02285 AI and MAS, F23
Conflict based search

Todays subjects:

- Multi-agent pathfinding (MAPF)
- Baseline algorithm: Multibody A*
- Explore three (and a half) other algorithms
- In particular, conflict based search

MAPF problem

Multi-agent pathfinding (MAPF) problem

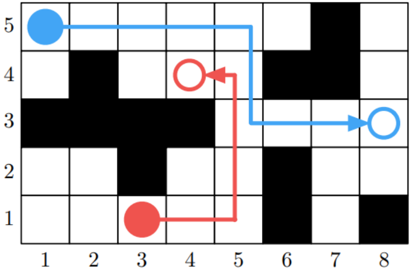
Given a problem instance

- a graph $G = (V, E)$, where vertices V of the graph are possible locations for the agents, and the edges E are the possible transitions between locations;
- a set $a_1, a_2, ..., a_k$ of k agents;
- a set $v_{1,0}, v_{2,0}, ..., v_{k,0}$ of start locations for each agent;
- a set $g_1, g_2, ..., g_k$ of goal locations for each agent;

A solution is a plan $\pi_i = (v_{i,0}, v_{i,1}, ..., g_i)$ for each agent a_i, i.e., a sequence of vertices that lead from the start to the goal.

MAPF problem

- Simpler than the hospital domain
  - No boxes
  - All agents have one goal
  - But considers general graphs, not only grids
- Today we will look at different algorithms for solving this problem



Solution 1: Multibody A*

- A* graph search algorithm in the space of joint actions, i.e., multibody A* (MBA*)
- Similar to what you did in the warmup assignment

Solution 1: Multibody A*

- A* graph search algorithm in the space of joint actions, i.e., multibody A* (MBA*)
- Similar to what you did in the warmup assignment
- Heuristic:

$$h(s) = \sum_{i=1}^{k} dist(v_{i,s}, g_i)$$

  where

  - $v_{i,s}$ is the vertex where agent a_i is located in state s, and
  - $dist : V \mapsto V$ is the shortest distance from any vertex to any other vertex (can be pre-computed, e.g., using the Floyd-Warshall algorithm in $O(|V|^3)$ time)

Solution 1: Multibody A*

- A* graph search algorithm in the space of joint actions, i.e., multibody A* (MBA*)
- Similar to what you did in the warmup assignment
- Heuristic:

$$h(s) = \sum_{i=1}^{k} dist(v_{i,s}, g_i)$$

  where

  - $v_{i,s}$ is the vertex where agent a_i is located in state s, and
  - $dist : V \mapsto V$ is the shortest distance from any vertex to any other vertex (can be pre-computed, e.g., using the Floyd-Warshall algorithm in $O(|V|^3)$ time)
- MBA* is complete (finds a solution if there is one)

Solution 1: Multibody A*

- A* graph search algorithm in the space of joint actions, i.e., multibody A* (MBA*)
- Similar to what you did in the warmup assignment
- Heuristic:

$$h(s) = \sum_{i=1}^{k} dist(v_{i,s}, g_i)$$

  where

  - $v_{i,s}$ is the vertex where agent a_i is located in state s, and
  - $dist : V \mapsto V$ is the shortest distance from any vertex to any other vertex (can be pre-computed, e.g., using the Floyd-Warshall algorithm in $O(|V|^3)$ time)
- MBA* is complete (finds a solution if there is one)
- MBA* is optimal (finds the shortest solution if heuristic is admissible)

Solution 1: Multibody A*

- Branching factor is b^k where b is the branching factor of one agent

Solution 1: Multibody A*

- Branching factor is b^k where b is the branching factor of one agent
- We say the solution depth is d if there is a (shortest) sequence of d joint actions that lead all agent to their respective goals

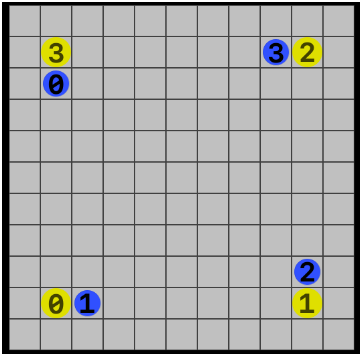
Solution 1: Multibody A*

- Branching factor is b^k where b is the branching factor of one agent
- We say the solution depth is d if there is a (shortest) sequence of d joint actions that lead all agent to their respective goals
- Worst case: We have to check all search tree branches of length d, i.e., expand b^{kd} states

