Warm-up exercise: Introduction to Multi-Agent Path Finding

Acknowledgement: The exercise is provided by University of Southern California (prof. Sven Koenig and his group)

Practical sessions

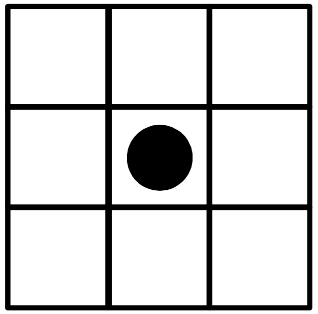
- Warm-up exercise should be done **individually;** only code needs to be delivered via Brightspace. The delivery of a solution to the warm-up exercise is required before starting the main assignment. The warm-up exercise will be graded as pass/fail
- The deadline for the warm-up exercise is 23rd of September
- The main assignment will be done **in pairs**; we decided to couple you randomly taking into account your Bachelor education
- Most of the questions should be asked during practicums
- Consultation hours: Thursdays at 12:30-13:30

If you have questions about your code, please send them to TAs on Wednesdays latest!

Multi-Agent Path Finding (MAPF)

Robot

Agent



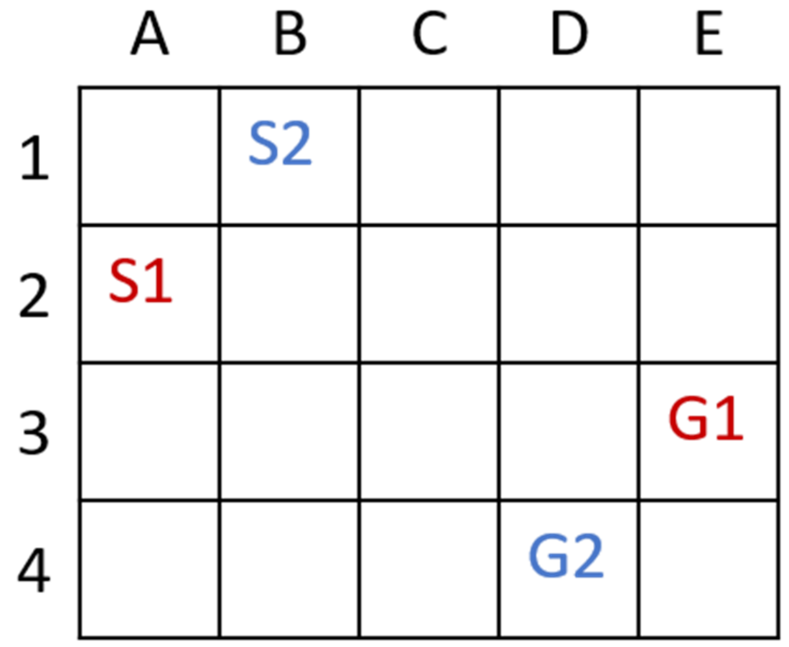
[from: YouTube]

Stickers on the ground establish a grid!

amazon

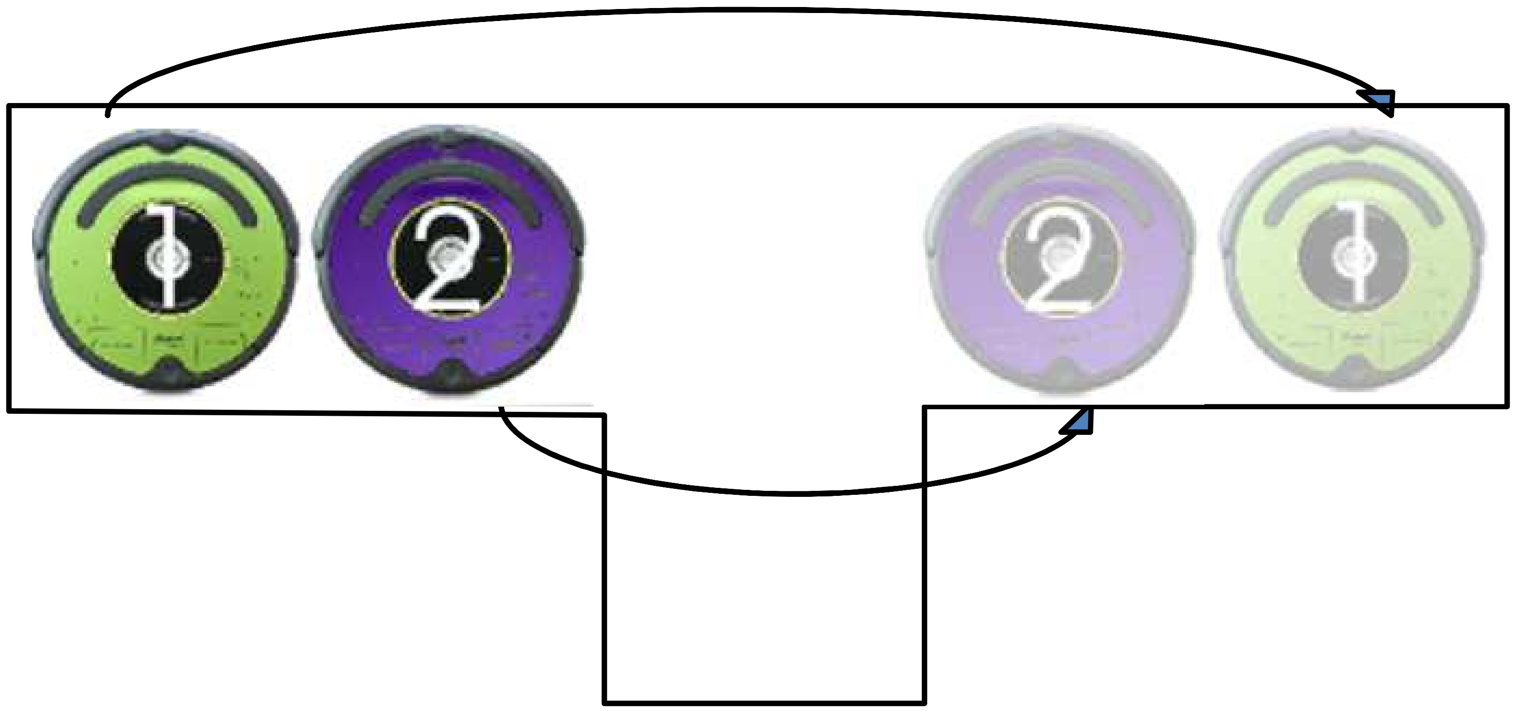
- Simplifying assumptions
  - Point agents
  - No kinematic constraints
  - Discretized environment
    - we use grids here but the techniques work on planar graphs in general

Multi-Agent Path Finding (MAPF)

