Option 2: agents in team\(2\) cannot stay in \( x \) at time \( t \)

---

Target Assignment and Path Finding (TAPF)

- An example

![Diagram showing target assignment and path finding](image)

Target Assignment and Path Finding (TAPF)

- Finding paths for single teams separately

![Diagram showing target assignment and path finding](image)

Target Assignment and Path Finding (TAPF)

- Store paths and key

![Diagram showing target assignment and path finding](image)

Target Assignment and Path Finding (TAPF)

- Look for collisions in paths

![Diagram showing target assignment and path finding](image)
**Target Assignment and Path Finding (TAPF)**

- **Store colliding teams**

Root key = 2
Colliding Teams (team1, team2)

- **Pop a tree node**

Root key = 2
Colliding Teams (team1, team2)
Earliest Collision (team1, team2, d,1)

- **Two options**

Root key = 2
Colliding Teams (team1, team2)
Earliest Collision (team1, team2, d,1)

- **Look for collisions in paths**

Constraints (team1, d,1)
**Target Assignment and Path Finding (TAPF)**

- Store colliding teams

**Root key = 2**
- Colliding Teams (team1, team2)
- Earliest Collision (team1, team2, d, 1)

**Root key = 3**
- Colliding Teams (team1, team2)

**Team1** key = 3
- Colliding Teams (team1, team2)

**Team2** key = 3
- Colliding Teams (team1, team2)

**None**

**BINGO!**

Hang Ma (hangma@usc.edu)
Target Assignment and Path Finding (TAPF)

- Edge weights — reducing possible collisions
- Idea: choose paths that have fewest collisions with other teams, when finding paths for a single team
- Take into account the paths of other teams
- Bias the search using a min-cost max-flow algorithm that finds a max flow with minimal total edge weights

Hang Ma (hangma@usc.edu)

Target Assignment and Path Finding (TAPF)

- Edge Weights are Crucial

Setups:
- 30×30 4-neighbor grids with 10% randomly blocked cells.
- 5 agents per team.
- 5-minute time limits.

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Time (s)</th>
<th>Success</th>
<th>Cost (s)</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.34</td>
<td>1</td>
<td>0.41</td>
<td>0.72</td>
</tr>
<tr>
<td>15</td>
<td>0.57</td>
<td>1</td>
<td>1.06</td>
<td>0.44</td>
</tr>
<tr>
<td>20</td>
<td>0.78</td>
<td>1</td>
<td>2.06</td>
<td>0.44</td>
</tr>
<tr>
<td>25</td>
<td>1.07</td>
<td>1</td>
<td>2.06</td>
<td>0.58</td>
</tr>
<tr>
<td>30</td>
<td>1.35</td>
<td>1</td>
<td>3.32</td>
<td>0.22</td>
</tr>
<tr>
<td>35</td>
<td>1.60</td>
<td>1</td>
<td>4.60</td>
<td>0.22</td>
</tr>
<tr>
<td>40</td>
<td>1.85</td>
<td>1</td>
<td>5.95</td>
<td>0.22</td>
</tr>
<tr>
<td>45</td>
<td>2.10</td>
<td>1</td>
<td>7.25</td>
<td>0.22</td>
</tr>
<tr>
<td>50</td>
<td>2.35</td>
<td>1</td>
<td>8.55</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Hang Ma (hangma@usc.edu)

Target Assignment and Path Finding (TAPF)

- Theorem: CBM is complete and optimal for minimizing makespan for TAPF instances

Hang Ma (hangma@usc.edu)

Target Assignment and Path Finding (TAPF)

- Comparisons

Setups:
- 30×30 4-neighbor grids with 10% randomly blocked cells.
- 5-minute time limits.
- CBM: specialized solver
- Versus
- ILP (Integer Linear Program): useful tool and easy to model

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Time (s)</th>
<th>Success</th>
<th>Cost (s)</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.34</td>
<td>1</td>
<td>18.24</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>0.57</td>
<td>1</td>
<td>35.44</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0.78</td>
<td>1</td>
<td>62.85</td>
<td>0.94</td>
</tr>
<tr>
<td>25</td>
<td>1.07</td>
<td>1</td>
<td>88.55</td>
<td>0.82</td>
</tr>
<tr>
<td>30</td>
<td>1.35</td>
<td>1</td>
<td>108.75</td>
<td>0.66</td>
</tr>
<tr>
<td>35</td>
<td>1.60</td>
<td>1</td>
<td>121.99</td>
<td>0.46</td>
</tr>
<tr>
<td>40</td>
<td>1.85</td>
<td>1</td>
<td>152.98</td>
<td>0.14</td>
</tr>
<tr>
<td>45</td>
<td>2.10</td>
<td>1</td>
<td>161.52</td>
<td>0.14</td>
</tr>
<tr>
<td>50</td>
<td>2.35</td>
<td>1</td>
<td>161.95</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Hang Ma (hangma@usc.edu)