Solution 1: Multibody A*

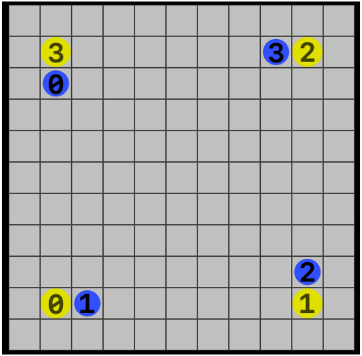
- Branching factor is b^k where b is the branching factor of one agent
- We say the solution depth is d if there is a (shortest) sequence of d joint actions that lead all agent to their respective goals
- Worst case: We have to check all search tree branches of length d, i.e., expand b^{kd} states
- Time/space complexity: $O(b^{kd})$ (upper bound)

Solution 1: Multibody A*

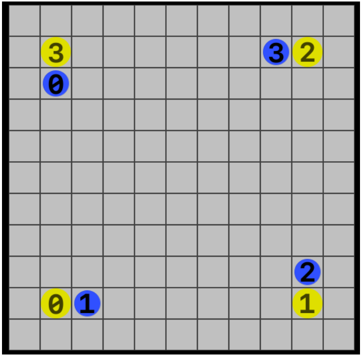


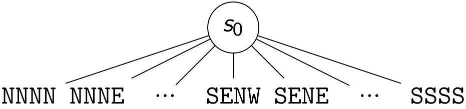
- In a grid domain $b = 5$: Move(N), Move(E), Move(S), Move(W), Noop
- $k = 4$ agent
- Solution depth $d = 7$
- Heuristic guides search perfectly!