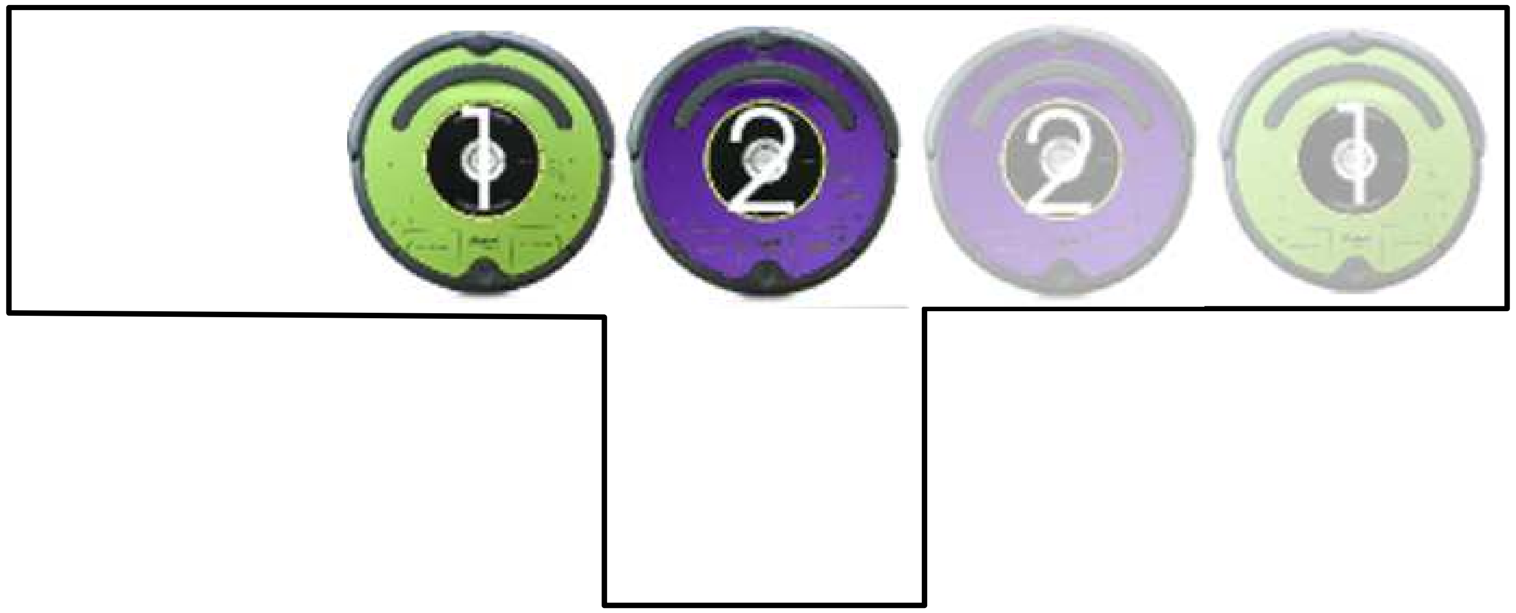


S1 (S2) = start cell of the red (blue) agent
G1 (G2) = goal cell of the red (blue) agent

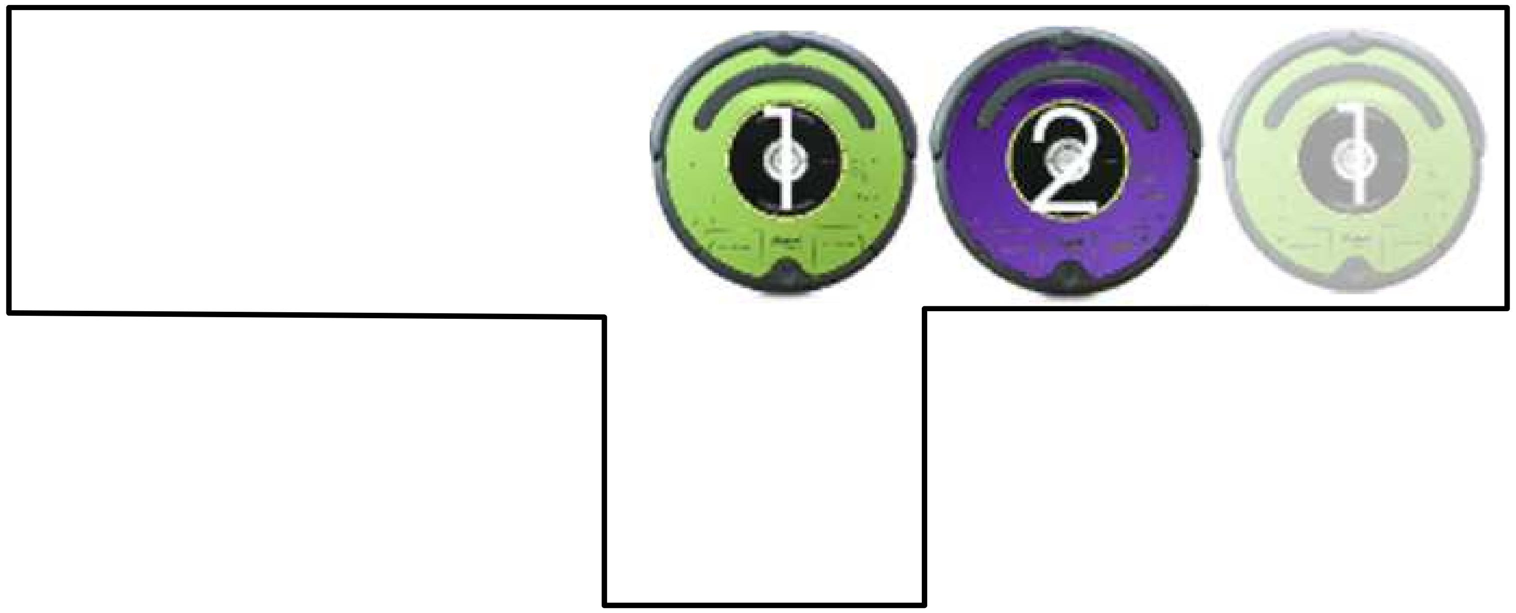
Multi-Agent Path Finding (MAPF)



