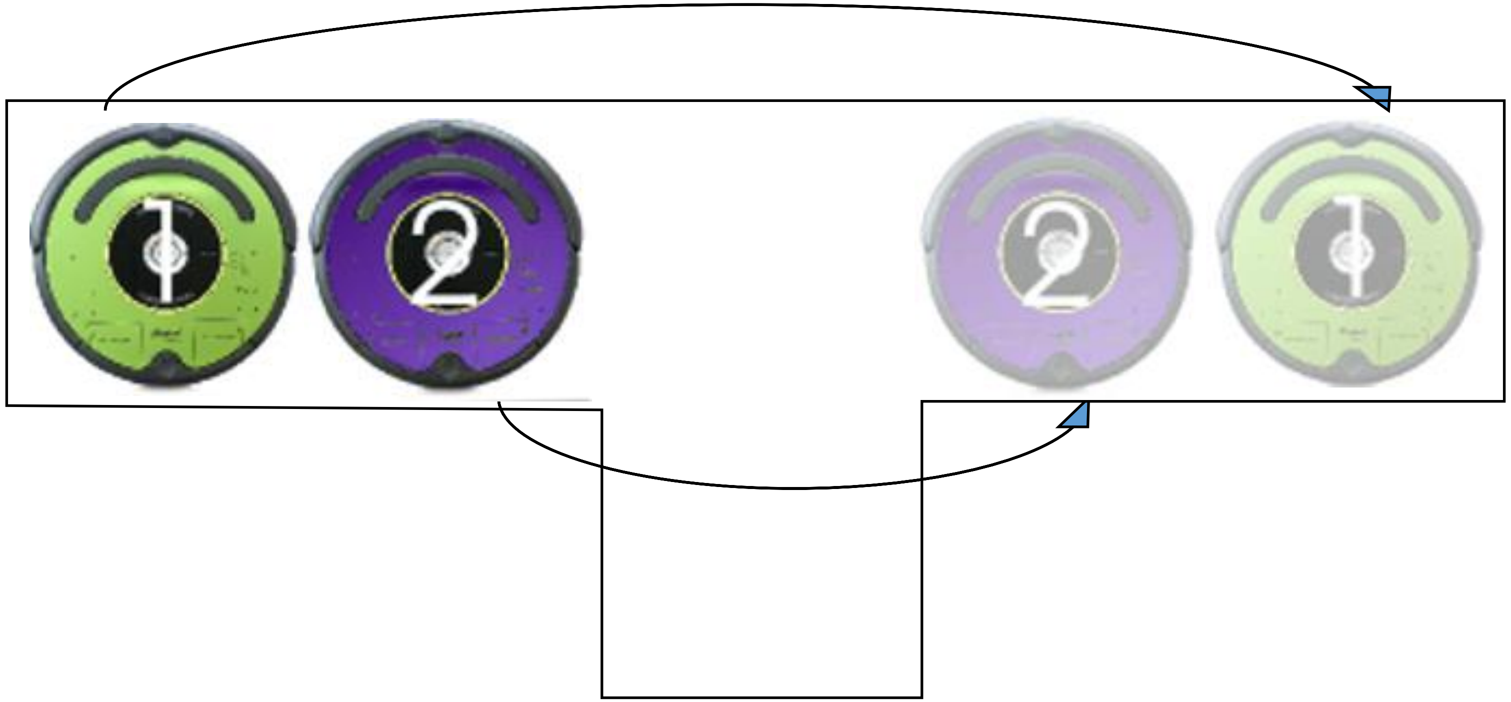
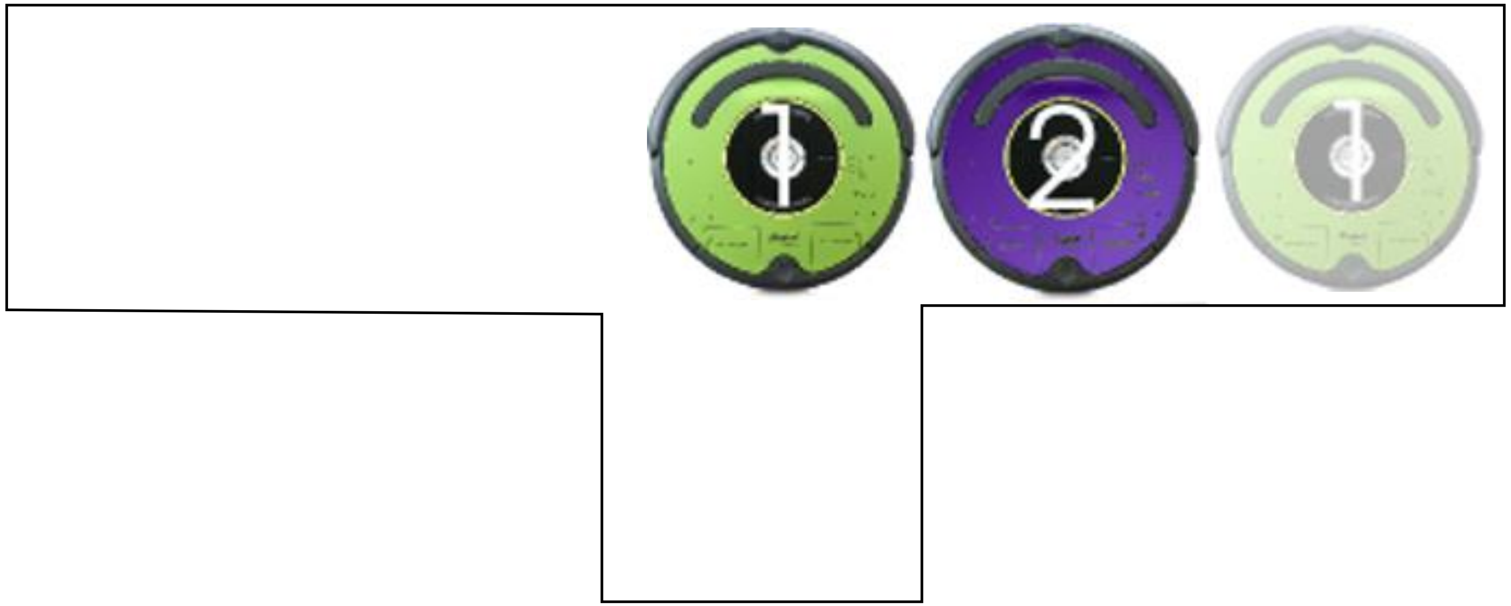
Multi-Agent Path Finding

Multi-Agent Path Finding (MAPF)

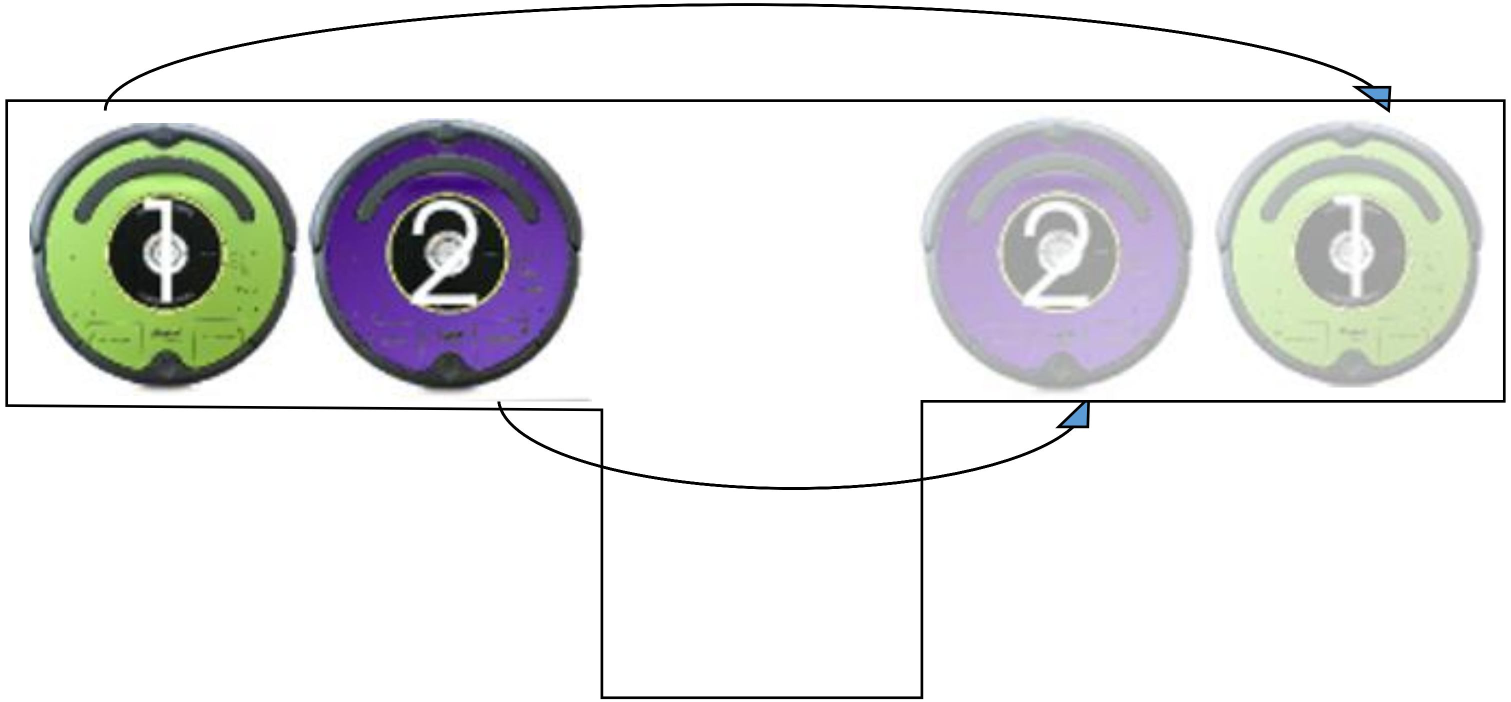


Multi-Agent Path Finding (MAPF)

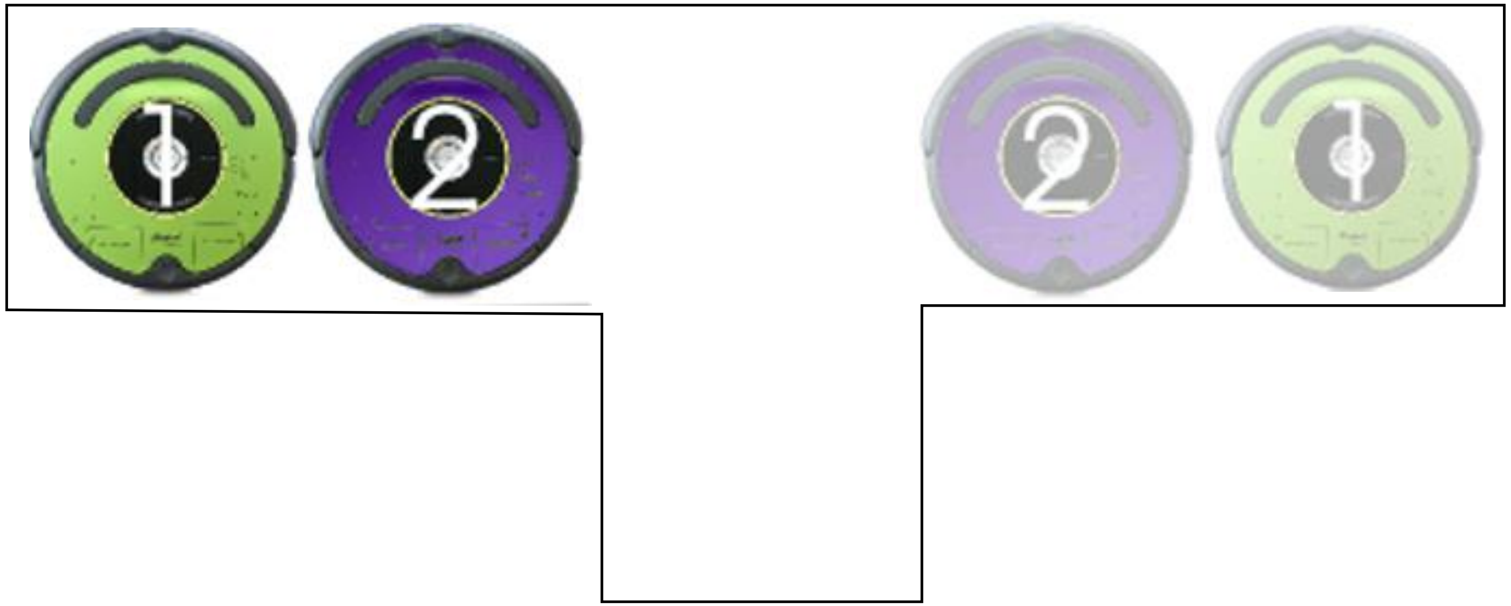
Multi-Agent Path Finding (MAPF)



