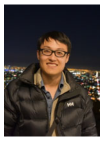
Progress on Multi-Agent Path Finding in Real-World Scenarios

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

USC Viterbi School of Engineering



www-scf.usc.edu/~hangma/
hangma@usc.edu

IDM Lab

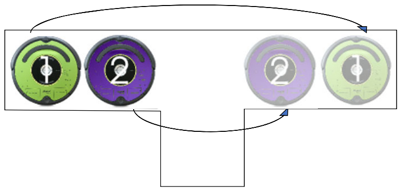
Liron Cohen | Sven Koenig | T. K. Satish Kumar | Jiaoyang Li | Tansel Uras | Hong Xu

Link to IDM Lab:
idm-lab.org
Project on "multi-agent path planning":
idm-lab.org/project-p.html

Multi-Agent Path Finding (MAPF)

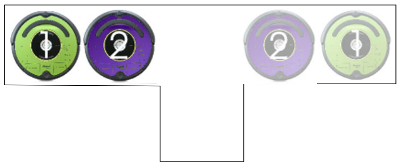
- Multi-agent path finding (MAPF)
  - Given: a number of agents (each with a start and goal location) and a known environment
  - Task: find collision-free paths for the agents from their start to their goal locations that minimize some objective
- Objectives
  - Makespan: latest arrival time of an agent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

Multi-Agent Path Finding (MAPF)



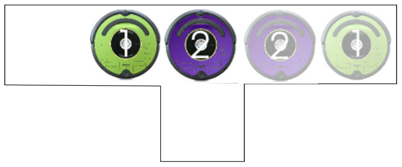
4-neighbor grid

Multi-Agent Path Finding (MAPF)



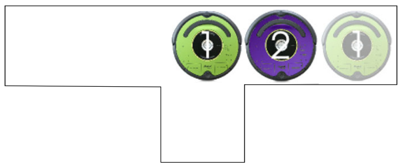
4-neighbor grid

Multi-Agent Path Finding (MAPF)



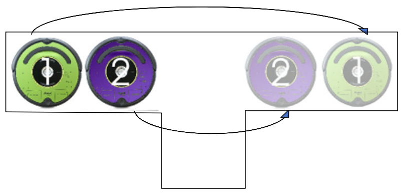
4-neighbor grid

Multi-Agent Path Finding (MAPF)



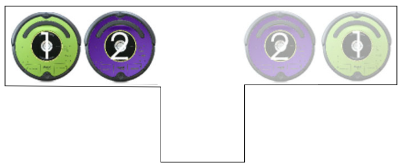
4-neighbor grid

Multi-Agent Path Finding (MAPF)



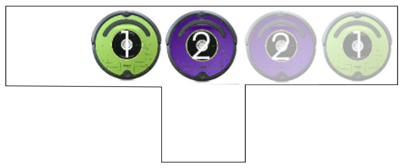
4-neighbor grid

Multi-Agent Path Finding (MAPF)



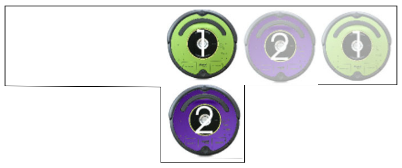
4-neighbor grid

Multi-Agent Path Finding (MAPF)



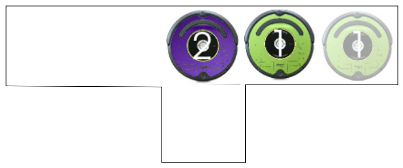
4-neighbor grid

Multi-Agent Path Finding (MAPF)



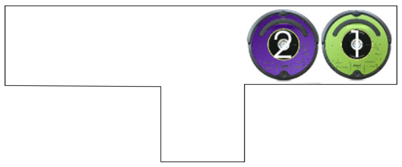
4-neighbor grid

Multi-Agent Path Finding (MAPF)



4-neighbor grid

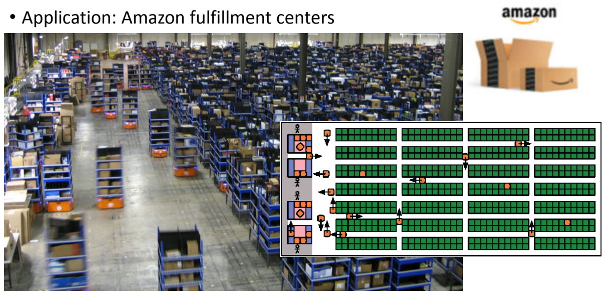
Multi-Agent Path Finding (MAPF)



4-neighbor grid

Multi-Agent Path Finding (MAPF)

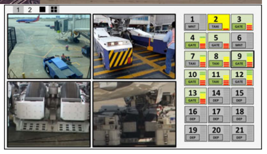
- Application: Amazon fulfillment centers



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



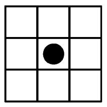
Multi-Agent Path Finding (MAPF)

- Application: automated ports

Multi-Agent Path Finding (MAPF)

Robot

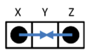
Agent

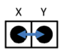


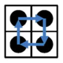
- Simplifying assumptions
  - Point robots
  - No kinematic constraints
  - Discretized environment
    - we use grids here but most techniques work on planar graphs in general

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







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

[www.random-ideas.net]

Multi-Agent Path Finding (MAPF)

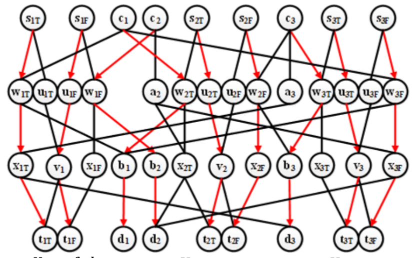
- Reduction from (≤3, =3)-SAT: It is NP-complete to determine whether a given (≤3, =3)-SAT instance is satisfiable