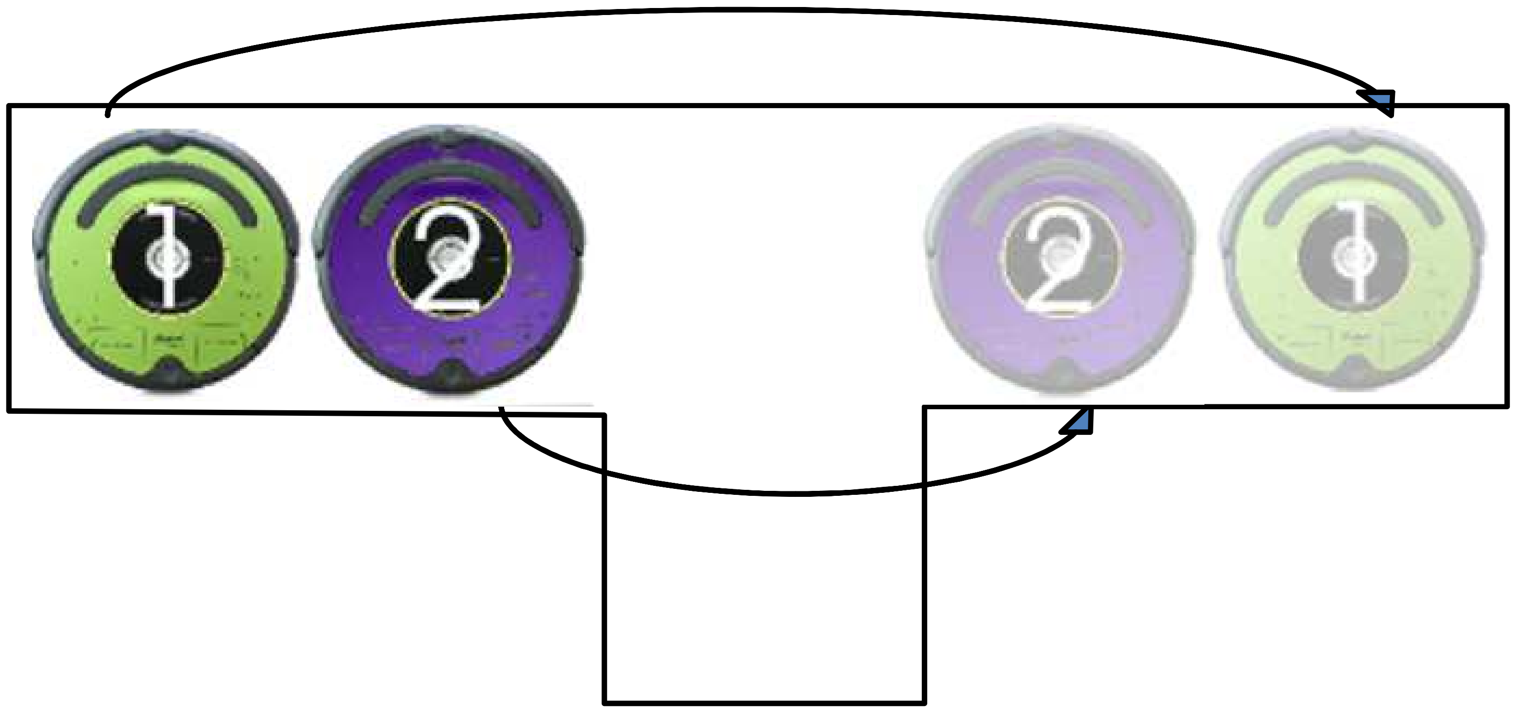
Multi-Agent Path Finding (MAPF)



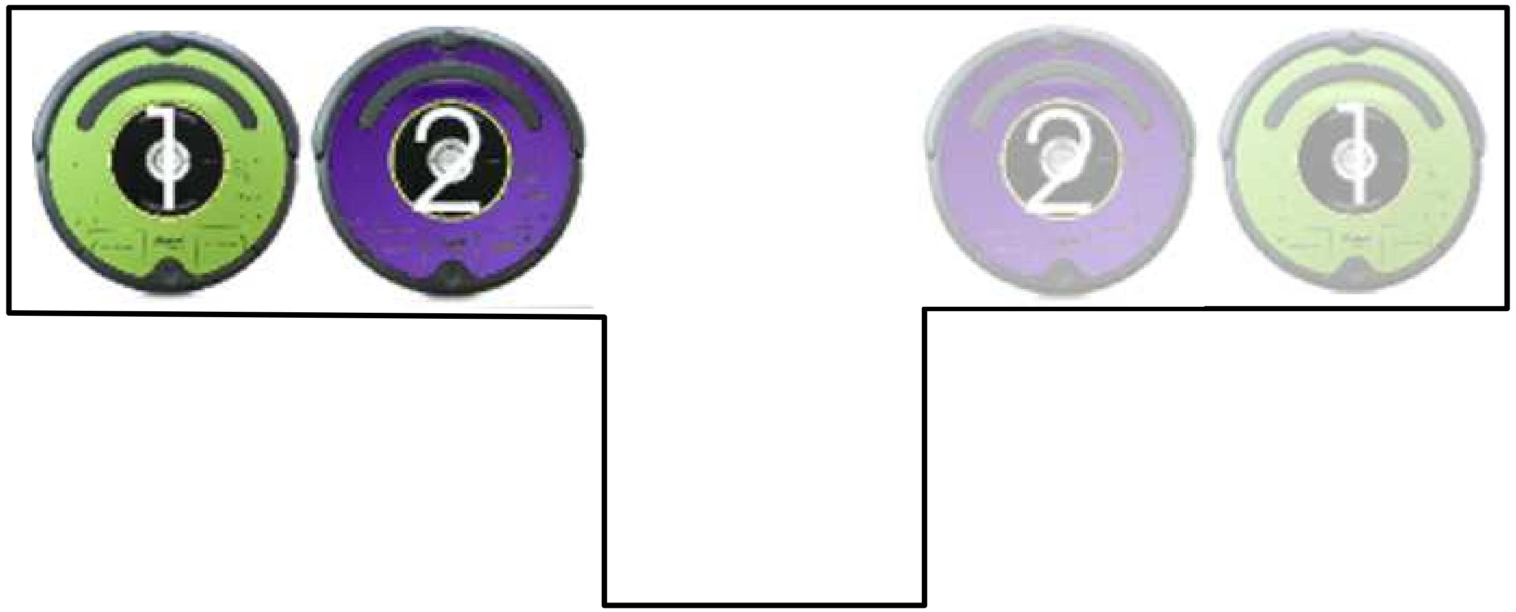
Multi-Agent Path Finding (MAPF)