Solution 1: Multibody A*

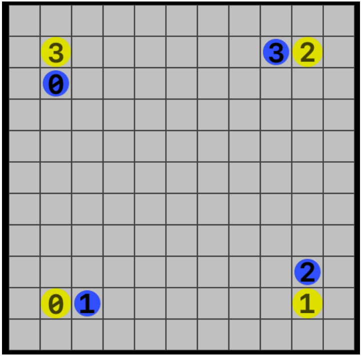


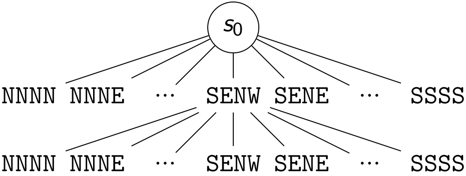
Solution 1: Multibody A*



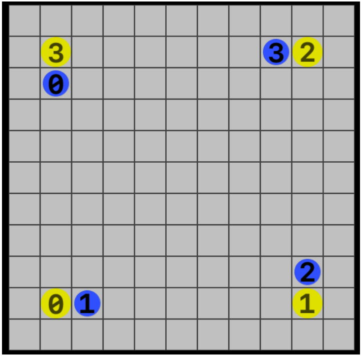


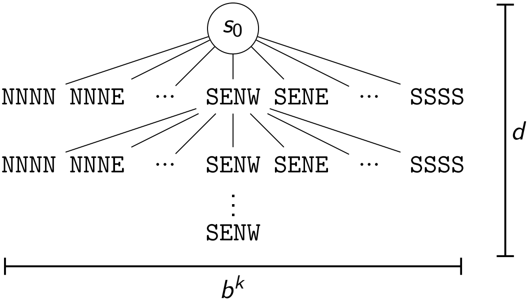
Solution 1: Multibody A*



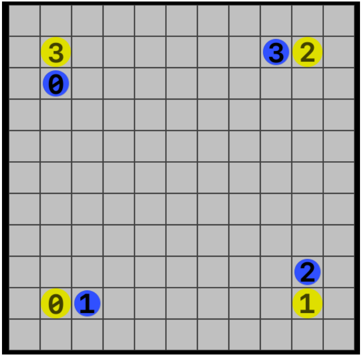


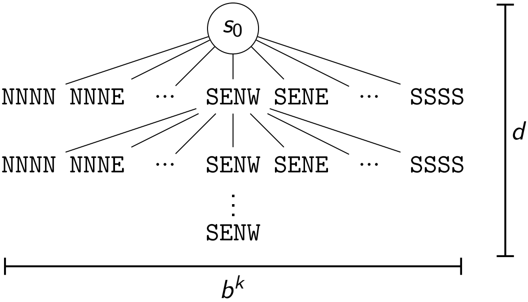
Solution 1: Multibody A*





Solution 1: Multibody A*

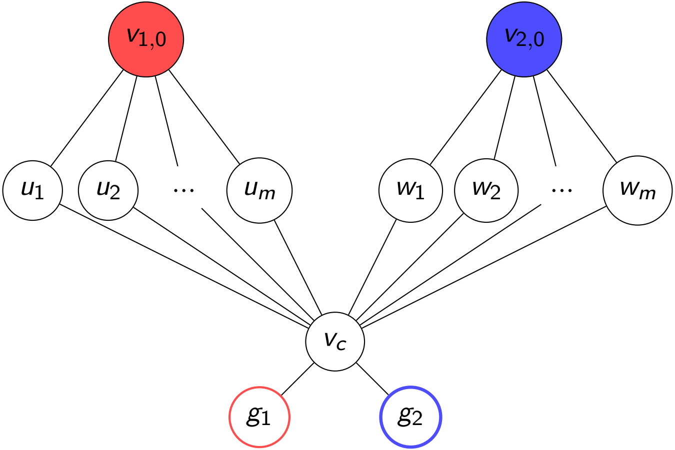




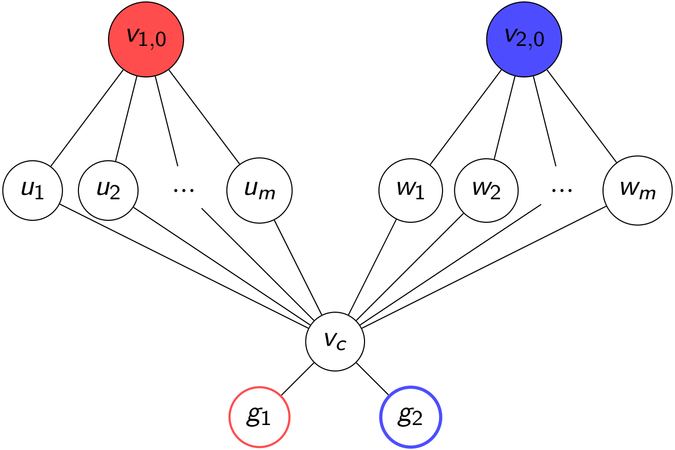
We expand $d = 7$ states, but generate $db^k = 7 \times 5^4 \approx 4300$ states(!)

If each agent could plan for itself, we would generate only $dbk = 7 \times 5 \times 4 = 140$ states

Solution 1: Multibody A*



Solution 1: Multibody A*



Worst case: All search tree paths branches of length 3 are expanded before we find the solution of length 4 (one agent has to wait), i.e., all $m^2 = b^k$ nodes are expanded.

Solution 2: ???

- Multibody vs multi-agent planning: Simplicity vs efficiency (avoiding exponential blowup)

Solution 2: ???

- Multibody vs multi-agent planning: Simplicity vs efficiency (avoiding exponential blowup)
- "Best" case (heuritic guides search perfectly):
  - Reduce branching factor
  - Solve k problems of time/space complexity $O(db)$
  - ... instead of one problem of $O\left(db^k\right)$ (140 vs 4300)

Solution 2: ???

- Multibody vs multi-agent planning: Simplicity vs efficiency (avoiding exponential blowup)
- "Best" case (heuritic guides search perfectly):
  - Reduce branching factor
  - Solve k problems of time/space complexity $O(db)$
  - ... instead of one problem of $O\left(db^k\right)$ (140 vs 4300)
- Worst case:
  - Solve k problems of time/space complexity $O\left(b^d\right) + \text{overhead}$
  - ... instead of one problem of $O\left(b^{kd}\right)$

Solution 2: ???

- Multibody vs multi-agent planning: Simplicity vs efficiency (avoiding exponential blowup)
- "Best" case (heuritic guides search perfectly):
  - Reduce branching factor
  - Solve k problems of time/space complexity $O(db)$
  - ... instead of one problem of $O\left(db^k\right)$ (140 vs 4300)
- Worst case:
  - Solve k problems of time/space complexity $O\left(b^d\right) + \text{overhead}$
  - ... instead of one problem of $O\left(b^{kd}\right)$
- Can we use CDPS (cooperative distributed problem solving) for MAPF?
  1. Problem decomposition
  2. Subproblem solution
  3. Solution synthesis

Solution 2: ???

Discuss: How do we ensure the plans of each agent are compatible?