Multi-Agent Path Finding (MAPF)



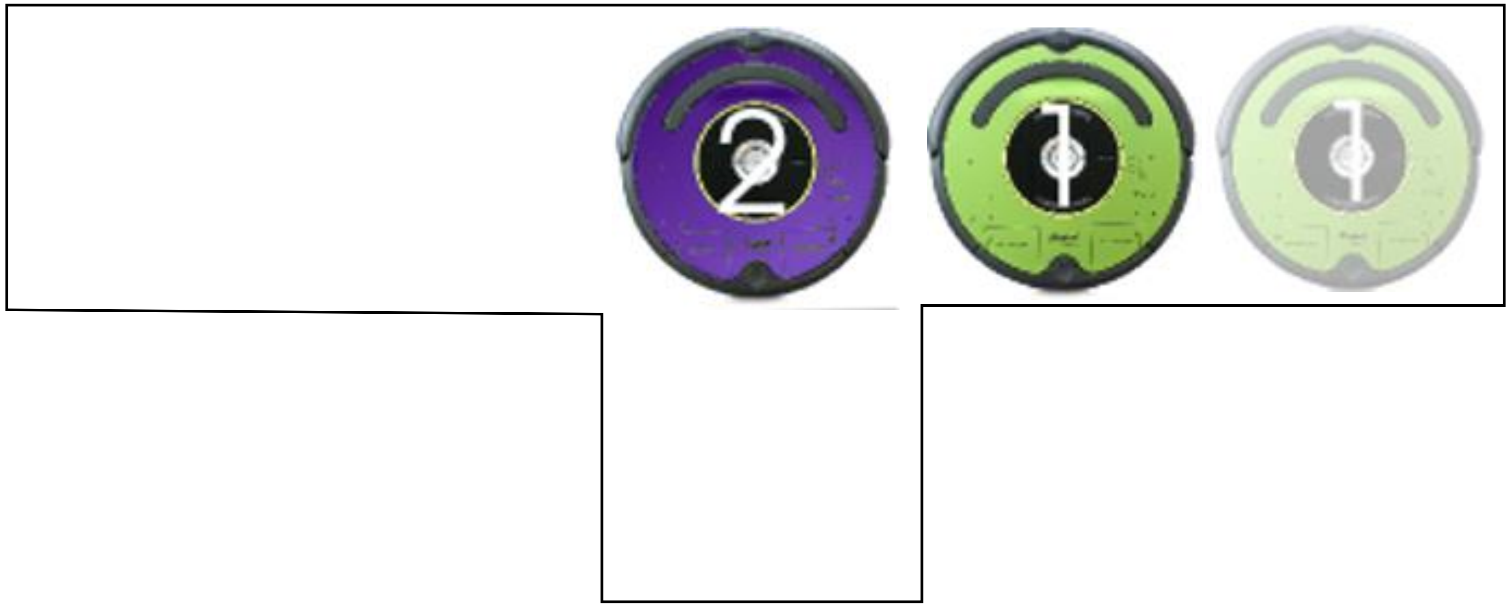
Multi-Agent Path Finding (MAPF)



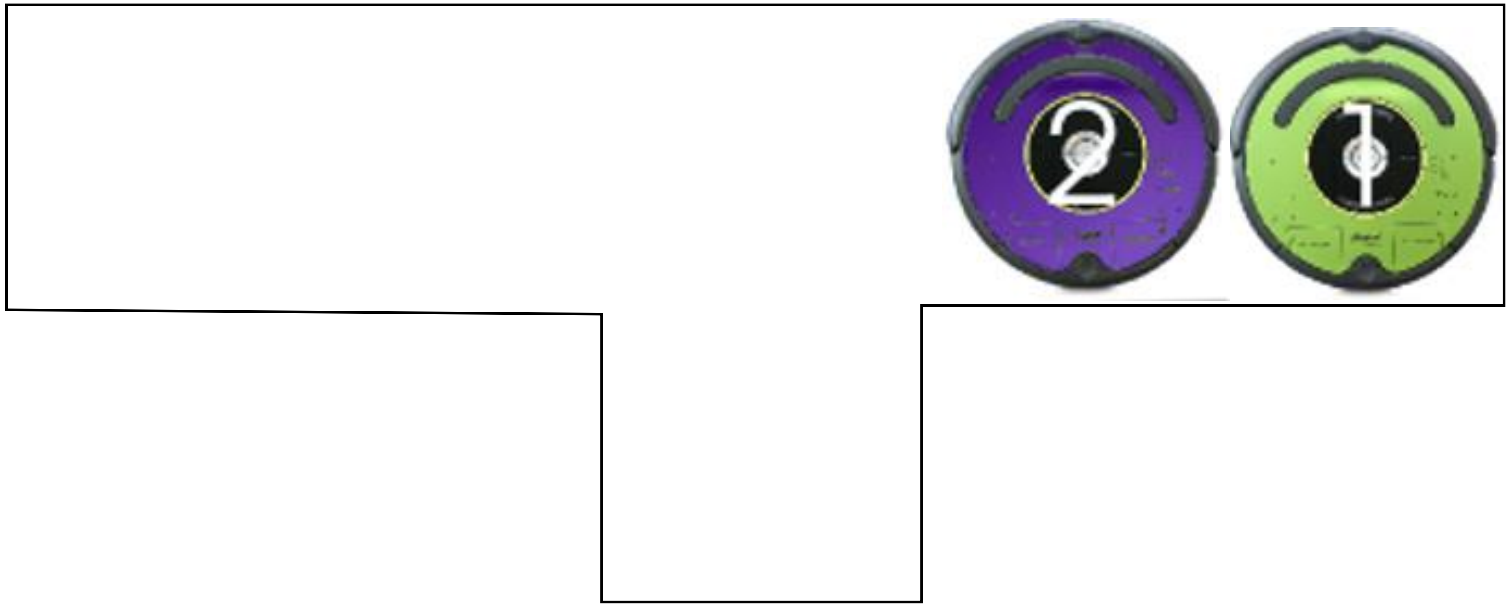
Multi-Agent Path Finding (MAPF)

Multi-Agent Path Finding (MAPF)

Multi-Agent Path Finding (MAPF)



Multi-Agent Path Finding (MAPF)

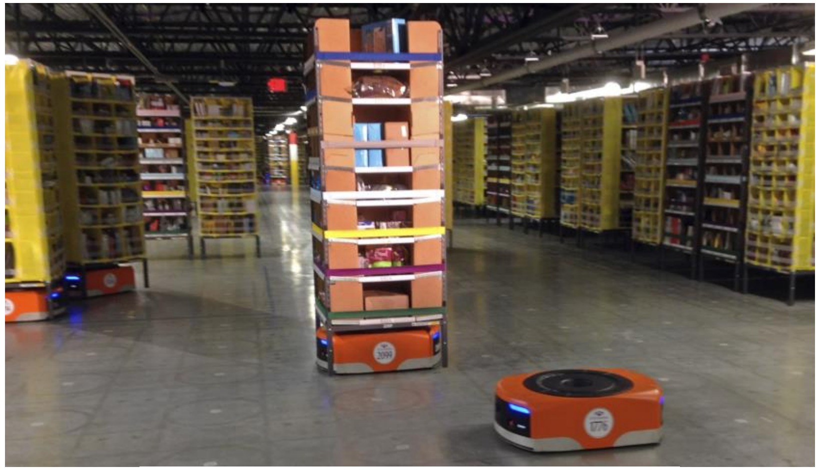


- Optimization problem with the objective
  to minimize task-completion time (called makespan) or
  the sum of travel times (called flowtime)

Multi-Agent Path Finding (MAPF)

- Application: Amazon fulfillment centers
- 2003 Kiva Systems founded
- 2012 Amazon acquires Kiva Systems for $775 million
- 2015 Kiva Systems becomes Amazon Robotics

[www.npr.org – Getty Images]



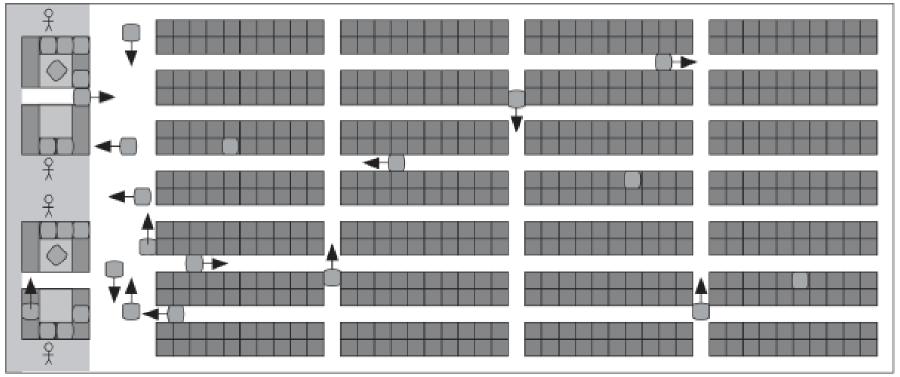
[www.theguardian.com - AP]

- > 3,000 robots on > 110,000 square meters in Tracy, California

Multi-Agent Path Finding (MAPF)

- Application: Amazon fulfillment centers

[from: YouTube]



[Wurman, D’Andrea and Mountz]