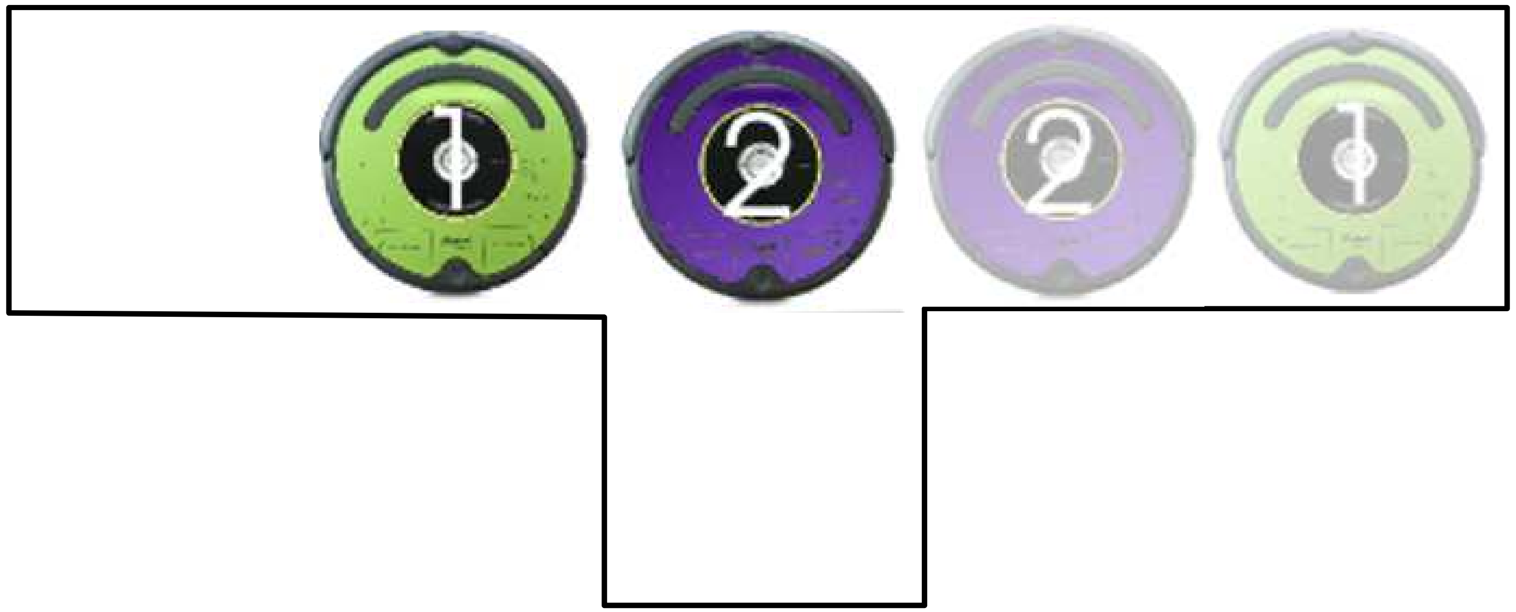
Multi-Agent Path Finding (MAPF)



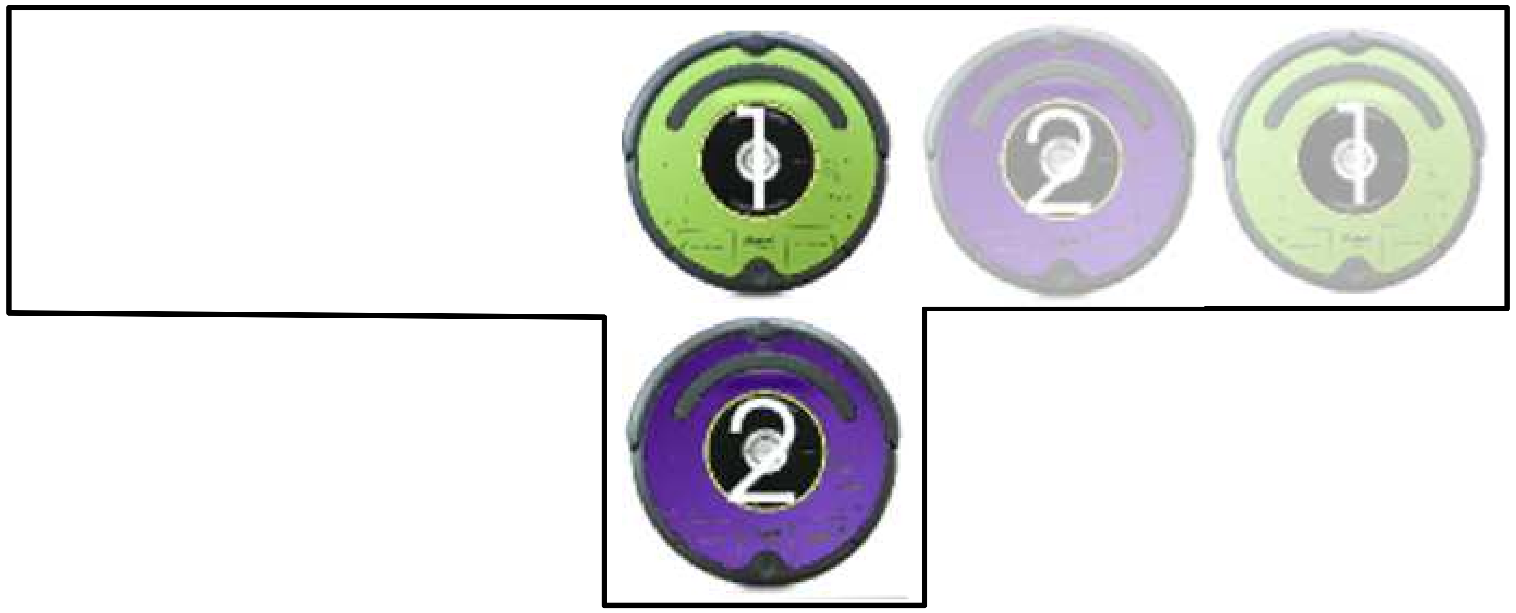
Multi-Agent Path Finding (MAPF)



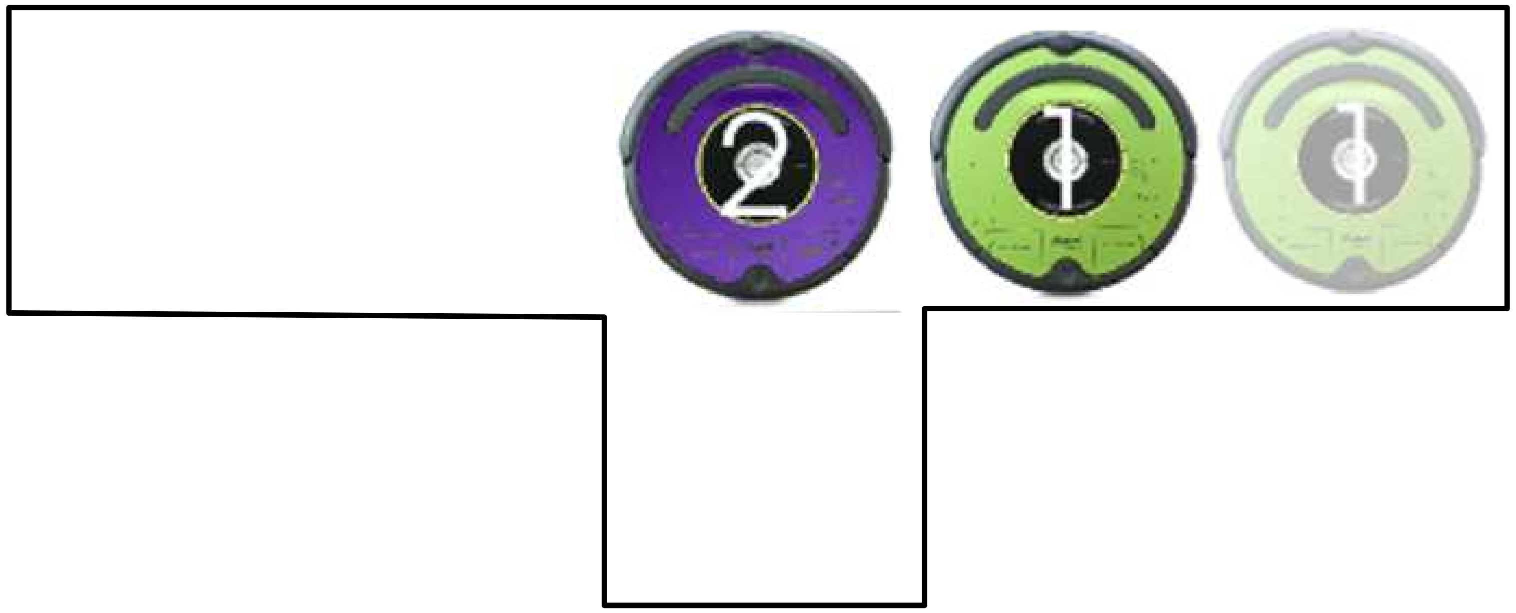
Multi-Agent Path Finding (MAPF)