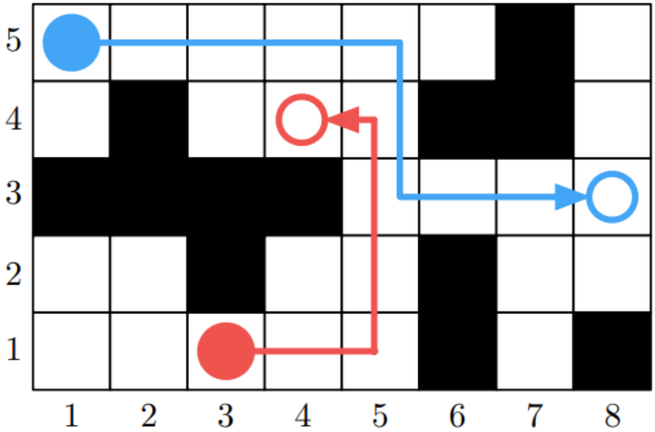


Illustration from August Møbius' thesis *Multi-Agent Pathfinding*

Solution 2: Space-time A*

Space-time A* (STA*): "A* with time constraints"

Definition

A constraint (a, v, t) precludes agent a from being in location $v \in V$ at time t

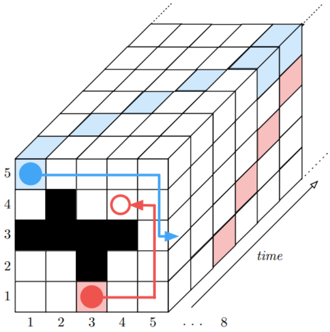


Illustration from August Møbius' thesis *Multi-Agent Pathfinding*

Solution 2: Space-time A*

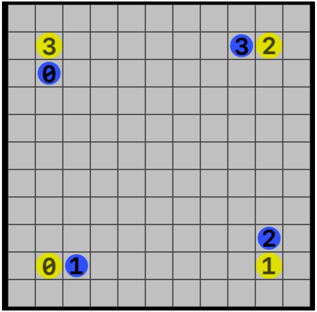
```
def SpaceTimeAStar(problem, agents):
    solution = dict()
    constraints = []
    k = len(agents)
    for i in range(k):
        plan = AStar(problem[agents[i]], constraints)
        constraints += [(agents[j], v, t)
                        for j in range(i+1, k)
                        for t, v in enumerate(plan)]
        solution[agent] = plan
    return solution
```

Assume a sub-routine `AStar`(p, c) with an admissible heuristic that can solve single-agent problem instances given

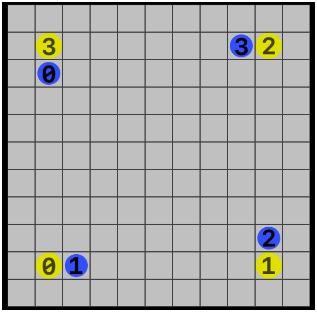
- a single-agent problem instance, p, and
- a (possibly empty) list of *constraints*, c.

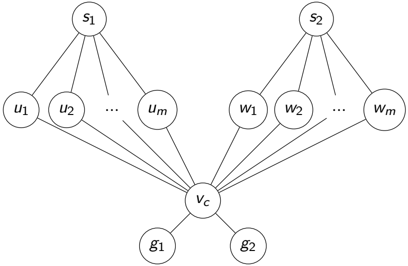
Here, the constraints c are the solutions of the previous agents

Solution 2: Space-time A*

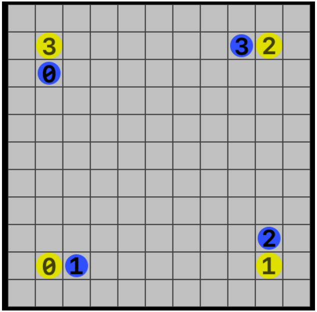


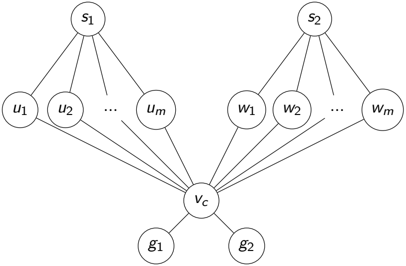
Solution 2: Space-time A*

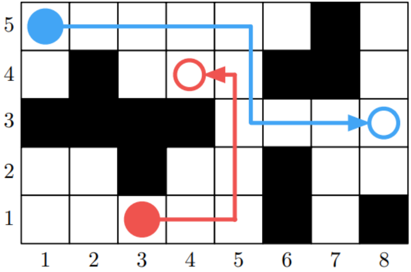




Solution 2: Space-time A*







Solution 2: Space-time A*

Questions:

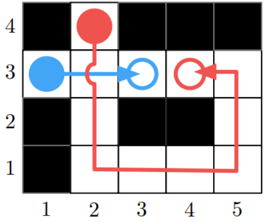
- Is it optimal? Can you give an example where we're not guaranteed to find the shortest solution?
- Is it complete? Can you give an example where we're not guaranteed to find a solution when there is one?