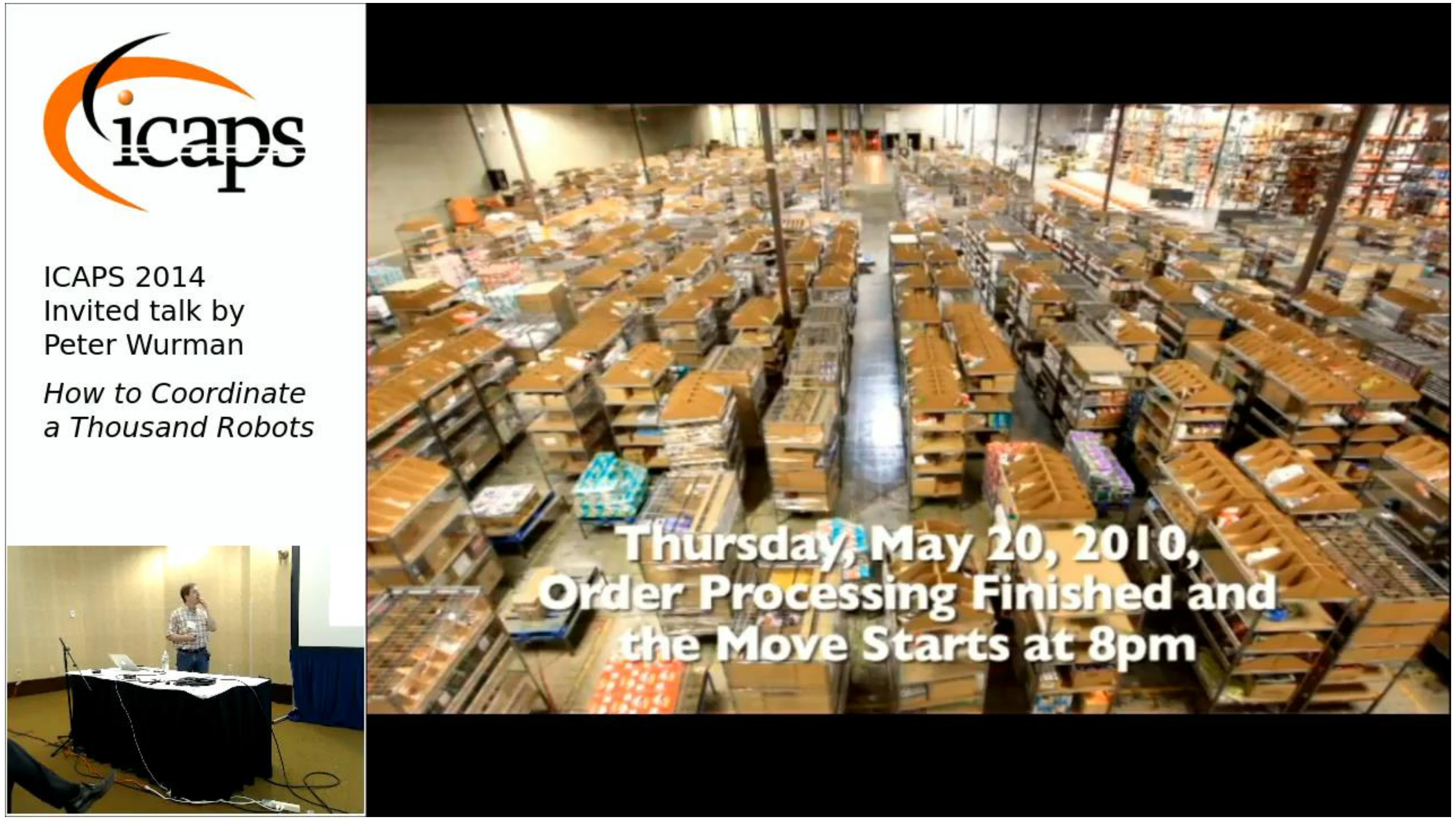
Multi-Agent Path Finding (MAPF)

- Application: Amazon fulfillment centers

[from: YouTube]

Multi-Agent Path Finding (MAPF)

- Application: Amazon fulfillment centers



[from: YouTube]

Multi-Agent Path Finding (MAPF)

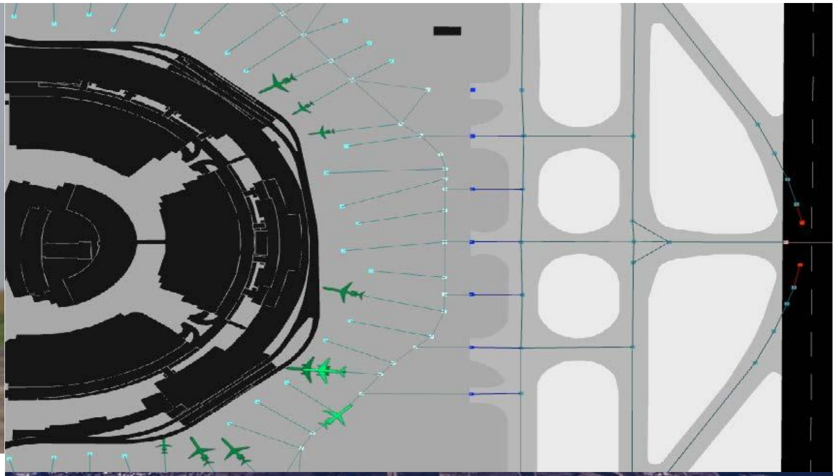
- Application: Automated warehousing in general

inVia Robotics

CAINIAO 菜鸟

Multi-Agent Path Finding (MAPF)

- Application: Autonomous engines-off taxiing

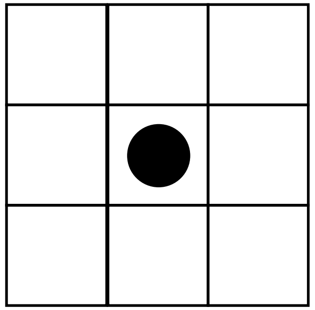


  - Reduce pollution
  - Reduce energy consumption
  - Reduce human workload
  - Reduce traffic congestion
  - Reduce airport size

Multi-Agent Path Finding (MAPF)

Robot

Agent



[from: YouTube]

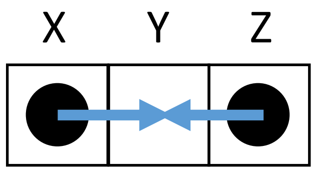
Stickers on the ground
establish a grid!

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

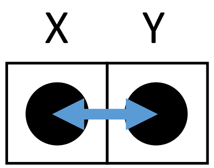
Multi-Agent Path Finding (MAPF)

- Each agent can move N, E, S or W into any adjacent unblocked cell (provided an agent already in that cell leaves it while the agent moves into it or earlier) or wait in its current 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



Multi-Agent Path Finding (MAPF)

- Suboptimal MAPF algorithms
  - Theorem [Yu and Rus]: MAPF can be solved in polynomial time on undirected grids without makespan or flowtime optimality
  - Unfortunately, good throughput is important in practice!

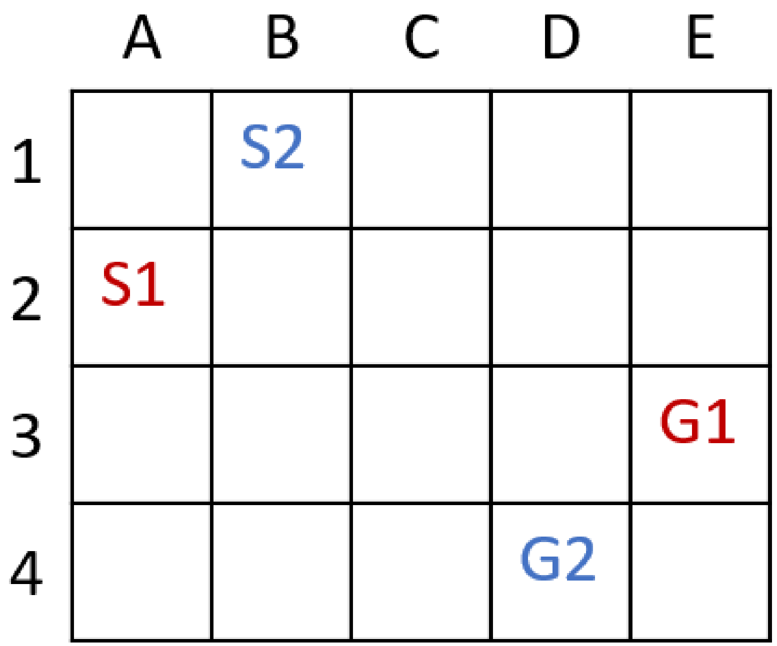
Multi-Agent Path Finding (MAPF)

- Optimal MAPF algorithms
  - Theorem [Yu and LaValle]: MAPF is NP-hard to solve optimally for makespan or flowtime minimization

[www.random-ideas.net]

- Bounded-suboptimal MAPF algorithms
  - Theorem [Ma, Tovey, Sharon, Kumar and Koenig]: 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)

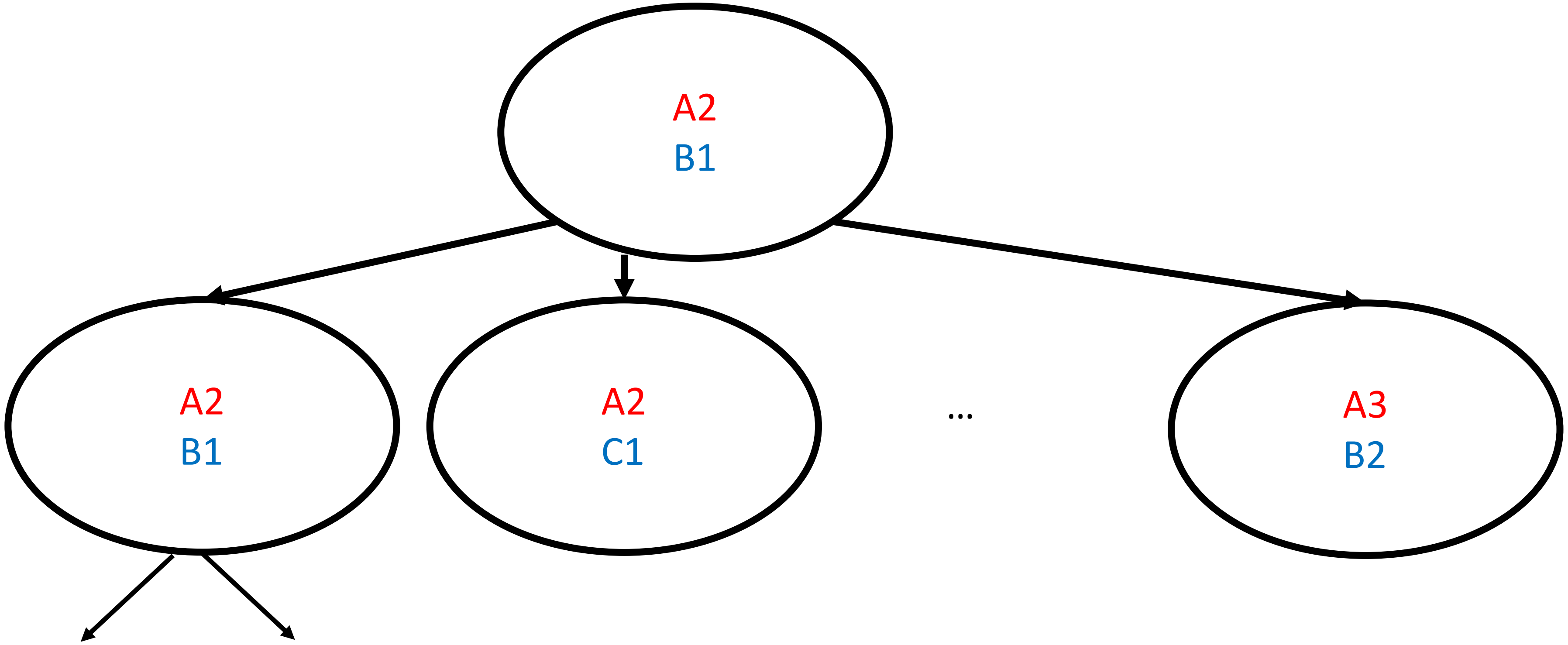


S1 (S2) = start cell of the red (blue) agent
G1 (G2) = goal cell of the red (blue) agent