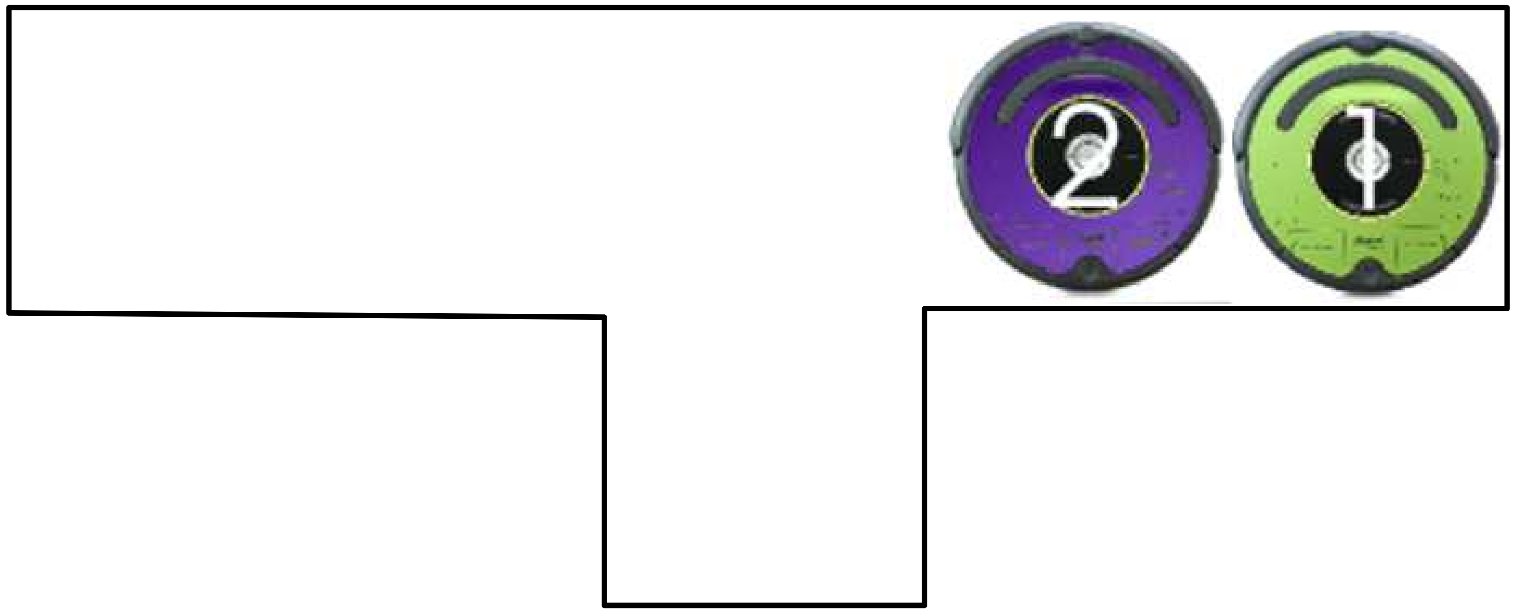
Multi-Agent Path Finding (MAPF)



Multi-Agent Path Finding (MAPF)



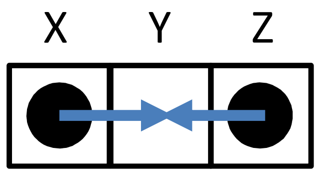
Multi-Agent Path Finding (MAPF)

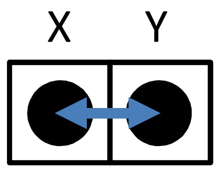


- Optimization problem with the objective
  to minimize task-completion time (called makespan) or
  the sum of travel times (called flowtime)

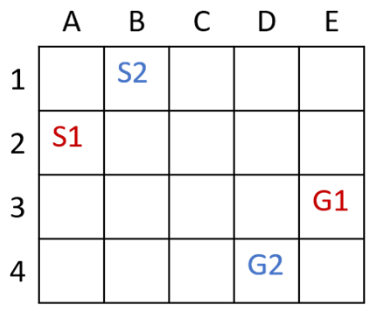
Multi-Agent Path Finding (MAPF)