Solution 2: Space-time A*

Questions:

- Is it optimal? Can you give an example where we're not guaranteed to find the shortest solution?
- Is it complete? Can you give an example where we're not guaranteed to find a solution when there is one?

We have serialisable subgoals, but STA* doesn't choose subgoals intelligently



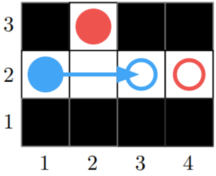


Illustration from August Møbius' thesis *Multi-Agent Pathfinding*

Solution 3: ...

Can we get the best of both worlds?

- Agents plan independently if they can
- Agents plan as a multibody if they conflict

Solution 3: ...

Can we get the best of both worlds?

- Agents plan independently if they can
- Agents plan as a multibody if they conflict

Definition

(a_i, a_j, v, t) is a conflict of agents a_i and a_j both planning to be in location $v \in V$ at time t.

Solution 3: ...

Can we get the best of both worlds?

- Agents plan independently if they can
- Agents plan as a multibody if they conflict

Definition

(a_i, a_j, v, t) is a conflict of agents a_i and a_j both planning to be in location $v \in V$ at time t.

Algorithm called *Independence detection* (ID)

Solution 3: Independence detection

```
def IndependenceDetection(problem, agents):
    solution = dict()
    groups = [[agent] for agent in agents]
    for group in groups:
       solution[group] = AStar(problem[group], [])
    while not IsValid(solution):
        # Merge confliting groups
        (group1,group2,v,t) = FindFirstConflict(solution)
        del solution[group1]
        del solution[group2]
        merged = [group1, group2]
        groups.append(merged)

        # Plan for the merged group
        solution[merged] = AStar(problem[merged], [])
    return solution
```

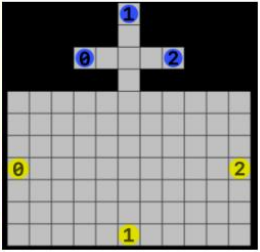
$(group_1, group_2, v, t) =$ `FindFirstConflict(solution)` is a conflict of (multibody) agents $group_1$ and $group_1$ because they both plan to be in location v at time t.

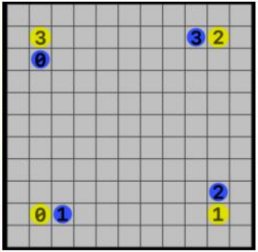
Solution 3: Independence detection

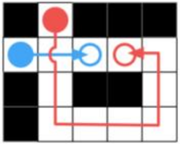
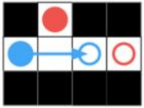
- Optimal and complete
- In the worst case, we end up with a group containing all agents

Solution 3: Independence detection

- Optimal and complete
- In the worst case, we end up with a group containing all agents





Solution 4: Conflict based search

- Use reservations (like Space-time A*)
  - but without blocking
- Re-plan in case of conflicts (like Independence Detection)
  - but without grouping

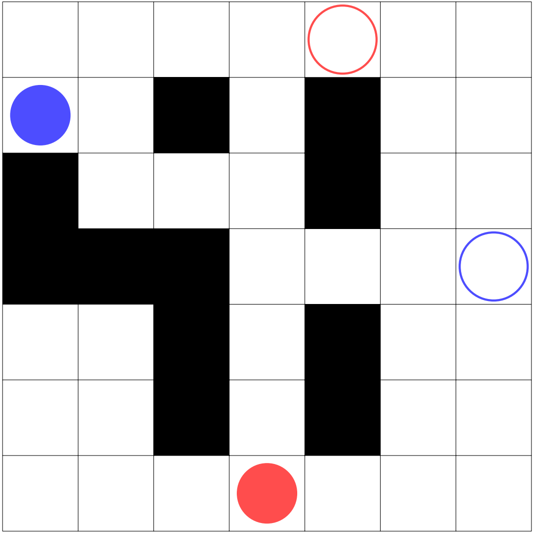
Solution 4: Conflict based search

- Use reservations (like Space-time A*)
  - but without blocking
- Re-plan in case of conflicts (like Independence Detection)
  - but without grouping
- Idea:
  - Use a high-level *constraint tree* to keep track of constraints
  - Use a low-level A* search for each agent (taking constraints into account)