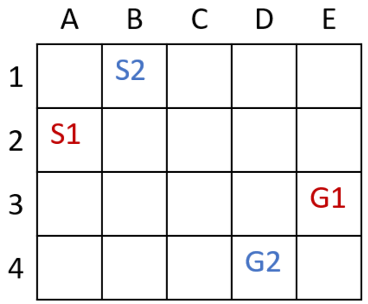
	A	B	C	D	E
1		S2			
2	S1				
3					G1
4				G2	

A*-Based Search

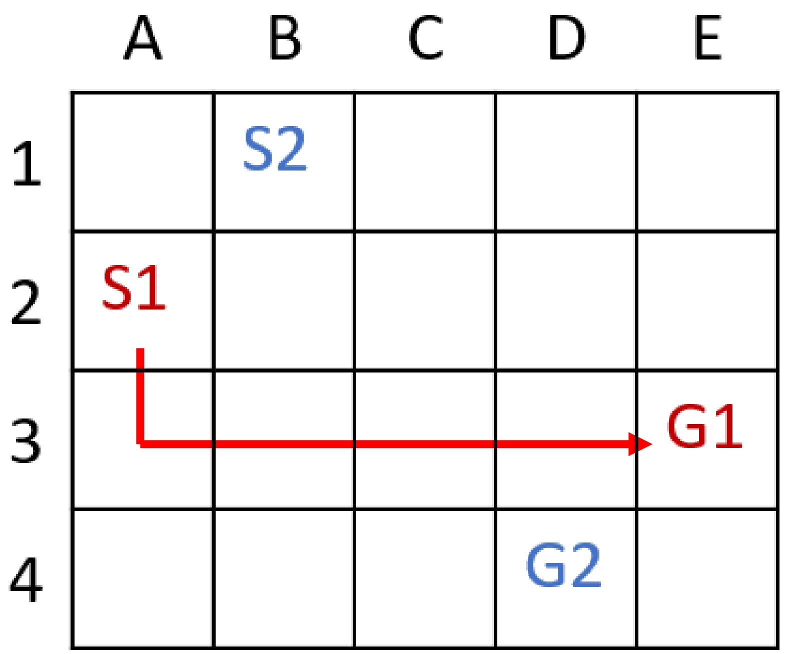
- A*-based search in the joint cell space: Optimal (or bounded-suboptimal) but extremely inefficient MAPF solver



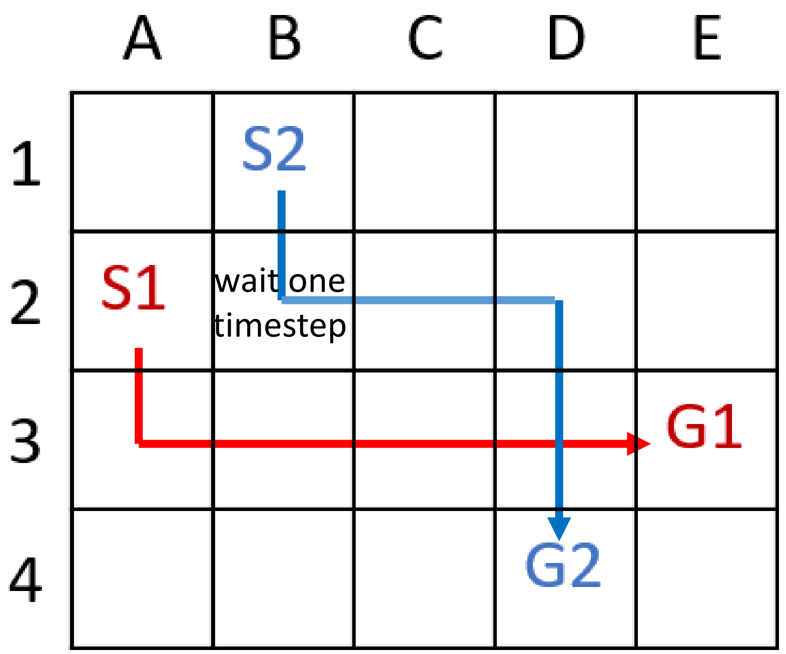
Prioritized Planning (Cooperative A*)



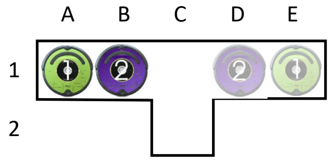
- Prioritized Planning (= sequential search: plan for one agent after another in space (= cell)-time space in a given order): efficient but suboptimal (and even incomplete) MAPF solver



First, find a time-minimal path for the agent with priority 1.

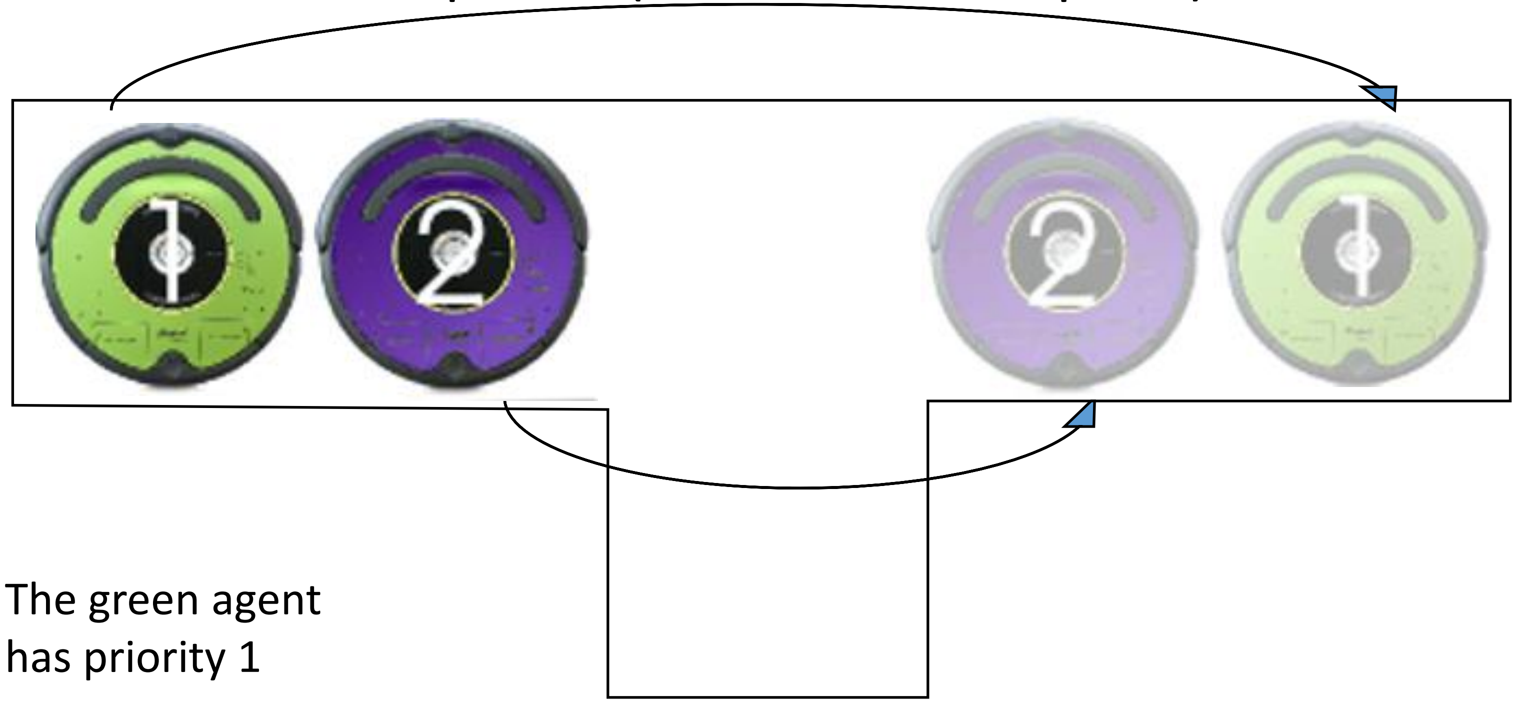


Then, find a time-minimal path for the agent with priority 2 that does not collide with the paths of higher-priority agents.



Prioritized Planning (Cooperative A*)

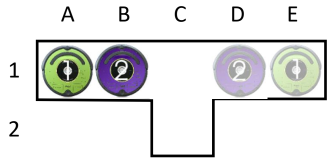
- Prioritized Planning (= sequential search: plan for one agent after another in space (= cell)-time space in a given order): efficient but suboptimal (and even incomplete) MAPF solver



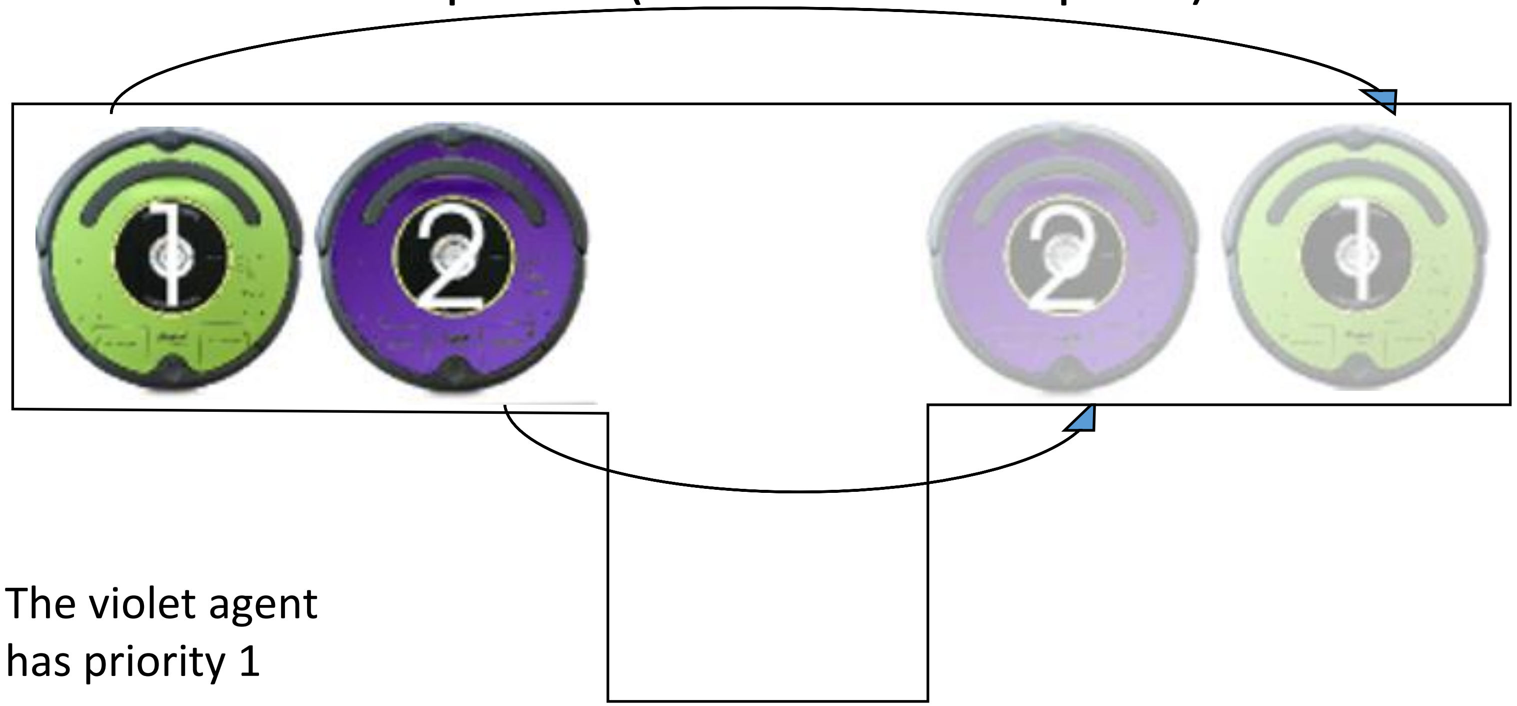
The green agent has priority 1

- Prioritized Planning finds first path A1, B1, C1, D1, E1 for the green agent and then path B1, C1, C2, C1, D1 for the violet agent. Thus, Prioritized Planning finds a solution.