- 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

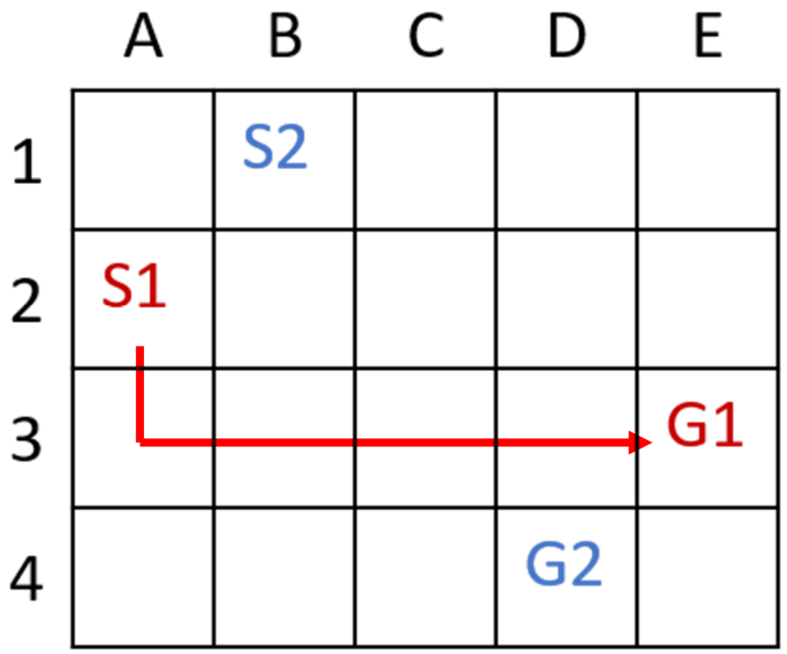




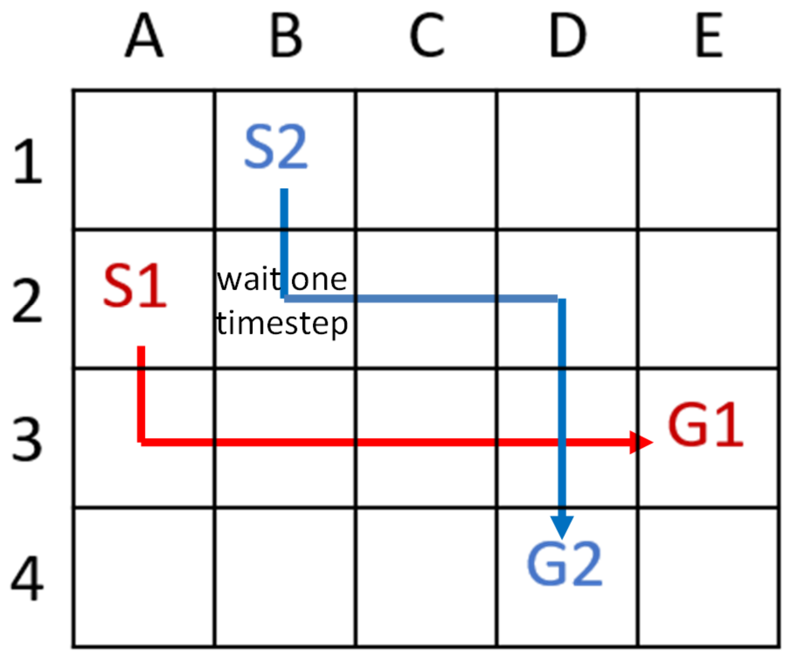
Priority-Based Search



- Priority-based (= 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

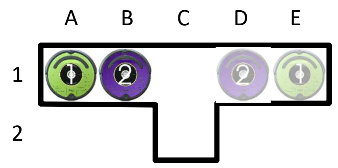


First, find a time-minimal path for the agent with priority 1.

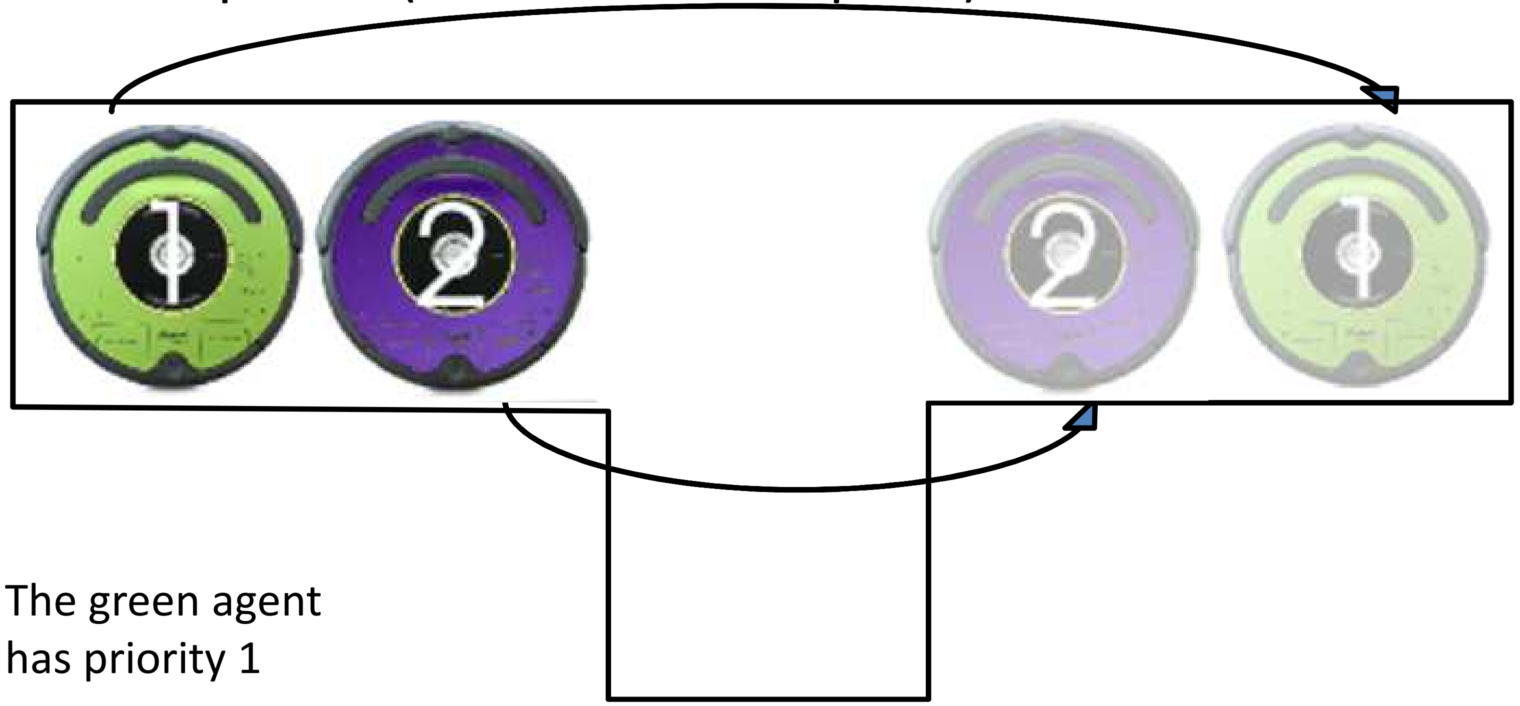


Then, find a time-minimal path for the agent with priority 2 that does not collide with the paths of higher-priority agents.

Priority-Based Search



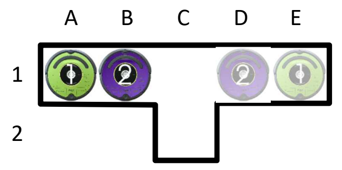
- Priority-based (= sequential) search (plan for one agent after another in space (= cell)-time space in a given order): efficient but suboptimal (and even incomplete) MAPF solver