- Each clause contains at most 3 literals
- Each variable appears in exactly 3 clauses
- Each variable appears uncomplemented at least once
- Each variable appears complemented at least once

- Example: $(X_1 \vee X_2 \vee \overline{X_3}) \wedge (\overline{X_1} \vee X_2 \vee \overline{X_3}) \wedge (X_1 \vee \overline{X_2} \vee X_3)$

Multi-Agent Path Finding (MAPF)

- Example: $(X_1 \vee X_2 \vee \overline{X_3}) \wedge (\overline{X_1} \vee X_2 \vee \overline{X_3}) \wedge (X_1 \vee \overline{X_2} \vee X_3)$



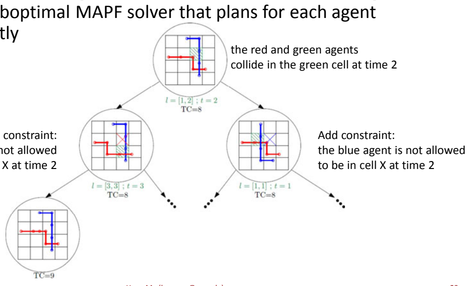
$X_1 \equiv false$ $X_2 \equiv true$ $X_3 \equiv true$

Multi-Agent Path Finding (MAPF)

- Makespan is 3 if and only if (≤3, =3)-SAT instance is satisfiable
- Makespan is 4 if and only if (≤3, =3)-SAT instance is unsatisfiable
- Any MAPF approximation algorithm with ratio 4/3 – ϵ thus computes a MAPF plan with makespan 3 whenever the (≤3, =3)-SAT instance is satisfiable and therefore solves it

Conflict-Based Search with Highways

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Bounded-suboptimal MAPF solver that plans for each agent independently



the red and green agents collide in the green cell at time 2

Add constraint: the red agent is not allowed to be in cell X at time 2

Add constraint: the blue agent is not allowed to be in cell X at time 2

4-neighbor grid

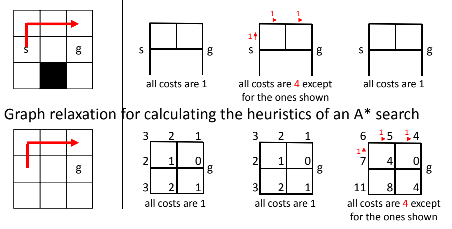
Conflict-Based Search with Highways

- 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

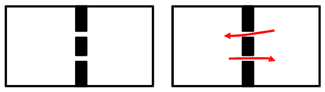
optimal | suboptimality bound 4

- Graph for an A* search
- Graph relaxation for calculating the heuristics of an A* search



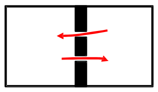
Conflict-Based Search with Highways

- Conflict-based search with highways (ECBS+HWY): Bounded suboptimal MAPF solver
  - Conflict-based search
  - Experience graphs create lanes (called highways) for the agents to avoid head-to-head collisions, which decreases the computation time of conflict-based search



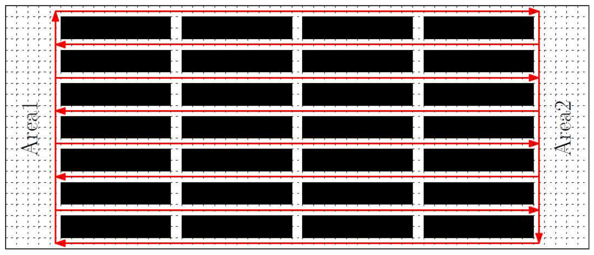
Conflict-Based Search with Highways

- Conflict-based search with highways (ECBS+HWY)
  - Highways provide consistency and thus predictability of agent movement, which might be important for human co- workers
  - Highways do not make MAPF instances unsolvable because they are only used as advice rather than hard constraints



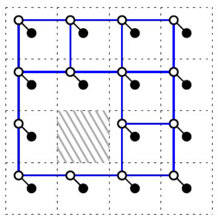
Conflict-Based Search with Highways

- Conflict-based search with highways (ECBS+HWY)



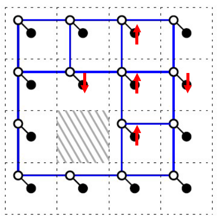
Conflict-Based Search with Highways

- Learning highways with graphical models
- Plan a shortest path for each agent independently
- Direction vector of a cell: Average of entry and exit directions of each path for the given cell
- Features
  - Collision?
  - Direction of direction vector (N, E, S, W)
  - Magnitude of direction vector
  - > 0.5?



Conflict-Based Search with Highways

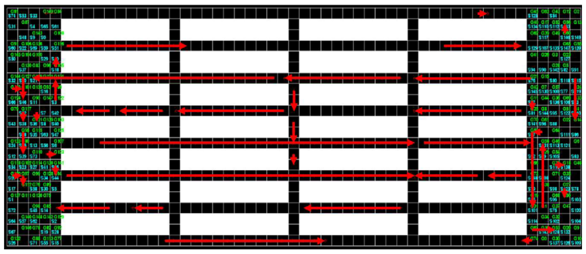
- Learning highways with graphical models
- Plan a shortest path for each agent independently
- Direction vector of a cell: Average of entry and exit directions of each path for the given cell
- Features
  - Collision?
  - Direction of direction vector (N, E, S, W)
  - Magnitude of direction vector
  - > 0.5?