Prioritized Planning (Cooperative A*)



- Prioritized Planning (= sequential search: plan for one agent after another in space (= cell)-time space in a given order): efficient but suboptimal (and even incomplete) MAPF solver



The violet agent
has priority 1

- Prioritized Planning finds first path B1, C1, D1 for the violet agent and then no path for the green agent. Thus, Prioritized Planning does not find a solution.

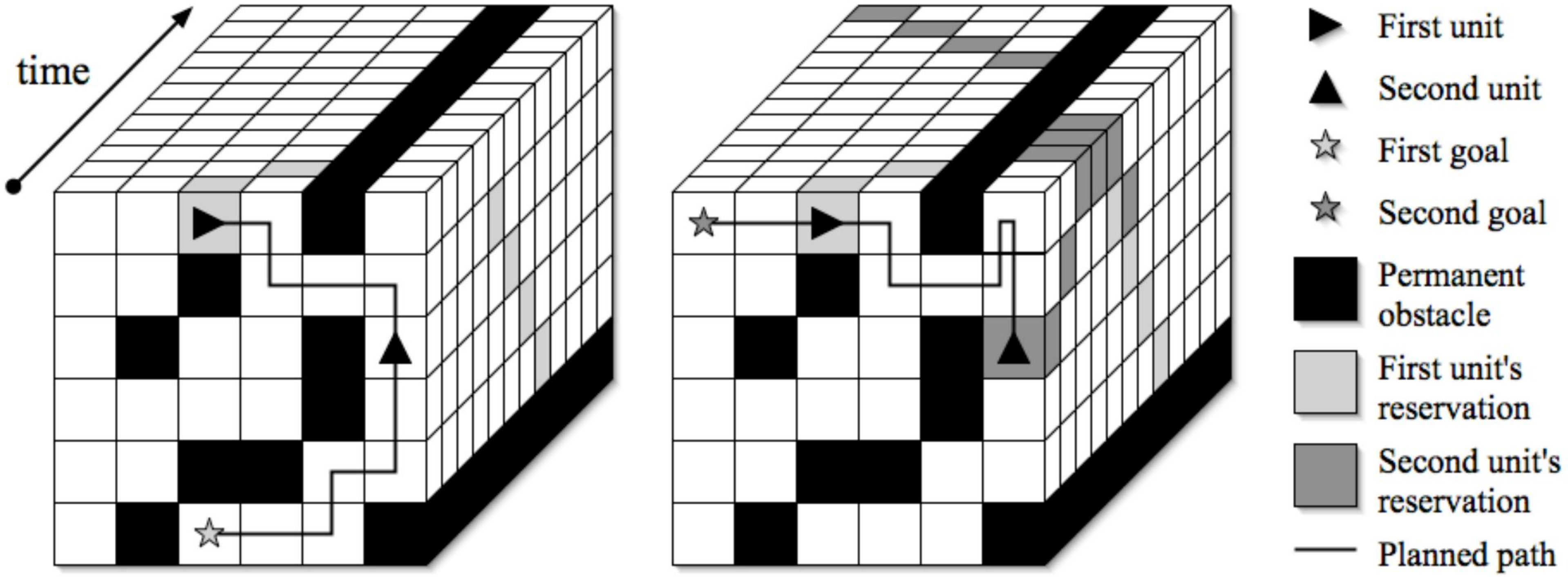
Prioritized Planning (Cooperative A*)

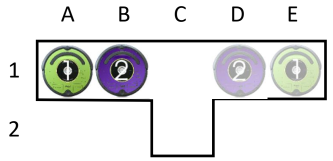
- You could implement space (= cell)-time A* with a reservation table (specific for a particular agent) as follows

- The states are pairs (cell, t) for all cells and times
- If the agent can move from cell X to cell Y (in the absence of other agents), create direct edges
  - from state (X,0) to state (Y,1)
  - from state (X,1) to state (Y,2)
  - …
- If the agent is not allowed to be in cell X at time t (because a collision with a higher-priority agent would result), delete state (X,t)
- If the agent is not allowed to move from cell X to cell Y at time t (because a collision with a higher-priority agent would result), delete the directed edge from state (X,t) to state (Y,t+1)
- Search the resulting state space for a time-minimal path from state (start cell, 0) to any state (goal cell, t) for all times t

Prioritized Planning (Cooperative A*)

- You could implement space (= cell)-time A* with a reservation table (specific for a particular agent) but you might not want to build it explicitly since it is often large.



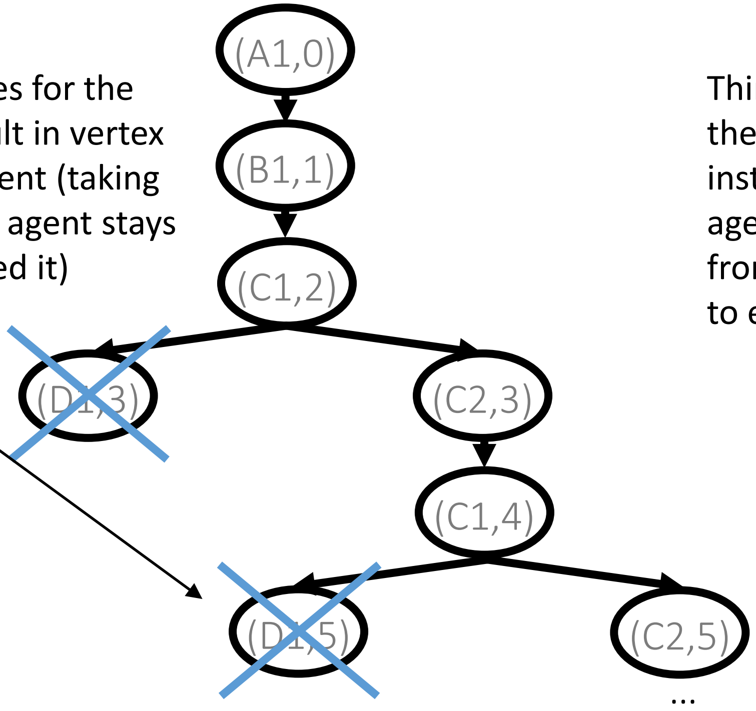


Prioritized Planning (Cooperative A*)

- You could implement space (= cell)-time A* with a reservation table (specific for a particular agent) but you might not want to build it explicitly since it is often large. Rather, you never want to generate the states or edges that you would have deleted in the reservation table in the A* search tree

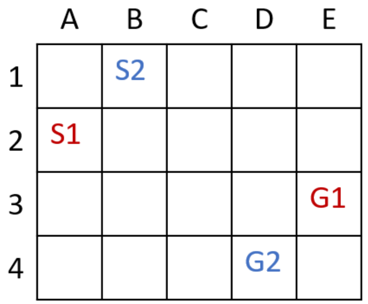
Do not generate these states for the green agent since they result in vertex collisions with the violet agent (taking into account that the violet agent stays in cell D3 once it has reached it)

The violet agent has priority 1



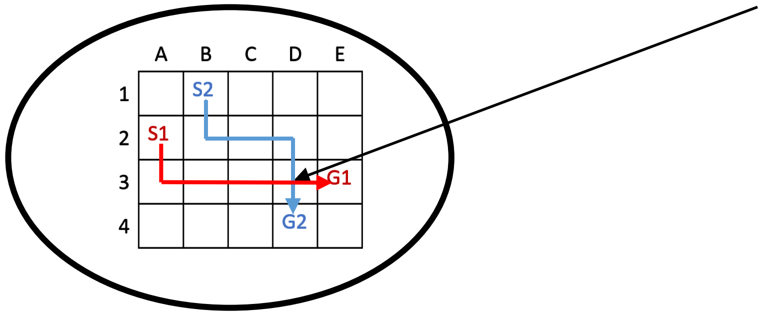
Think about how to detect that there is no path for the green agent instead of believing that the green agent can now repeatedly move from cell C1 via cell C2 back to cell C1 to eventually get to its goal cell E1

Conflict-Based Search (CBS)



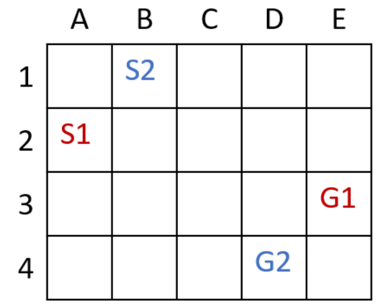
- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

Find time-minimal paths for all agents independently



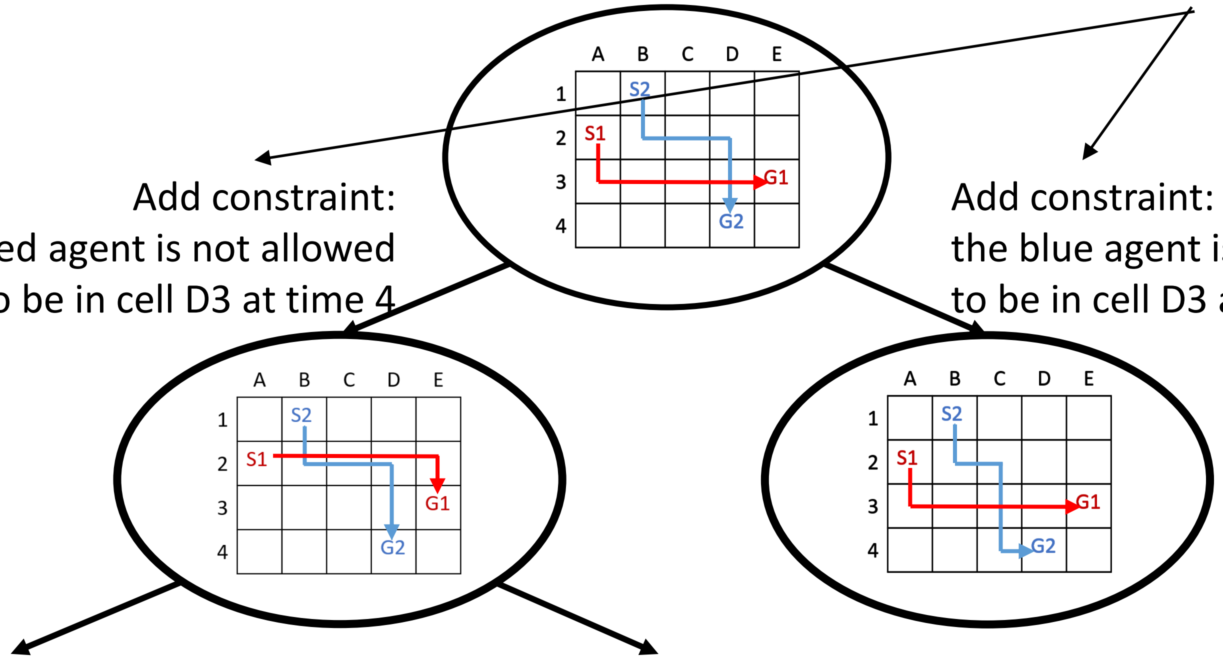
Conflict (here: vertex collision)