Target Assignment and Path Finding (TAPF)

- Spectrum: Anonymous — Non-Anonymous

Fixed 100 agents in total, 2 to 50 teams

<table>
<thead>
<tr>
<th>Number of Teams</th>
<th>Average Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.85</td>
</tr>
<tr>
<td>5</td>
<td>0.95</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
</tr>
<tr>
<td>20</td>
<td>0.99</td>
</tr>
<tr>
<td>50</td>
<td>0.99</td>
</tr>
</tbody>
</table>

[2 teams, 50 agents per team] — [50 teams, 2 agents per team]
Anonymous — Non-Anonymous
P — NP-hard

Hang Ma (hangma@usc.edu)
The package exchange robot routing problem (PERR) involves:

- Each agent carries exactly one package.
- Each package needs to be delivered to a given goal location.
- Two agents in adjacent locations can exchange packages.

**Example:**

1. **K-type PERR**
   - Each agent moves N, E, S or W into an adjacent unblocked cell.
   - Not allowed ("vertex collision"): Agent 1 moves from X to Y, Agent 2 moves from Z to Y.
   - Not allowed ("edge collision"): Agent 1 moves from X to Y, Agent 2 moves from Y to X.

**Theorem:**

- PERR instances can be solved with versions of conflict-based search and multi-commodity flow algorithms.
Execution of MAPF Plans

- Planning uses models that are not completely accurate
- Robots do not move exactly at the nominal speed
- Robots have unmodeled kinematic constraints
- Plan execution will therefore likely deviate from the plan
- Replanning whenever plan execution deviates from the plan is intractable since it is NP-hard to find good plans

Execution of MAPF Plans

- MAPF-POST makes use of a simple temporal network to post-process the output of a multi-agent path finding solver in polynomial time to allow for plan execution on robots
- Takes into account edge lengths
- Takes into account velocity limits (for both robots and edges)
- Guarantees a safety distance among robots
- Avoids replanning in many cases
Execution of MAPF Plans

• Minimize makespan and flowtime
  • Schedule each arrival in a location as early as allowed by the constraints

\[ \text{Minimize } \sum_{k=1}^{K} t^{k}(e) \text{ subject to:} \]
\[ t^{k}(e) = 0 \text{ and } \forall e = (v, v') \in E, \]
\[ t^{k}(e) - t^{k}(e') \geq L(e'), \]
\[ t^{k}(e) - t^{k}(e') \leq U(e) \]

Maximize safety distance

• Assume that each agent moves with a constant velocity of at least \( v_{min} \) along every Type 1 edge.
• Then, the safety distance is \( 2S_{min}/L_{max} \)

Execution of MAPF Plans

• Main loop
  • Run Conflict-Based Search with Highways to find a MAPF plan (slow)
  • Construct a simple temporal network for the MAPF plan
  • Determine the earliest arrival times in the nodes

• Calculate speeds for the robots from the earliest arrival times
Execution of MAPF Plans

- Main loop
  - Run Conflict-Based Search with Highways to find a MAPF plan (slow)
  - Construct a simple temporal network for the MAPF plan
  - Determine the earliest arrival times in the nodes
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths in the MAPF plan with these speeds

- If plan execution deviates from the plan, then

Execution of MAPF Plans

- Main loop
  - Run Conflict-Based Search with Highways to find a MAPF plan (slow)
  - Construct a simple temporal network for the MAPF plan
  - Determine the earliest arrival times in the nodes
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths in the MAPF plan with these speeds

- If plan execution deviates from the plan, then

Execution of MAPF Plans

- Main loop
  - Run Conflict-Based Search with Highways to find a MAPF plan (slow)
  - Construct a simple temporal network for the MAPF plan
  - Determine the earliest arrival times in the nodes
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths in the MAPF plan with these speeds

- If plan execution deviates from the plan, then

Execution of MAPF Plans

- Main loop
  - Run Conflict-Based Search with Highways to find a MAPF plan (slow)
  - Construct a simple temporal network for the MAPF plan
  - Determine the earliest arrival times in the nodes
  - Calculate speeds for the robots from the earliest arrival times
  - Move robots along their paths in the MAPF plan with these speeds

- If plan execution deviates from the plan, then

- MAPF solver: ECBS+HWY
- MAPF POST: C++, boost graph library, Gurobi LP solver
- PC: i7-4600U 2.1 GHz, 12 GB RAM
- Terrain: 4x3 gridworld with 1m² cells and δ = 0.4m
- Architecture: ROS with decentralized execution
  - Robot controller with state (x, y, θ) (attempts to meet deadline)
  - PID controller (correction for heading error and drift)
- Robot simulator: V-REP
- Robots: iRobot Create2 robots
- Test environment: VICON MX Motion Capture System
Execution of MAPF Plans