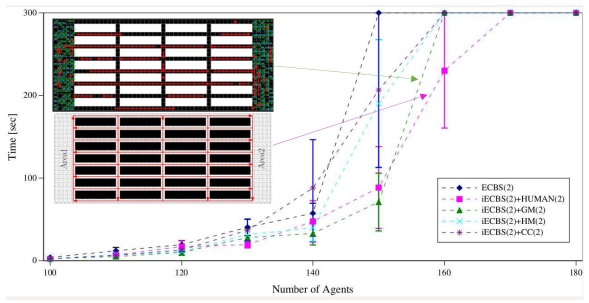
Conflict-Based Search with Highways

- 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)

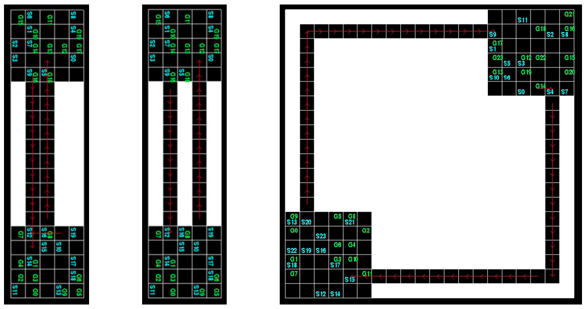
Conflict-Based Search with Highways



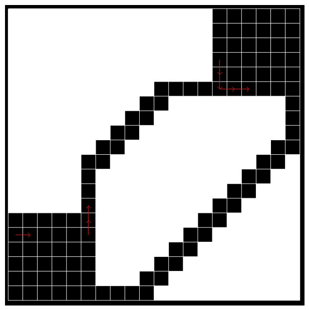
Conflict-Based Search with Highways



Conflict-Based Search with Highways



Conflict-Based Search with Highways

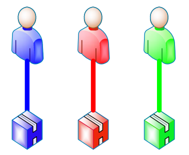


Conflict-Based Search with Highways

- 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.

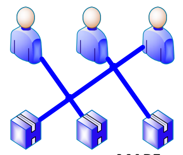
runs	time limit	38 "easy"	12 "hard"	50 total
1	300 sec	100.00%	0.00%	76.00%
3	100 sec	97.65%	96.87%	97.60%
5	60 sec	98.57%	98.81%	98.70%

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

(Non-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

Anonymous MAPF

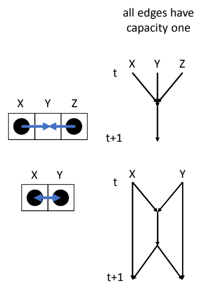
- Given: a number of agents (each with a start location), an equal number of goal locations, and a known environment
- Task: assign a different goal location to each agent and then find collision-free paths for the agents from their start to their goal locations that minimize makespan or flowtime

Anonymous MAPF

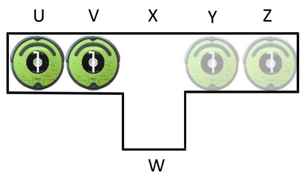
- Theorem [Yu and Lavalle]: An anonymous MAPF instance admits a MAPF plan with makespan at most T if and only if the time-expanded network with T periods admits a max flow of the number of agents.

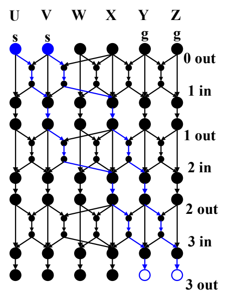
Anonymous MAPF

- 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

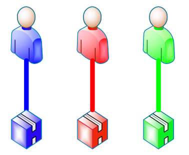


Anonymous MAPF



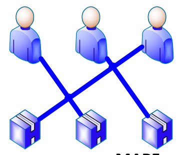


Target Assignment and Path Finding (TAPF)



non-anonymous MAPF

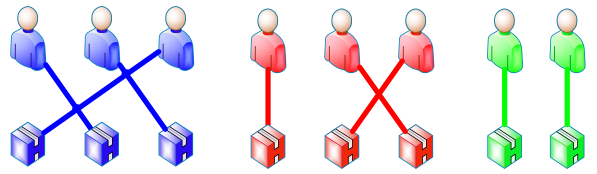
NP-hard
solved with A* approaches
e.g. conflict-based search or M*



anonymous MAPF

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

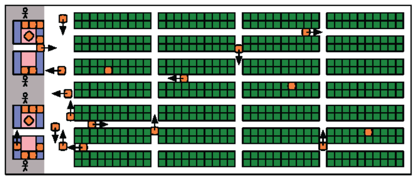
Target Assignment and Path Finding (TAPF)



TAPF = mix of non-anonymous and anonymous MAPF

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

Target Assignment and Path Finding (TAPF)



[Wurman, D'Andrea and Mountz]

Team 0: Agents that move from the packing stations to the storage locations

Team 1: Agents that move from the storage locations to Packing Station 1

Team 2: Agents that move from the storage locations to Packing Station 2

Team 3: Agents that move from the storage locations to Packing Station 3

Target Assignment and Path Finding (TAPF)

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

- Theorem: TAPF (with k>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)

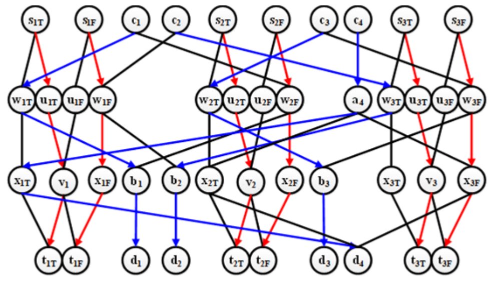
- Reduction from 2/$\overline{2}$/3-SAT: It is NP-complete to determine whether a given 2/$\overline{2}$/3-SAT instance is satisfiable

- Each variable appears in exactly 3 clauses
- Each variable appears uncomplemented in a clause of size two
- Each variable appears complemented in a clause of size two
- Each variable appears in a clause of size three

- Example: $(X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_3) \wedge (X_2 \vee \overline{X_3}) \wedge (X_1 \vee X_2 \vee \overline{X_3})$

Target Assignment and Path Finding (TAPF)

- Example: $(X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_3) \wedge (X_2 \vee \overline{X_3}) \wedge (X_1 \vee X_2 \vee \overline{X_3})$



Target Assignment and Path Finding (TAPF)

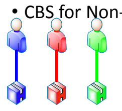
- Task: find the target assignments and collision-free paths that minimize the makespan.