Conflict-Based Search (CBS)



- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

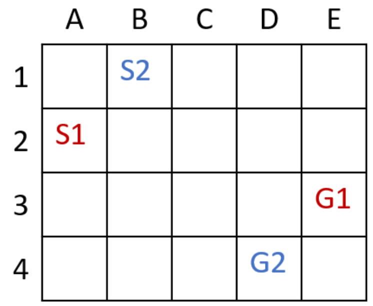
Such vertex constraints simply correspond to one blocked cell each in the reservation table



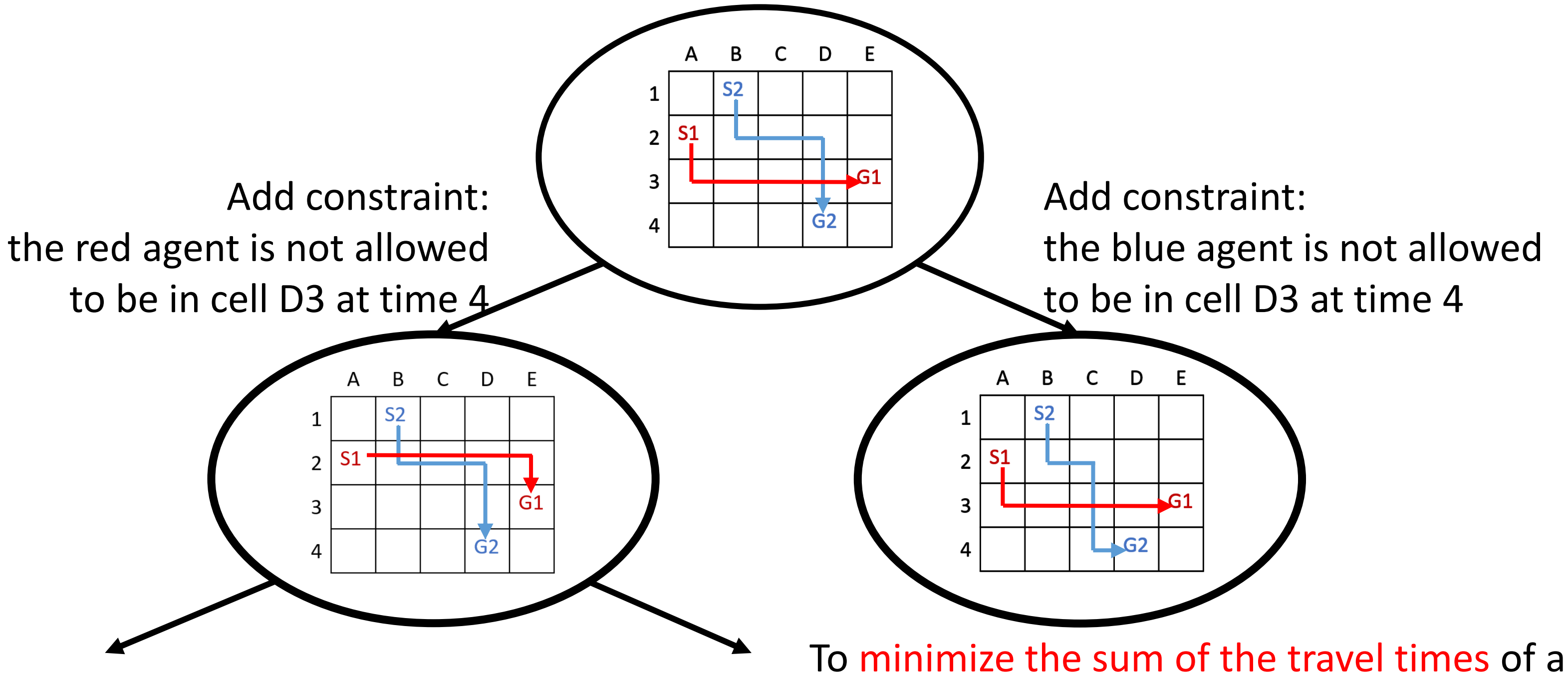
Add constraint:
the red agent is not allowed
to be in cell D3 at time 4

Add constraint:
the blue agent is not allowed
to be in cell D3 at time 4

Conflict-Based Search (CBS)

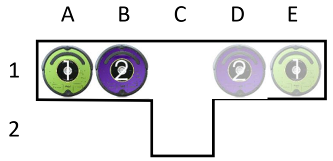


- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible



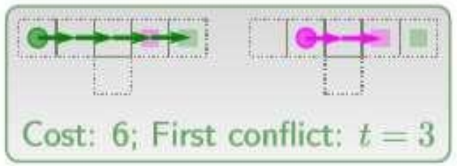
To minimize the sum of the travel times of all agents perform a best-first search on this tree with

- g = cost = sum of travel times of all agents (here: 10)
- h = 0



Conflict-Based Search (CBS)

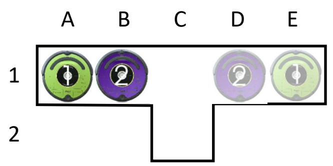
- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible



A1, B1, C1, D1, E1 B1, C1, D1

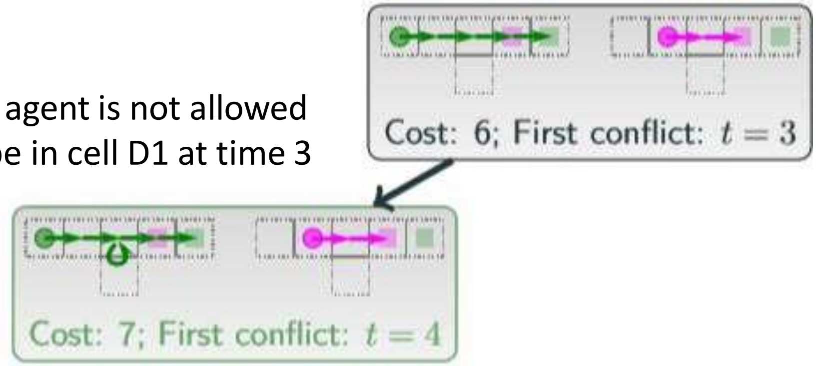
- Find time-minimal paths for both agents independently, which results in a vertex collision in cell D1 at time 3; clearly, the green agent cannot be in cell D1 at time 3 or the violet agent cannot be in cell D1 at time 3

Conflict-Based Search (CBS)



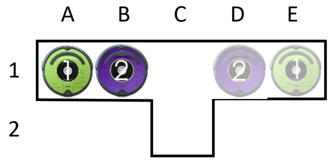
- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

the green agent is not allowed
to be in cell D1 at time 3



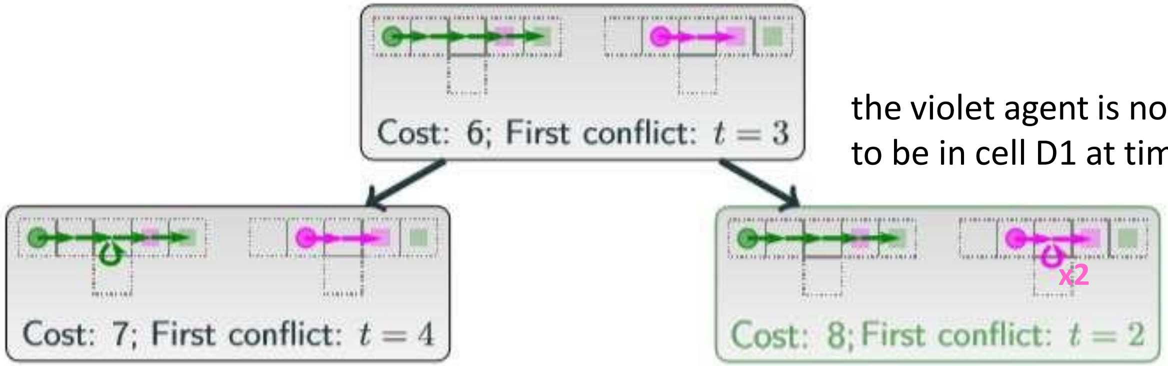
A1, B1, C1, C1 (= wait), D1, E1 B1, C1, D1

- Work on the leaf node with the smallest cost; impose the vertex constraint: the green agent is not allowed to be in cell D1 at time 3; create a new child node, and replan the path of the green agent, which results in a vertex collision in cell D1 at time 4



Conflict-Based Search (CBS)

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

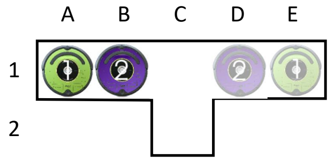


the violet agent is not allowed to be in cell D1 at time 3

A1, B1, C1, D1, E1   B1, C1, C1 (= wait), C1 (=wait), D1

- Impose also the vertex constraint: the violet agent is not allowed to be in cell D1 at time 3, create a new child node, and replan the path of the violet agent, which results in a vertex collision in cell C1 at time 2

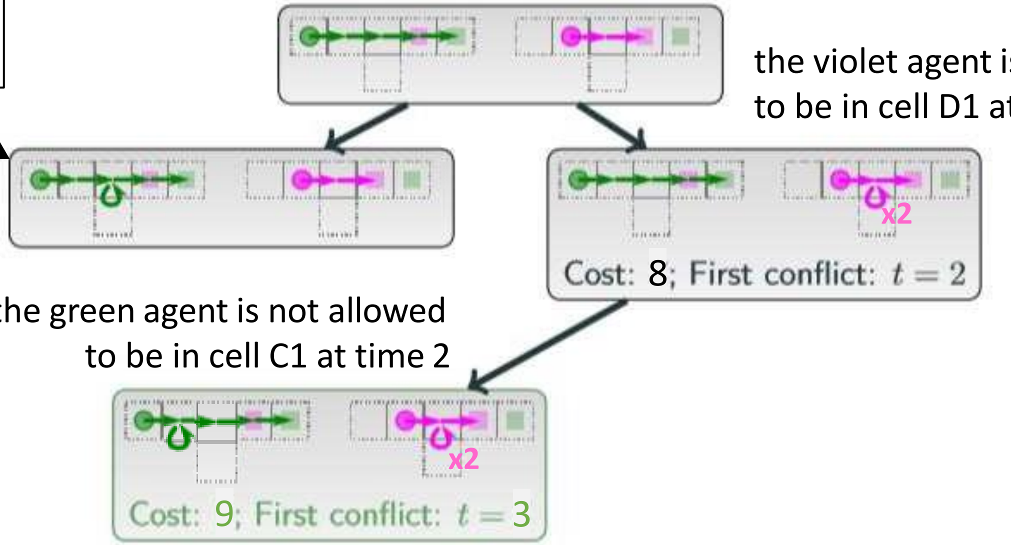
Conflict-Based Search (CBS)



- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

we omitted this node expansion, which results in child nodes with cost no smaller than 8