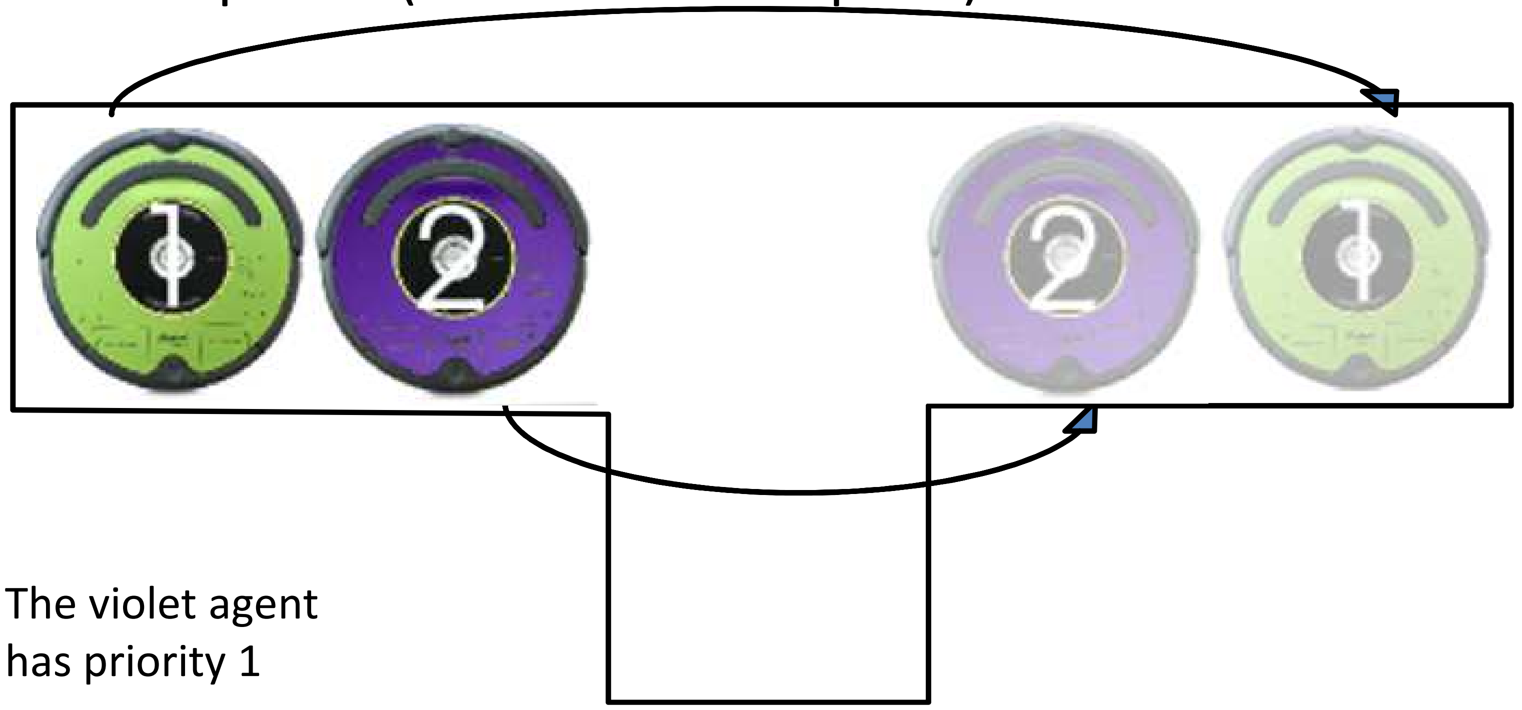
The green agent has priority 1

- Priority-based search 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, priority-based search finds a solution.

Priority-Based Search



- Priority-based (= 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

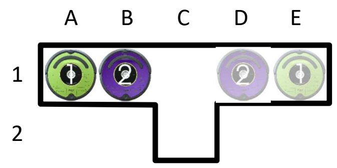
- Priority-based search finds first path B1, C1, D1 for the violet agent and then no path for the green agent. Thus, priority-based search does not find a solution.

Priority-Based Search

- 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

Priority-Based Search

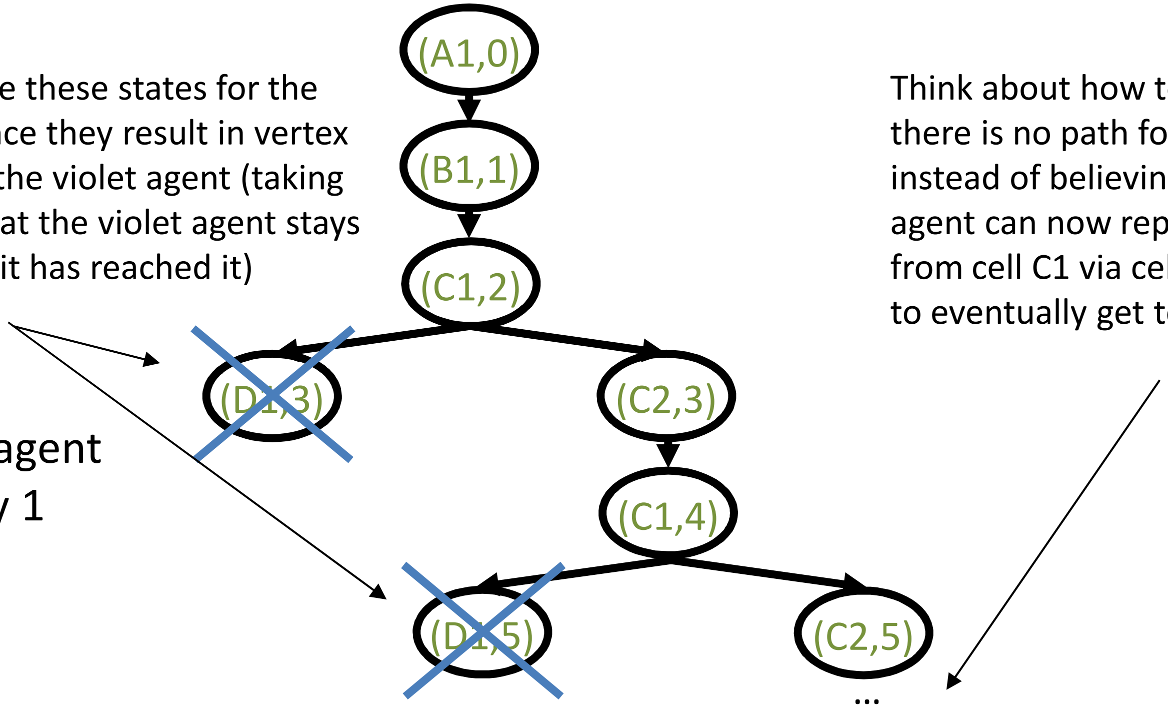


- You could implement space (= cell)-time A* with a reservation table (specific for a particular agent) but you might not want to build it explicitly since it is often large. Rather, you never want to generate the states or edges that you would have deleted in the reservation table in the A* search tree

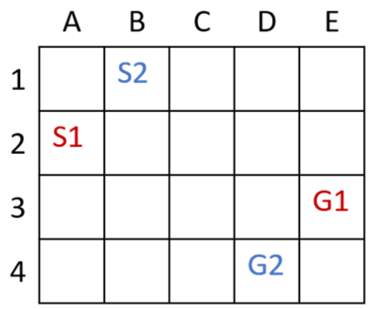
Do not generate these states for the green agent since they result in vertex collisions with the violet agent (taking into account that the violet agent stays in cell D3 once it has reached it)