- 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



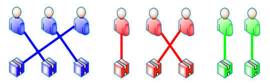
- CBS:
  1. Find paths for each single agent separately
  2. Look for collisions in paths
  3. If there is a collision between a_1 and a_2:

     Option 1 or **Option 2** to avoid collision
     - **Collision**: <Agent a_1, Agent a_2, Location *x*, Time *t*>
     - **Constraint**: <Agent, Location, Time>

       Option 1: a_1 cannot stay in *x* at time *t*.

       Option 2: a_2 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 *team1* and *team2*:

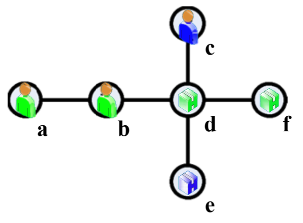
     Option 1 or **Option 2** to avoid collision
     - **Collision**: <Team *team1*, Team *team2*, Location *x*, Time *t*>
     - **Constraint**: <Team, Location, Time>

       Option 1: agents in *team1* cannot stay in *x* at time *t*.

       Option 2: agents in *team2* cannot stay in *x* at time *t*.

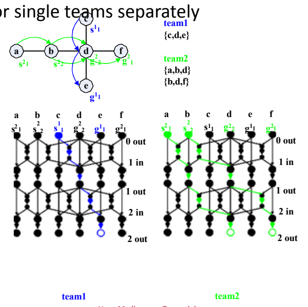
Target Assignment and Path Finding (TAPF)

- An example



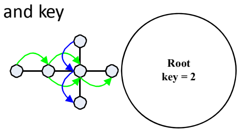
Target Assignment and Path Finding (TAPF)

- Finding paths for single teams separately



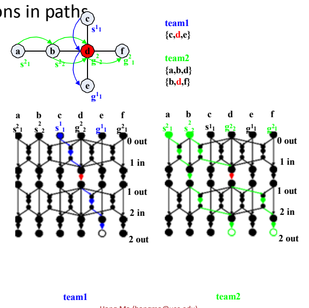
Target Assignment and Path Finding (TAPF)

- Store paths and key



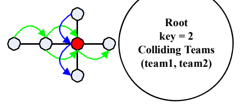
Target Assignment and Path Finding (TAPF)

- Look for collisions in paths



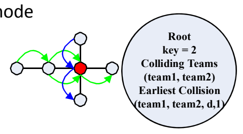
Target Assignment and Path Finding (TAPF)

- Store colliding teams



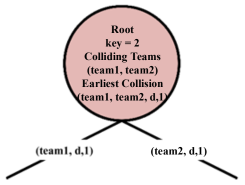
Target Assignment and Path Finding (TAPF)

- Pop a tree node



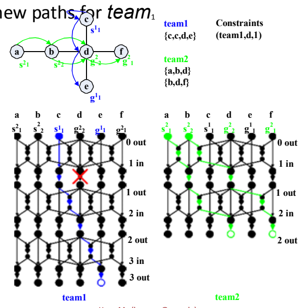
Target Assignment and Path Finding (TAPF)

- Two options



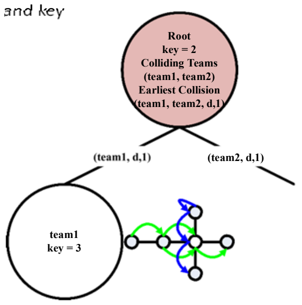
Target Assignment and Path Finding (TAPF)

- Option 1: find new paths for *team*$_1$



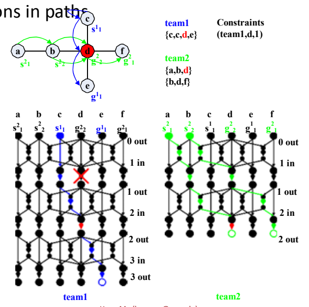
Target Assignment and Path Finding (TAPF)

- Store paths and key



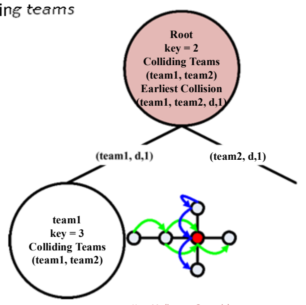
Target Assignment and Path Finding (TAPF)

- Look for collisions in paths



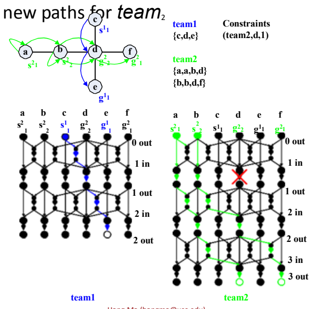
Target Assignment and Path Finding (TAPF)

- Store colliding teams



Target Assignment and Path Finding (TAPF)

- Option 2: find new paths for *team*$_2$

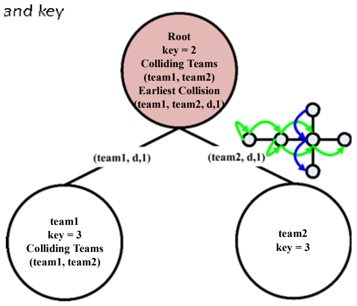


team1	Constraints
{c,d,e}	(team2,d,1)

team2
{a,a,b,d}
{b,b,d,f}

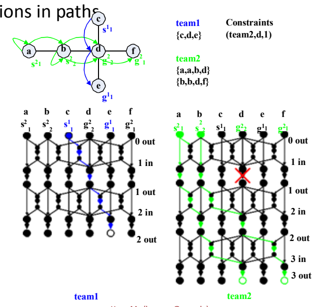
Target Assignment and Path Finding (TAPF)