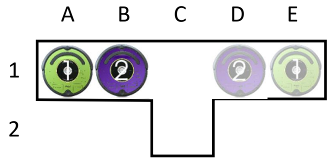
the violet agent is not allowed to be in cell D1 at time 3



the green agent is not allowed to be in cell C1 at time 2

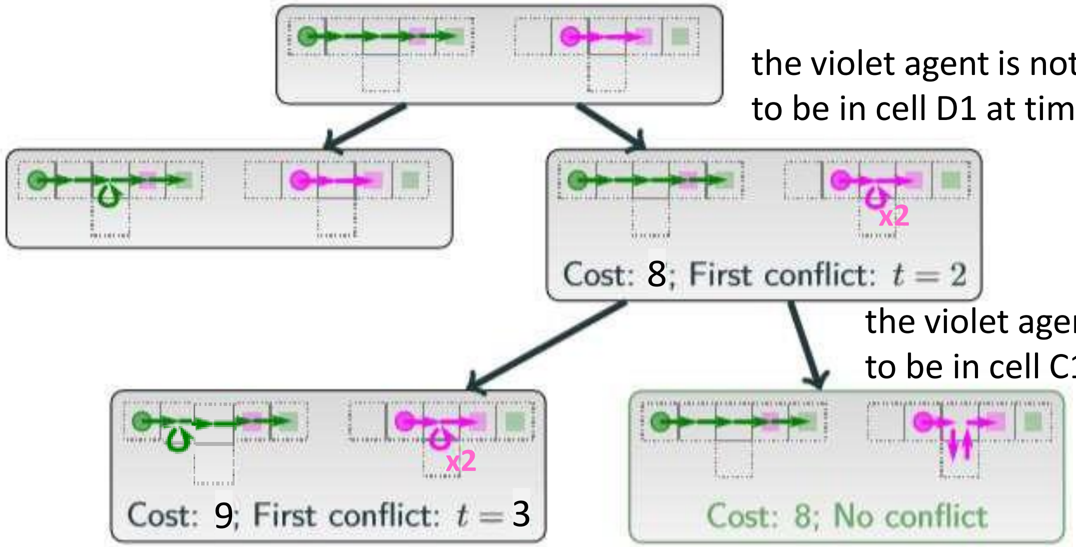
A1, B1, B1 (=wait), C1, D1, E1 B1, C1, C1 (= wait), C1 (=wait), D1

- Work on the leaf node with the smallest cost; impose the vertex constraint: the green agent is not allowed to be in cell C1 at time 2 (in addition to the previous vertex constraint), create a new child new, and replan the path of the green agent, which results in a vertex collision in cell C1 at time 3



Conflict-Based Search (CBS)

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

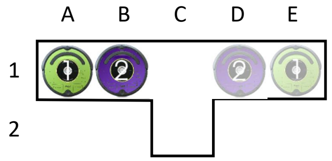


the violet agent is not allowed to be in cell D1 at time 3

the violet agent is not allowed to be in cell C1 at time 2

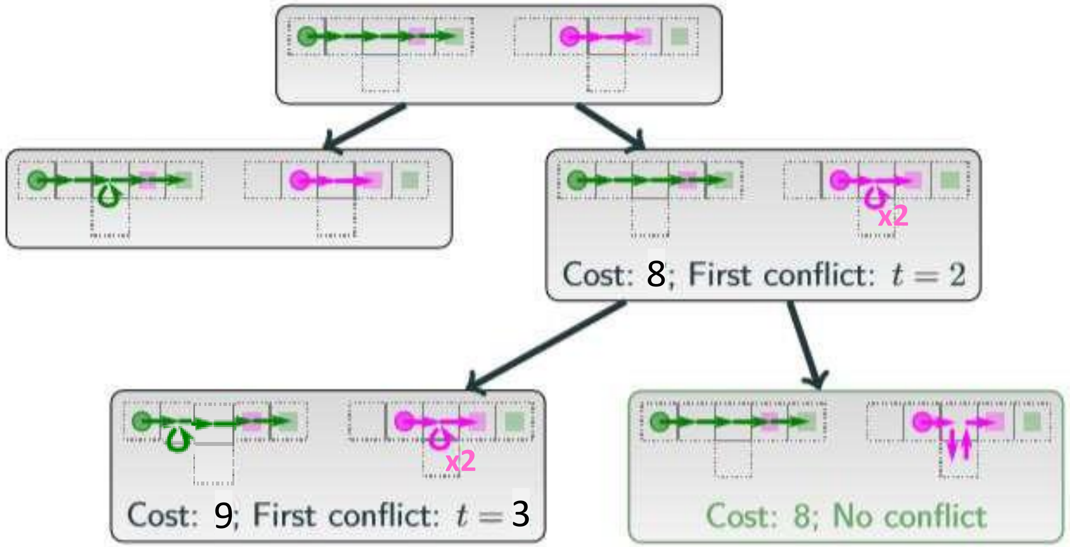
A1, B1, C1, D1, E1   B1, C1, C2, C1, D1

- Impose also the vertex constraint: the violet agent is not allowed to be in cell C1 at time 2 (in additional to the previous vertex constraint), work on the child node with the smallest cost, and replan the path of the violet agent, which results in no vertex or edge collisions



Conflict-Based Search (CBS)

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible



A1, B1, C1, D1, E1 B1, C1, C2, C1, D1

(re-)planning good paths matters:
we could have saved node expansions if we planned this path for the violet agent when generating the parent node

- Work on the leaf node with the smallest cost and terminate since this node has no vertex or edge collisions

Solution Quality of Conflict-Based Search (CBS)

Definition 1. For a given node N in the search tree, let CV(N) be the set of all plans that are: (1) consistent with the set of constraints of N and (2) are also valid (i.e., without conflicts).

Definition 2. We say that node N permits a plan p if and only if p $\in$CV(N).

Solution Quality of Conflict-Based Search (CBS)

Lemma 1. The cost of a node N in the search tree is a lower bound on minCost(CV(N)).

Proof. Since *N.cost* is the sum of individually optimal consistent paths of all agents, it has the minimum cost among all consistent plans. By contrast, *minCost*(*CV*(*N*)) has the minimum cost among all consistent and valid plans. Since the set of all consistent and valid plans is a subset of all consistent plans, it must be that *N.cost*≤*minCost*(*CV*(*N*)).

Solution Quality of Conflict-Based Search (CBS)

Lemma 2. For each valid plan p, there exists at least one node N in OPEN that permits p.

Proof. By induction: Base case: OPEN only contains the root node, which has no constraints. Consequently, the root node *permits* all valid plans. Now, assume this is true after the first $i-1$ expansions. During expansion i assume that node N is expanded. The successors of node N–$N1$ and $N2$ are generated. Let p be a valid plan. If p is permitted by another node in OPEN, we are done. Otherwise, assume p is permitted by N. We need to show that p must be permitted by at least one of its successors. The new constraints for $N1$ and $N2$ share the same t and x but constrain different agents. Suppose a plan p permitted by N has agent $a1$ in location x at time t. Agent $a1$ can only be constrained at one of $N1$ and $N2$, but not both, so one of these nodes must permit p. Thus, the induction holds.

Solution Quality of Conflict-Based Search (CBS)

Theorem 1. The solution returned by CBS is optimal.

Proof. When a goal node *Ng* is chosen for expansion by the high level, all valid plans are *permitted* by at least one node from OPEN(Lemma2). Let *p* be a valid plan (with cost *p*.*cost*) and let *N*(*p*) be the node that *permits p* in OPEN. Let *N.cost* be the cost of node *N*. *N*(*p*).*cost*≤*p*.*cost* (Lemma1). Since *Ng* is a goal node, *Ng*.*cost* is a cost of a valid plan. Since the high-level search explores nodes in a best-first manner according to there cost we get that *Ng*.*cost*≤*N*(*p*).*cost*≤*p*.*cost*.

Termination

Theorem 2. For every cost C, there is a finite number of nodes with cost C.

Proof. Assume a node N with cost C. After time step C all agents are at their goal position. Consequently, no conflicts can occur after time step C. Since constraints are derived from conflicts, no constraints are generated for time steps greater than C. As the cost of every node is monotonically non-decreasing, all of the predecessors of the node N have cost$\leq C$. Hence, neither N nor any of its predecessors can generate constraints for time step greater than C. Since there is a finite number of such constraints (at most $k \cdot |V| \cdot C$ constraints on vertices and $k \cdot |E| \cdot C$ constraints on edges), there is also a finite number of CT nodes that contain such constraints.

Termination

Theorem 3. CBS returns a solution if one exists.

Proof. CBS performs a best-first search, and the costs of the nodes are monotonically non-decreasing. Therefore, for every pair of costs X and Y, if $X<Y$ then CBS expands all nodes with cost X before expanding nodes of cost Y. Since for each cost there is a finite number of nodes (Theorem2), then the optimal solution must be found after expanding a finite number of nodes.

Termination

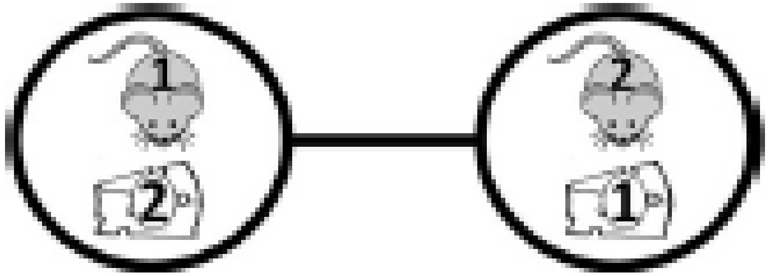
Without an upper bound on the optimal cost, CBS does not identify an unsolvable instance.

Why?

Termination

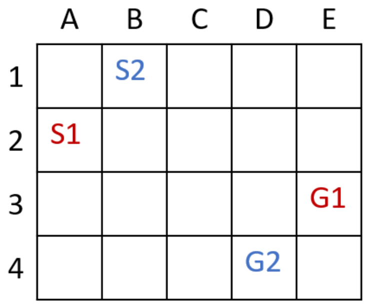
Without an upper bound on the optimal cost, CBS does not identify an unsolvable instance.

Why?