4-neighbor grid

Execution of MAPF Plans

4-neighbor grid

MAPF with Delay Probability

• Idea: addressing delays with planning rather than execution monitoring
• Formulation: Multi-Agent Path Finding with Delay Probabilities (MAPF-DP):
  • A generalization of multi-agent path finding (MAPF)
  • Takes into account the uncertainty of delay during execution
  • Every agent suffers from a delay probability:
    • It stays in its current location with the probability when executing a move action

MAPF with Delay Probability

• Tasks of MAPF-DP:
  • Planning: compute plans — one path for each agent
  • Execution: use execution policies — GO or STOP commands to control how the agents proceed along their paths
  • Objective: find a combination of a plan and an execution policy with small average makespan during plan execution
• Our Approach:
  • Valid plans and robustness => deadlock-free and collision-free execution
  • Two classes of decentralized robust plan-execution policies
  • A 2-level hierarchical algorithm for generating valid plans

Multi-Agent Pickup and Delivery (MAPD)

• Existing research on MAPF — a “one-shot” version:
  • One pre-determined task for each agent — navigates to its goal location
• MAPD — a “lifelong” version of MAPF:
  • A task can enter the system at any time
  • Agents have to constantly attend to a stream of new tasks

Multi-Agent Pickup and Delivery (MAPD)

• MAPD Algorithms
  1. Decoupled Task Assignment and Path Finding
    • Token Passing (TP): greedy task assignment and no task reassignment
    • Token Passing with Task Swaps (TPTS): local task reassignment between two agents
  2. Centralized Task Assignment and Path Finding CENTRAL
    • Roughly:
      • Effectiveness: TP < TPTS < CENTRAL
      • Efficiency: CENTRAL < TPTS < TP
Multi-Agent Pickup and Delivery (MADP)

- Tasks

MAPD: Executing Task

- In order to execute a task, the agent has to move from its current location via the pickup location to the delivery location:
  1. When the agent reaches the pickup location, it starts to execute the task.
  2. When it reaches the delivery location, it finishes the task.

MAPD: Free Agents

- A free agent can be assigned to any unexecuted task

MAPD: Occupied Agents

- An occupied agent has to finish executing its current task.

MAPD: Assignment of Agents to Tasks
MAPD: Objective
• Finish executing each task as quickly as possible.

MAPD: Effectiveness of a MAPD algorithm
• Service time: the average number of timesteps needed to finish executing each task after it enters the system.
• An algorithm solves a MAPD instance $\iff$ Service time of all tasks is bounded.

MAPD: Service time is $\frac{7+7}{2}=7$

MAPD: Solvability
• Not every MAPD instance is solvable

MAPD: Well-Formed MAPD Instances
• Being well-formed (based on [M. Cápa, Volkvek and Kleiner]): a sufficient condition that makes MAPD instances solvable
• Intuition: agents should only be allowed to rest (that is, stay forever) in locations, called parking locations, where they cannot block other agents

MAPD: Parking Locations
Task Parking Locations: all pickup and delivery locations of tasks (storage locations, inventory stations, etc.)
Non-task Parking Locations:
• All initial locations of agents
• Additional designated parking locations
MAPD: Well-Formed MAPD Instances
1. The number of tasks is finite;
2. The number of non-task parking locations is greater than or equal to the number of agents;
3. For any two parking locations, there exists a path between them that traverses no other parking locations.

MAPD: MAPD Algorithms
We present:
1. Two Decoupled Algorithms: complete for well-formed MAPD instances (solve all well-formed instances)
   - Token Passing (TP)
   - Token Passing with Task Swaps (TPTS)
2. One Centralized Algorithm:
   - CENTRAL

MAPD: A Running Example
Unexecuted Tasks: task₁, task₂,
Agent a₁ and agent a₂ are resting
Agent a₁ is assigned to task₁ and on the way to the pickup location s₁.

MAPD: Token Passing (TP)
Based on an idea similar to Cooperative A* [Silver]:
• Token: a synchronized shared block of memory that contains the current paths of all agents, set of unexecuted task, and agent assignments
• Only one agent has access to the token at each time
• Each agent assigns itself a task, plans its path, and passes the token to the next agent

Key idea of TP:
• A task can only be assigned once
• Once an agent is assigned to a task, it cannot be assigned to other tasks until it finishes the task

MAPD: TP: Running Example
Task Available for Assignment: task₂
Agent a₁ and agent a₂ request for token

MAPD: TP: Agent a₁’s Turn
Agent a₁ Has Token
1. It cannot assign itself to any task because agent a₂ rests in g₁, the only task available to it
2. It has to rest in a parking location that will not create any deadlock
3. It can continue to rest in its current location
Agent $a_2$’s Turn

1. It assigns itself to task $s_2$.
2. Task $s_2$ is no longer available to other agents.
3. It plans a cost-minimal collision-free path to execute task $s_2$. 