- Store paths and key



Target Assignment and Path Finding (TAPF)

- Look for collisions in paths

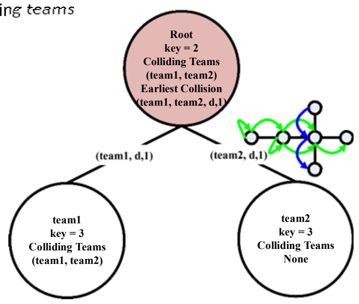


team1	Constraints
{c,d,e}	(team2,d,1)

team2
{a,a,b,d}
{b,b,d,f}

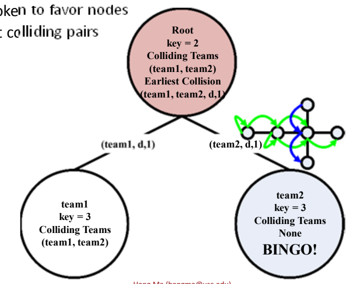
Target Assignment and Path Finding (TAPF)

- Store colliding teams



Target Assignment and Path Finding (TAPF)

- Pop a tree node
  Ties are broken to favor nodes with fewest colliding pairs



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

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.

	CBM		Unweighted CBM	
agents	time	success	time	success
10	0.34	1	0.41	0.72
15	0.57	1	1.06	0.44
20	0.78	1	2.06	0.22
25	1.07	1	1.58	0.08
30	1.71	1	6.73	0.02
35	1.92	1	-	0
40	2.95	1	-	0
45	3.66	1	-	0
50	5.32	1	-	0

Target Assignment and Path Finding (TAPF)

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

Target Assignment and Path Finding (TAPF)

- Comparisons
  - Setups:

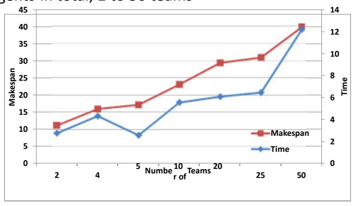
30× 30 4-neighbor grids with 10% randomly blocked cells.
5-minute time limits.
CBM: specialized solver
Versus
ILP (Integer Linear Program): useful tool and easy to model

	CBM		ILP	
agents	time	success	time (over solved instances)	success
10	0.34	1	18.24	1
15	0.57	1	35.44	1
20	0.78	1	62.85	0.94
25	1.07	1	88.55	0.82
30	1.71	1	108.75	0.66
35	1.92	1	121.99	0.46
40	2.95	1	152.98	0.14
45	3.66	1	161.52	0.14
50	5.32	1	161.95	0.04

Target Assignment and Path Finding (TAPF)

- Spectrum: Anonymous⟷Non-Anonymous

Fixed 100 agents in total, 2 to 50 teams



[2 teams, 50 agents per team] ⟷ [50 teams, 2 agents per team]

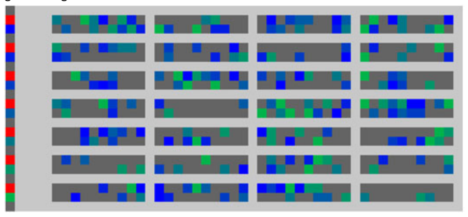
Anonymous ⟷ Non-Anonymous

P ⟷ **NP**-hard

Target Assignment and Path Finding (TAPF)

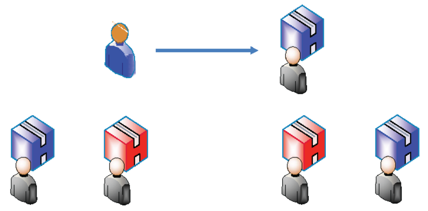
- Scalability: simulated warehouse system

Each instance has 420 agents: 210 “incoming” and 210 “outgoing”
CBM solves 40 out of the 50 Kiva instances within a time limit of 5 minutes each
Average running time over solved instances is 91.61 seconds

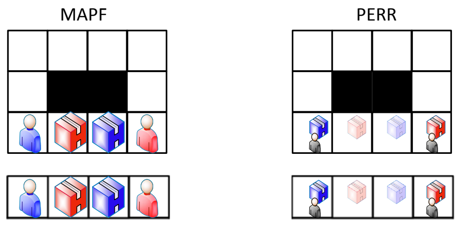


Package Exchange Robot Routing (PERR)

- The package exchange robot routing problem (PERR)
  - 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

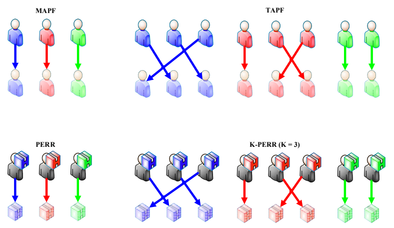


Package Exchange Robot Routing (PERR)



Package Exchange Robot Routing (PERR)

K-type package-exchange robot routing (K-PERR)

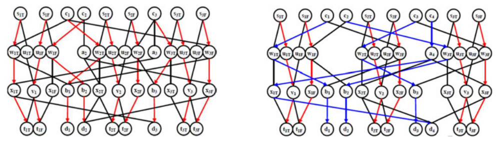


Package Exchange Robot Routing (PERR)

- Theorem: All (K-)PERR instances are solvable (as long as all goal locations are different and all agents are in the same connected components as their goal locations)
- Theorem: Plans with polynomial makespans and flowtimes can be found in polynomial time

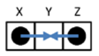
Package Exchange Robot Routing (PERR)

- Theorem: PERR (with k>1 groups) is NP-hard to solve optimally for makespan or flowtime minimization
- Theorem: PERR (with k>1 groups) is NP-hard to approximate within any factor less than 4/3 for makespan minimization on graphs in general
- Reductions from $\leq 3,=3$-SAT or $2/\overline{2}/3$-SAT as before (because transfers do not help for our constructions)

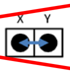


Package Exchange Robot Routing (PERR)