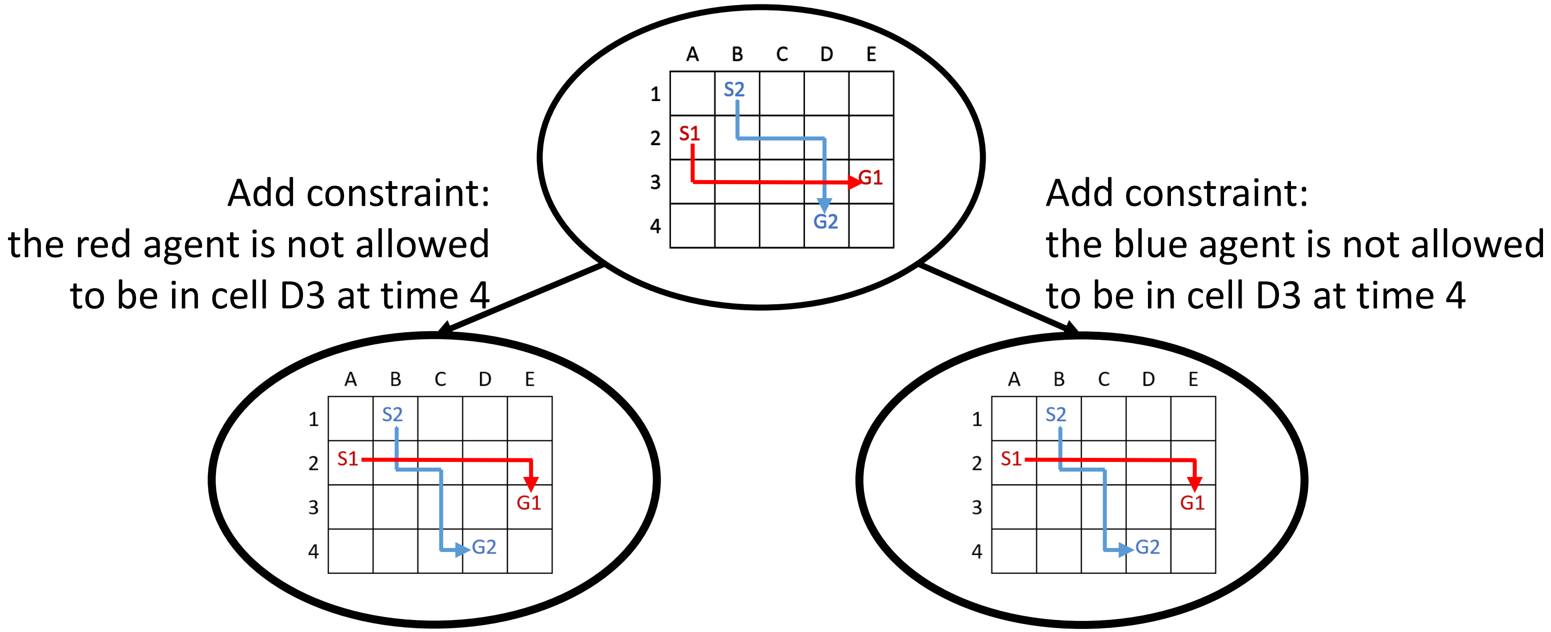


[Yu and Rus] provides an $O(|V|^3)$ bound for a graph with $|V|$ vertices.

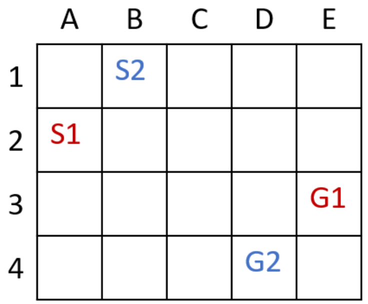
Conflict-Based Search with Disjoint Splitting



- Conflict-based search (without disjoint splitting) [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible



Conflict-Based Search with Disjoint Splitting

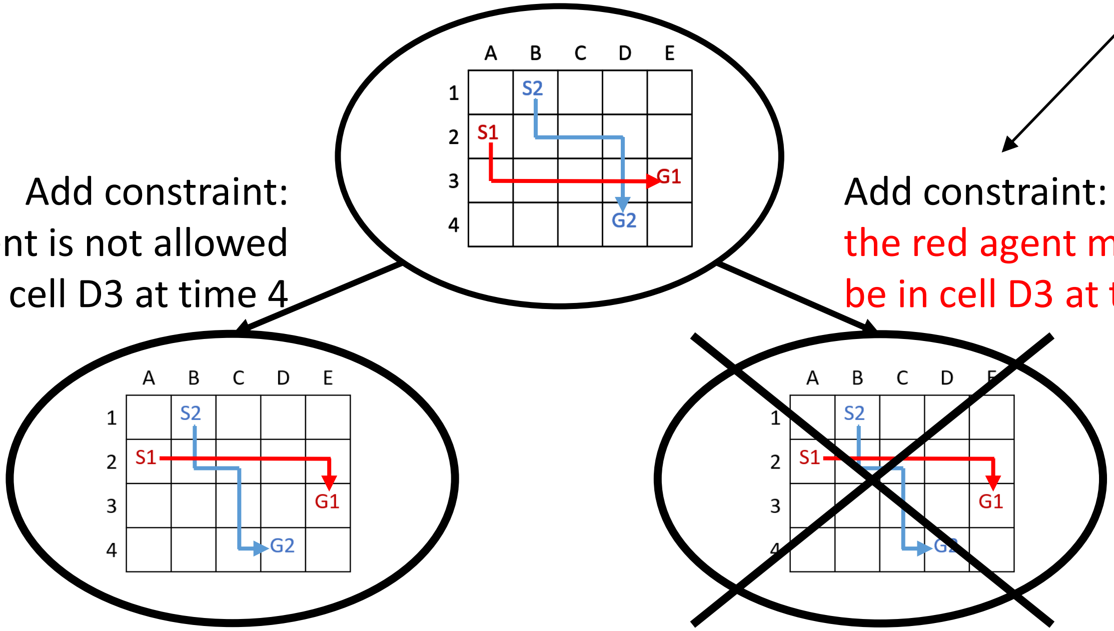


- Conflict-based search with disjoint splitting [Li, Harabor, Stuckey, Felner, Ma and Koenig]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

This implies that all other agents are not allowed to be in cell D3 at time 4 – so the constraint is stricter than before!

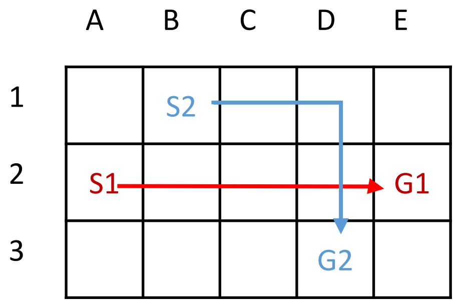
Add constraint:
the red agent is not allowed to be in cell D3 at time 4

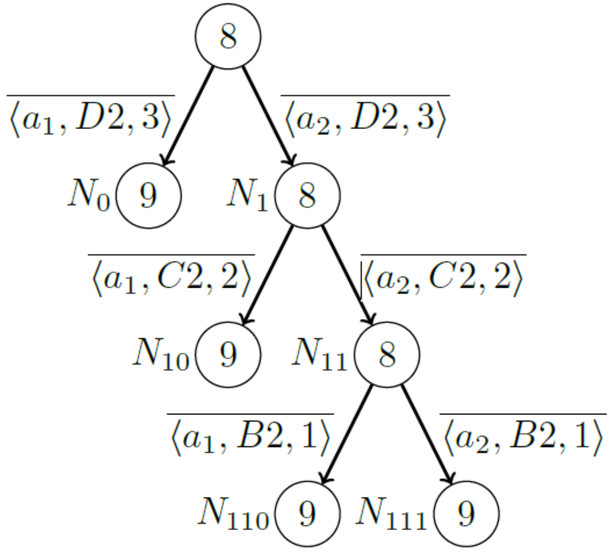
Add constraint:
the red agent must be in cell D3 at time 4



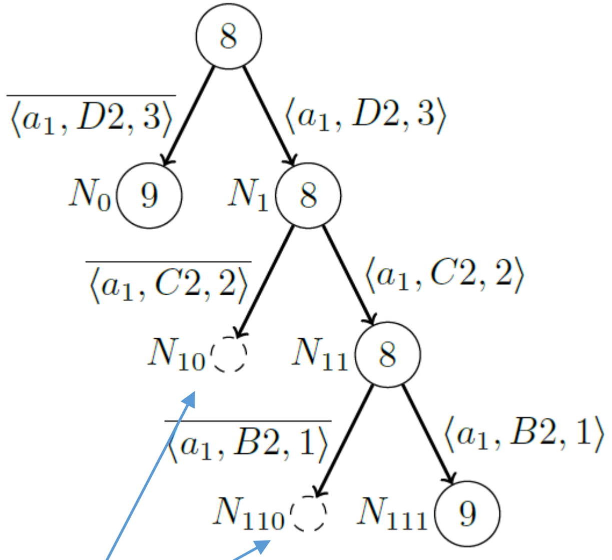
Conflict-Based Search with Disjoint Splitting

- Conflict-based search with disjoint splitting: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible





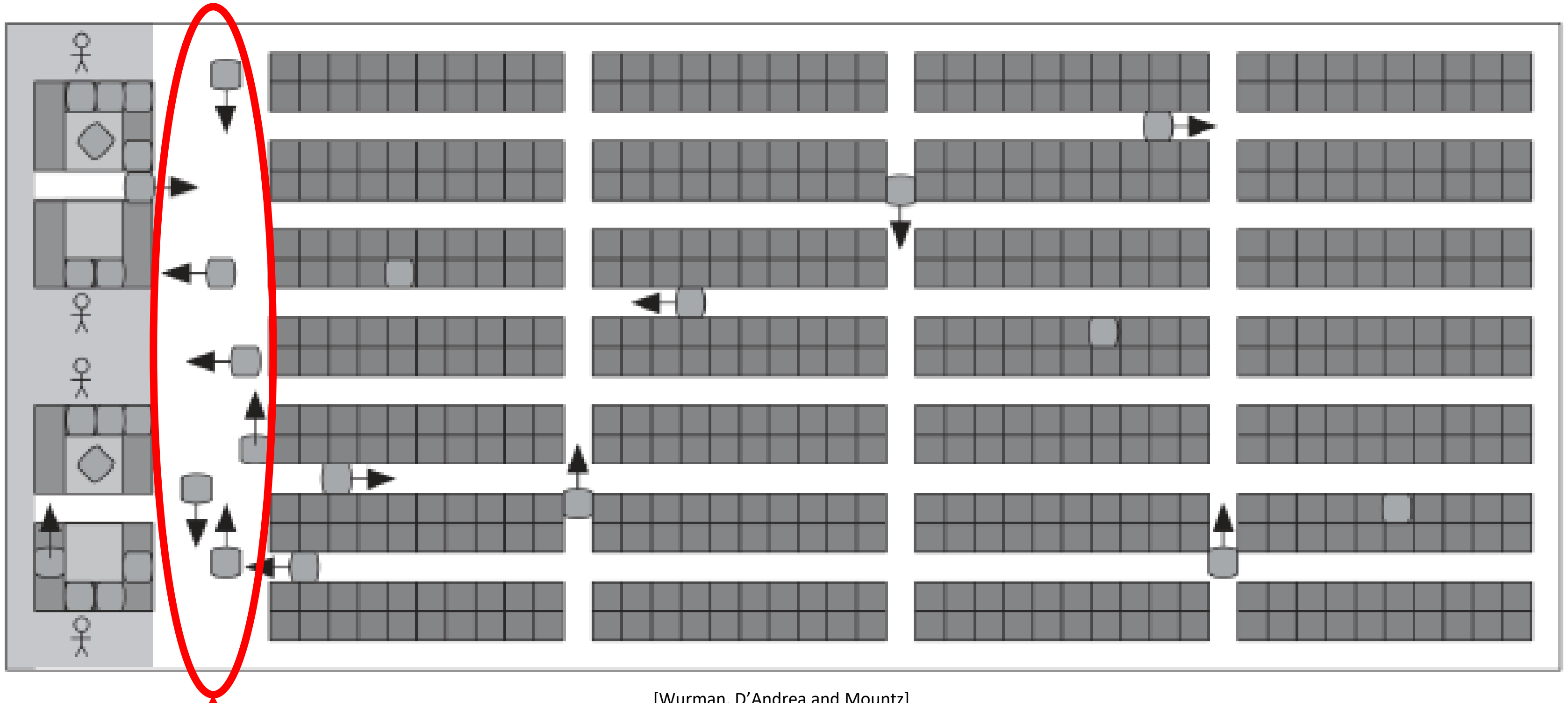
Conflict-based search without disjoint splitting



Pruned

Conflict-based search with disjoint splitting

Execution of MAPF Plans



[Wurman, D'Andrea and Mountz]

Use the MAPF methods here (in a small area of high congestion but with few agents) rather than over the whole fulfillment center

Summary

- Want to learn more about MAPF?
- Visit: http://mapf.info/