Hang Ma (hangma@usc.edu)
MAPD: TP: Completeness
- Theorem: All well-formed MAPD instances are solvable, and TP solves them

MAPD: Improving the Effectiveness of TP
- TP is simple but can be made more effective: A task with an assigned agent can be assigned a new agent (as long as the task has not been executed)
MAPD: Token Passing with Task Swaps (TPTS)

- TPTS: An agent is allowed to grab a task from another agent if it can finish the task earlier.

MAPD: TPTS: Running Example

Tasks Available for Assignment: task1, task2
Agent a1 and agent a2 request for token.

MAPD: TPTS: Agent a1’s Turn

Agent a1 has token
Agent a1 grabs task1 from agent a3

MAPD: TPTS: Agent a3 Making Decisions

Agent a3 has token
Agent a3 moves to a parking location that will not create any deadlock in the future.

MAPD: TPTS: Agent a2’s Turn

Agent a2 has token
Agent a2 grabs task1 from agent a1

MAPD: TPTS: Agent a1 Making Decisions

Agent a1 has token
Agent a1 assigns itself to task2.
MAPD: TPTS: Completeness
- Theorem: TPTS solves all well-formed MAPD instances

MAPD: Centralized MAPD Algorithm
- CENTRAL assigns agents to tasks in a centralized way:
  1. assigns parking locations to all free agents using Hungarian method
  2. plans paths for all of them from their current locations to their assigned parking locations by solving the resulting "one-shot" multi-agent path-finding problem

MAPD: CENTRAL: Running Example
- Tasks available for assignment: task1, task2

MAPD: CENTRAL: Candidate Parking Locations
- Pickup locations s1 and s2 + three additional "good" parking locations, one for each agent:

MAPD: CENTRAL: Assignment
- CENTRAL uses Hungarian method to find a cost-minimal assignment from parking locations to agents (pickup locations have priority over other parking locations):

MAPD: CENTRAL: Path Finding
- CENTRAL plans collision-free paths for all agents from their current locations to their assigned parking locations
- CENTRAL plans paths to delivery locations only when agents reach pickup locations
MAPD: Comparisons of Three Algorithms

MAPD: Experiments Setup
- Small Simulated Warehouse Environment: 21 × 35 4-neighbor grid with 50 agents
  - Gray cells are inventory stations and storage locations
  - Colored circles are the initial locations of agents

MAPD: Experiments Setup
- Large Simulated Warehouse Environment: 81 × 81 4-neighbor grid with 500 agents

MAPD: Experimental Results
- 500 Random Tasks, 10 to 50 Agents
  - Effectiveness:
    1. Service Time:
       CENTRAL < TPTS < TP
    2. Throughput – # tasks executed per 100 timesteps:
       TP < TPTS < CENTRAL
    3. Makespan – timestep when all tasks are finished:
       CENTRAL < TPTS < TP
  - Runtime per Timestep:
    TP < 10 milliseconds
    TPTS < 200 milliseconds
    CENTRAL < 4,000 milliseconds

MAPD: Experimental Results
- Results for TP: 1000 Random Tasks, 100 to 500 Agents
  - 100 agents: ~ 0.09 seconds per timestep
  - 500 agents: ~ 6 seconds per timestep

MAPD: Takeaways
- MAPD: A "lifelong" version of multi-agent path finding
- Three Algorithms:
  - Decoupled and complete for well-formed MAPD instances: TP, TPTS
  - Centralized: CENTRAL
- Task Assignment Effort: TP < TPTS < CENTRAL
- Effectiveness: TP < TPTS < CENTRAL
- Efficiency: CENTRAL < TPTS < TP
MAPD in Continuous Time

- In submission to ICAPS-18
- Take kinematic constraints of robots into account directly during planning
- Compute kinematically feasible paths that
  1. Work on non-holonomic robots
  2. Take their maximum translational and rotational velocities into account
  3. Provide a guaranteed safety distance between them

References

Conclusions

• This research is joint work with N. Ayanian, L. Cohen, W. Hö nig, S. Koenig, S. Kumar, J. Li, G. Sharon, C. Tovey, T. Uras and H. Xu
• Thank you for listening!
• My research is funded in part by a USC Annenberg Fellowship
• Funded in part by ARO, NASA, NSF and ONR

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

• For more information, see www-scf.usc.edu/~hangma/
or send me an email: hangma@usc.edu
• Our lab: idm-lab.org