- 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~~



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

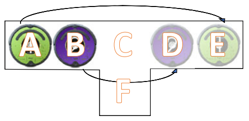
Execution of MAPF Plans

- Planning uses models that are not completely accurate
  - Robots are not completely synchronized
  - 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

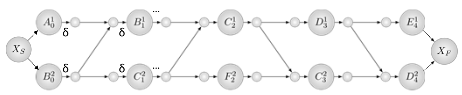
Execution of MAPF Plans



Agent 1 A → B → C → D → E
Agent 2 B → C → F → C → D

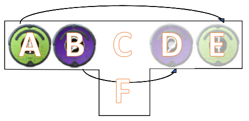
Precedence Graph

vertex = event that an agent arrives at a location



4-neighbor grid

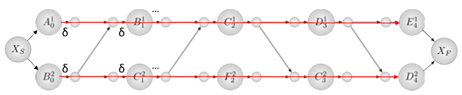
Execution of MAPF Plans



Agent 1 A → B → C → D → E
Agent 2 B → C → F → C → D

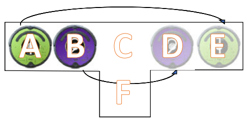
Precedence Graph

Type 1 edge = order in which the same agent arrives at locations



4-neighbor grid

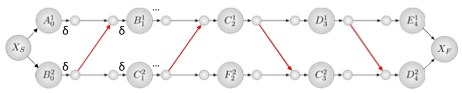
Execution of MAPF Plans



Agent 1 A → B → C → D → E
Agent 2 B → C → F → C → D

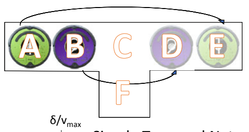
Precedence Graph

Type 2 edge = order in which two different agents arrive at the same location



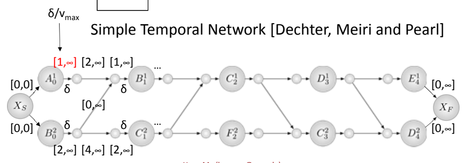
4-neighbor grid

Execution of MAPF Plans



Agent 1 A → B → C → D → E
Agent 2 B → C → F → C → D

Simple Temporal Network [Dechter, Meiri and Pearl]



4-neighbor grid

Execution of MAPF Plans

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

Minimize $\sum_{j=1}^{K} t(v^j)$
such that $t(X_S) = 0$
and, for all $e = (v, v') \in \mathcal{E}'$,
$t(v') - t(v) \geq LB(e)$
$t(v') - t(v) \leq UB(e)$

polynomial time

Execution of MAPF Plans

- 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 $2\delta v_{min}/v_{max}$

$\boxed{\text{Maximize } v^*_{min}}$
such that $t(X_S) = 0$
and, for all $e = (v, v') \in \mathcal{E}'$,
$t(v') - t(v) \geq LB(e)$
$t(v') - t(v) \leq UB(e)$
$\boxed{t(v') - t(v) \leq l(e)(v^*_{\min})^{-1} \text{ if } e \text{ is a Type 1 edge}}$

polynomial time

Execution of MAPF Plans

- Main loop
  - Run Conflict-Based Search with Highways to find a MAPF plan (slow)

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

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

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

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
  - If they do not exist, then
  - 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

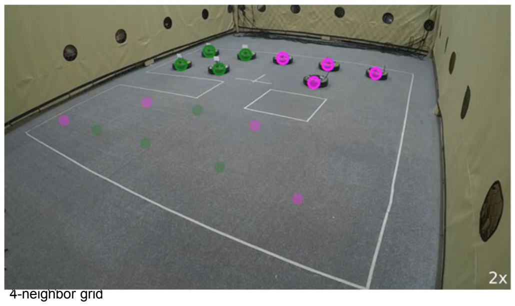
- MAPF solver: ECBS+HWY
- MAPF-POST: C++, boost graph library, Gurobi LP solver
- PC: i7-4600U 2.1 GHz, 12 GB RAM
- Terrain: 4x3 gridworld with 1m^2 cells and δ = 0.4m
- Architecture: ROS with decentralized execution
  - Robot controller with state [x,y,Θ] (attempts to meet deadline)
  - PID controller (corrects 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

8x

Execution of MAPF Plans



4-neighbor grid

2x

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)

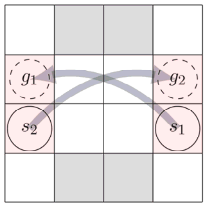
- MAPD Algorithms

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

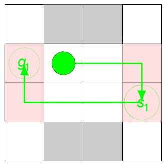


Multi-Agent Pickup and Delivery (MAPD)

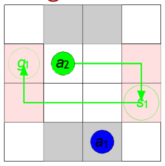
- In order to execute a task, the agent has to move from its current location via the pickup location to the delivery location:
  1. When the agent reaches the pickup location, it starts to execute the task
  2. When it reaches the delivery location, it finishes the task

MAPD: Executing Task

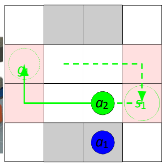
- In order to execute a task, the agent has to move from its current location via the pickup location to the delivery location:
  1. When the agent reaches the pickup location, it starts to execute the task.
  2. When it reaches the delivery location, it finishes the task.



MAPD: Free Agents



MAPD: Occupied Agents



MAPD: Assignment of Agents to Tasks

- A free agent can be assigned to any unexecuted task

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

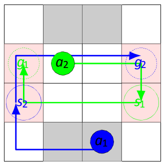
MAPD: Objective

- Finish executing each task as quickly as possible.

MAPD: Effectiveness of a MAPD algorithm

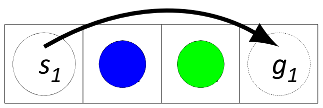
- Service time: the average number of timesteps needed to finish executing each task after it enters the system.
- An algorithm solves a MAPD instance $\Longleftrightarrow$ Service time of all tasks is bounded.