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

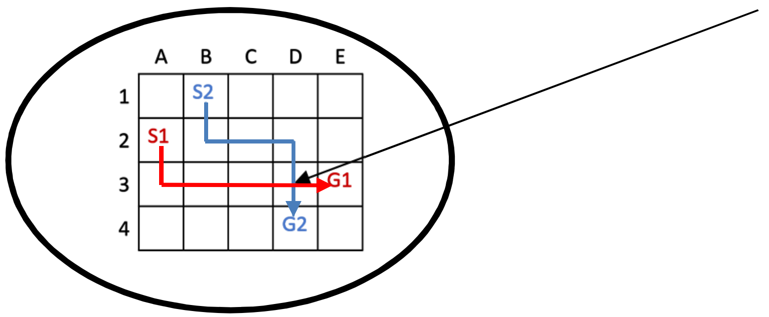


Conflict-Based Search



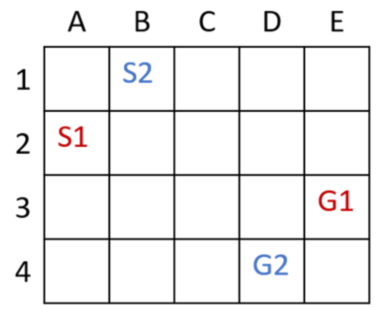
- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

Find time-minimal paths for all agents independently

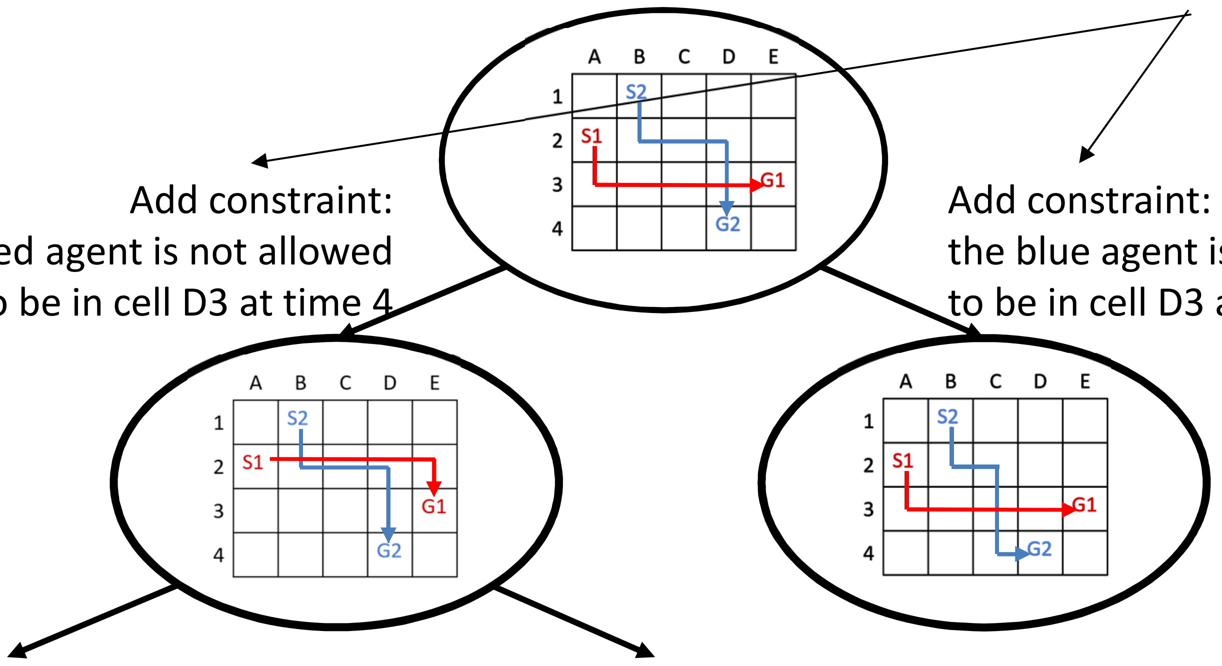


Conflict (here: vertex collision)

Conflict-Based Search



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

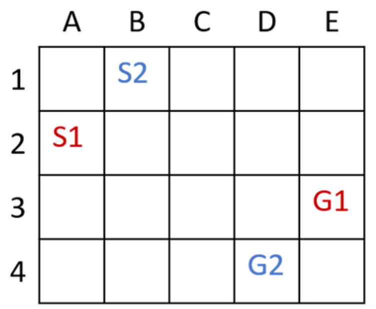


Such vertex constraints simply correspond to one blocked cell each in the reservation table

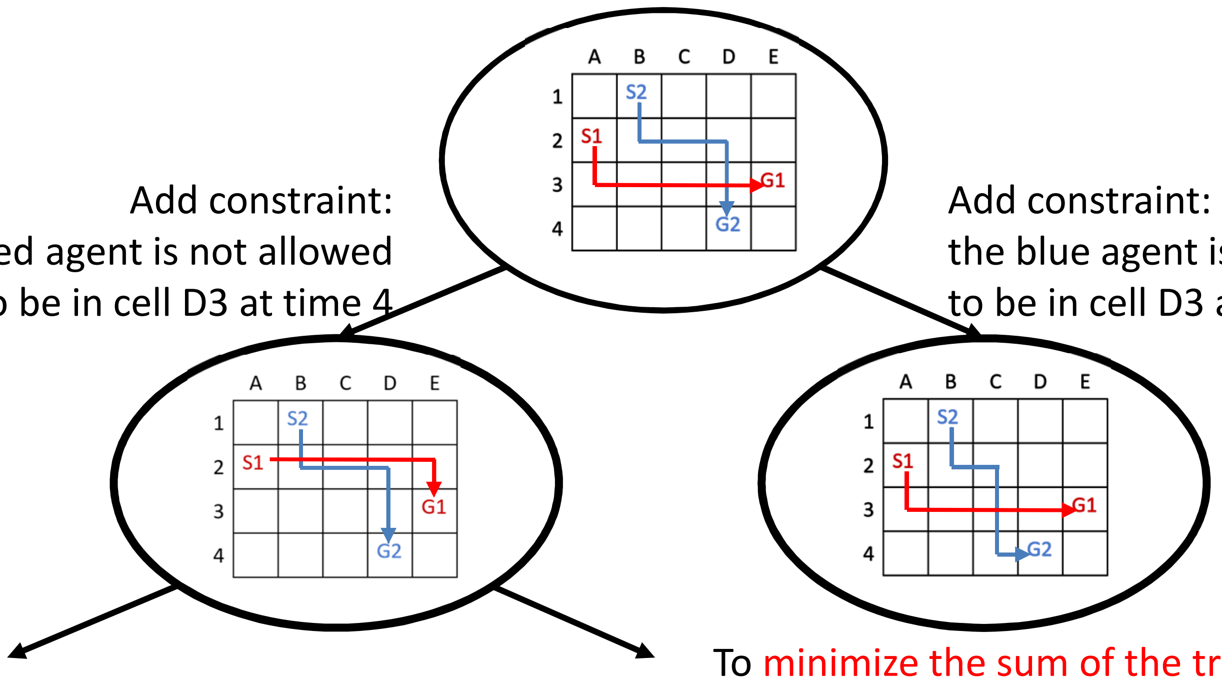
Add constraint:
the red agent is not allowed to be in cell D3 at time 4

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

Conflict-Based Search



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



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

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