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



MAPD: Solvability

- Not every MAPD instance is solvable



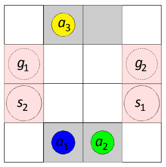
MAPD: Well-Formed MAPD Instances

- Being **well-formed** (based on [M. Cáp, Vokr´inek 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

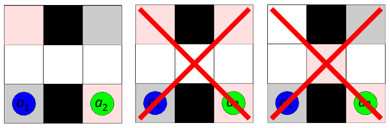
Task Parking Locations: all pickup and delivery locations of tasks (storage locations, inventory stations, etc.)

Non-task Parking Locations:
- All initial locations of agents
- Additional designated parking locations



MAPD: Well-Formed MAPD Instances

1. # tasks is finite;
2. # non-task parking locations ≥ # agents;
3. For any two parking locations, there exists a path between them that traverses no other parking locations.



MAPD: MAPD Algorithms

We present

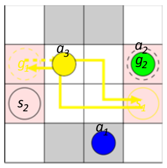
1. Two Decoupled Algorithms: complete for well-formed MAPD instances (solve all well-formed instances)
   Token Passing (**TP**)
   Token Passing with Task Swaps (**TPTS**)
2. One Centralized Algorithm:
   CENTRAL

MAPD: A Running Example

Unexecuted Tasks: *task*$_1$, *task*$_2$.

Agent a_1 and agent a_2 are resting

Agent a_3 is assigned to *task*$_1$ and on the way to the pickup location s_1



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, plan its path, and passes the token to the next agent

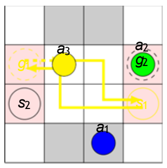
Key idea of TP:

- 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: *task*$_2$

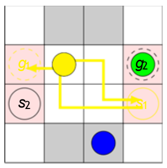
Agent a_1 and agent a_2 request for token



MAPD: TP: Agent a_1's Turn

Agent a_1 Has Token

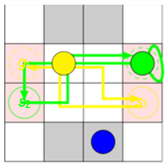
1. it cannot assign itself to any task because agent a_2 rests in g_2, the only task available to it
2. it has to rest in a parking location that will not create any deadlock
3. it can continue to rest in its current location



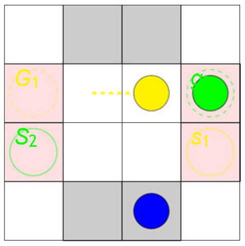
MAPD: TP: Agent a_2's Turn

Agent a_2 Has Token

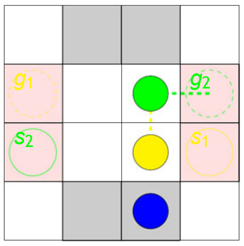
1. it assigns itself to $task_2$
2. $task_2$ is no longer available to other agents
3. it plans a cost-minimal collision-free path to execute $task_2$



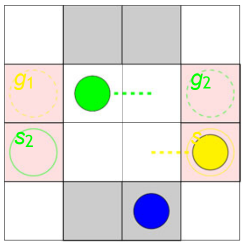
MAPD: TP: Animation



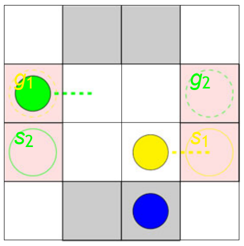
MAPD: TP: Animation



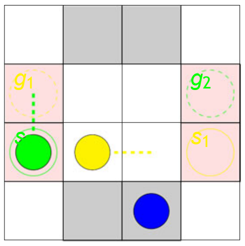
MAPD: TP: Animation



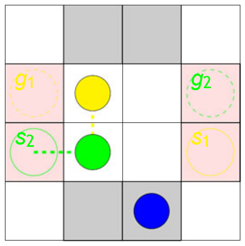
MAPD: TP: Animation



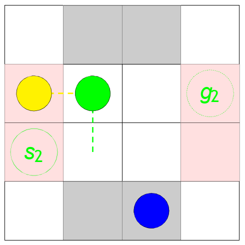
MAPD: TP: Animation



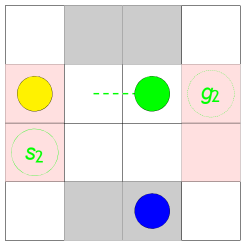
MAPD: TP: Animation



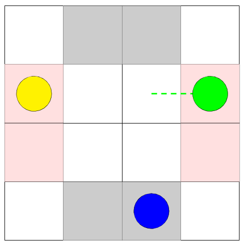
MAPD: TP: Animation



MAPD: TP: Animation



MAPD: TP: Animation



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)

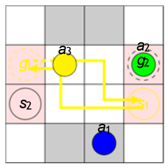
MAPD: Token Passing with Task Swaps (TPTS)

- 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: $task_1$, $task_2$

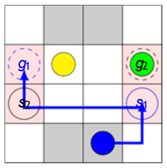
Agent a_1 and agent a_2 request for token



MAPD: TPTS: Agent a_1's Turn

Agent a_1 has token

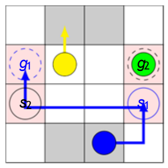
Agent a_1 grabs $task_1$ from agent a_3



MAPD: TPTS: Agent a_3 Making Decisions

Agent a_3 has token

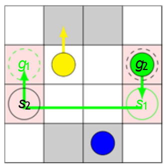
Agent a_3 moves to a parking location that will not create any deadlock in the future



MAPD: TPTS: Agent a_2's Turn

Agent a_2 has token

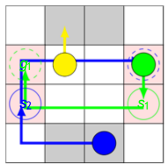
Agent a_2 grabs $task_1$ from agent a_1



MAPD: TPTS: Agent a_1 Making Decisions

Agent a_1 has token

Agent a_1 assigns itself to $task_2$



MAPD: TPTS: Completeness

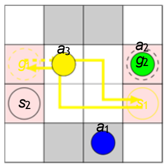
- Theorem: TPTS solves all well-formed MAPD instances

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

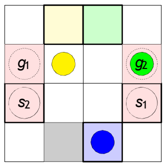
MAPD: CENTRAL: Running Example

- Tasks available for assignment: *task*$_1$, *task*$_2$



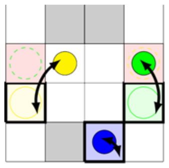
MAPD: CENTRAL: Candidate Parking Locations

- Pickup locations s_1 and s_2 + three additional "good" parking locations, one for each agent:



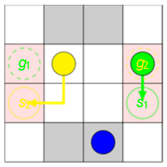
MAPD: CENTRAL: Assignment

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

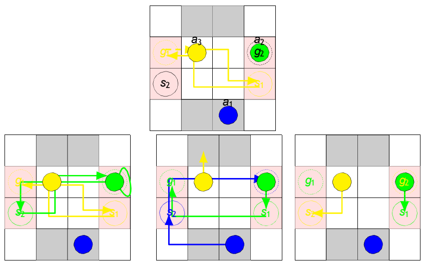


MAPD: CENTRAL: Path Finding

- 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

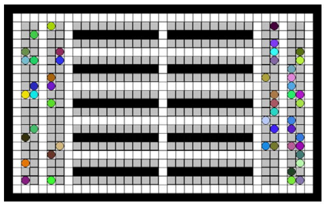


MAPD: Comparisons of Three Algorithms



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

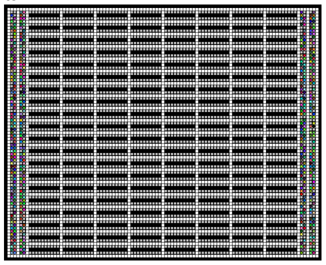


MAPD: Experimental Results

- 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: Experiments Setup

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



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

agents	100	200	300	400	500
service time	463.25	330.19	301.97	289.08	284.24
runtime (milliseconds)	90.83	538.22	1,854.44	3,881.11	6,121.06

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

MAPD in Continuous Time

MAPD in Continuous Time

4-neighbor grid

8x

MAPD in Continuous Time

4-neighbor grid

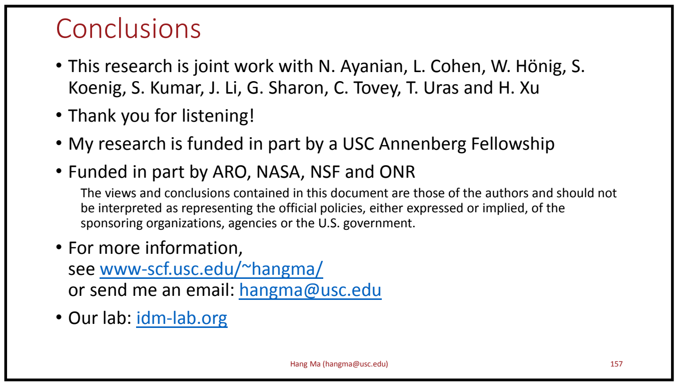
8x

Publications Covered

- H. Ma, S. Kumar and S. Koenig. Multi-Agent Path Finding with Delay Probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3605-3612, 2017
- H. Ma, J. Li, S. Kumar, S. Koenig. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 837-845, 2017
- H. Ma, W. Hönig, L. Cohen, H. Xu, S. Kumar, N. Ayanian, S. Koenig. Overview: A Hierarchical Framework for Plan Generation and Execution in Multi-Robot Systems. *IEEE Intelligent Systems*, to appear.
- W. Hönig, S. Kumar, H. Ma, L. Cohen, H. Xu, S. Koenig, N. Ayanian. Path Finding for Multi-Robot Systems with Kinematic Constraints in Occluded Environments. *Journal of Artificial Intelligence Research*, to appear.
- H. Ma, C. Tovey, G. Sharon, S. Kumar and S. Koenig. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3166-3173, 2016
- H. Ma and S. Koenig. Optimal Target Assignment and Path Finding for Teams of Agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1144-1152, 2016
- L. Cohen, T. Uras, S. Kumar, H. Xu, N. Ayanian and S. Koenig. Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 3067-3074, 2016
- W. Hönig, S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian and S. Koenig. Multi-Agent Path Finding with Kinematic Constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 477-485, 2016 (Outstanding Paper Award in the ICAPS-16 Robotics Track)
- W. Hönig, S. Kumar, H. Ma, S. Koenig and N. Ayanian. Formation Change for Robot Groups in Occluded Environments. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 4836-4842, 2016
- H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, S. Kumar, T. Uras, H. Xu, C. Tovey and G. Sharon. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. In *Proceedings of IJCAI-16 Workshop on Multi-Agent Path Finding*, 2016

References

- G. Sharon, R. Stern, A. Felner and N. Sturtevant. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence* 219:40-66, 2015.
- J. Yu and S. LaValle. Planning Optimal Paths for Multiple Robots on Graphs. In Proceedings of the *IEEE International Conference on Robotics and Automation (ICRA),* 3612-3617, 2013
- J. Yu and S. LaValle. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI),* 1444-1449, 2013
- M. Phillips, B. Cohen, S. Chitta and M. Likhachev. E-Graphs: Bootstrapping Planning with Experience Graphs. In *Proceedings of the Robotics: Science and Systems Conference (RSS),* 2012.
- P. Wurman, R. D'Andrea and M. Mountz. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* 29(1):9-20, 2008.
- R. Dechter, I. Meiri, J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61-95, 1991.



Conclusions

- This research is joint work with N. Ayanian, L. Cohen, W. Hönig, 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

Hang Ma (hangma@usc.edu) 157