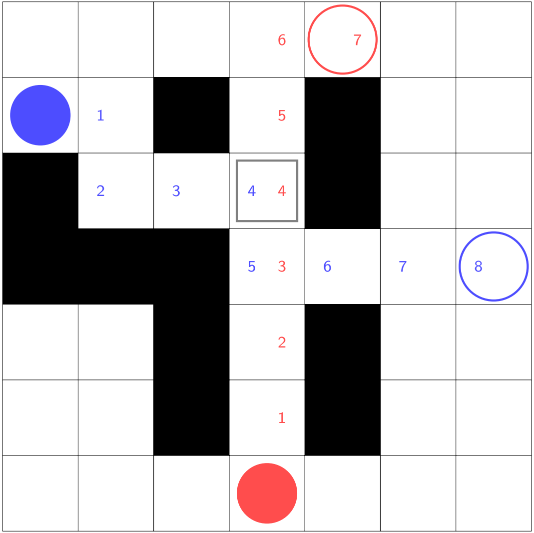
Solution 4: Conflict based search

- Use reservations (like Space-time A*)
  - but without blocking
- Re-plan in case of conflicts (like Independence Detection)
  - but without grouping
- Idea:
  - Use a high-level *constraint tree* to keep track of constraints
  - Use a low-level A* search for each agent (taking constraints into account)
- Conflict based search (CBS)

Solution 4: Conflict based search



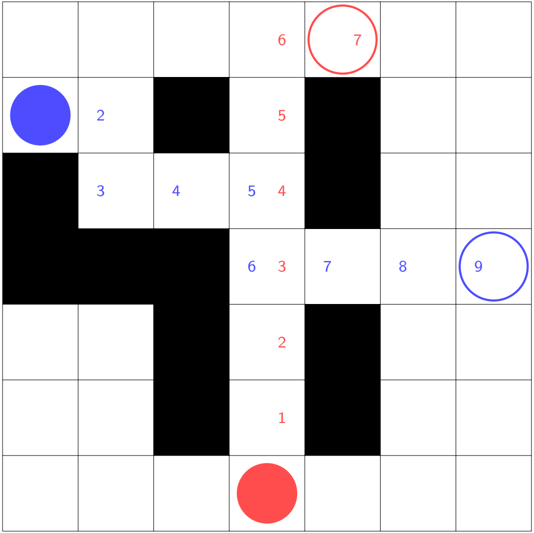
Solution 4: Conflict based search



$(\bullet, \bullet, D5, 4)$

Cost: $8 + 7$

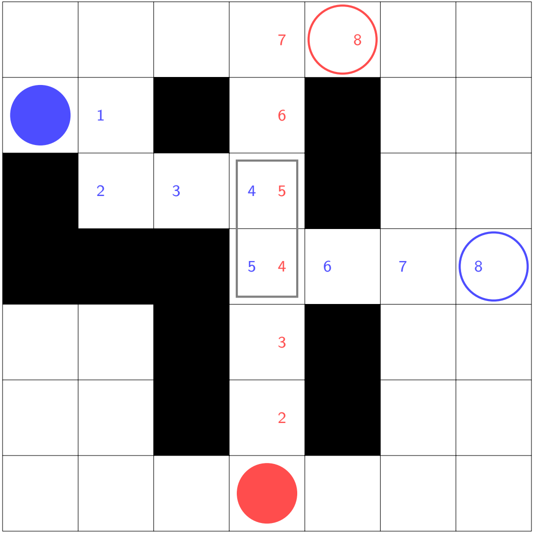
Solution 4: Conflict based search



$(\bullet, D5, 4)$

Cost: $9 + 7$

Solution 4: Conflict based search

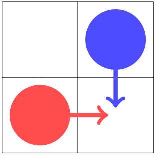


(●, $D5, 4$)

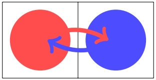
Cost: $8 + 8$

Conflicts

Vertex conflict

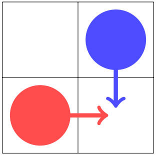


Edge conflict

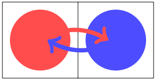


Conflicts

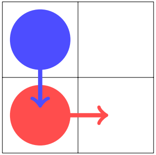
Vertex conflict



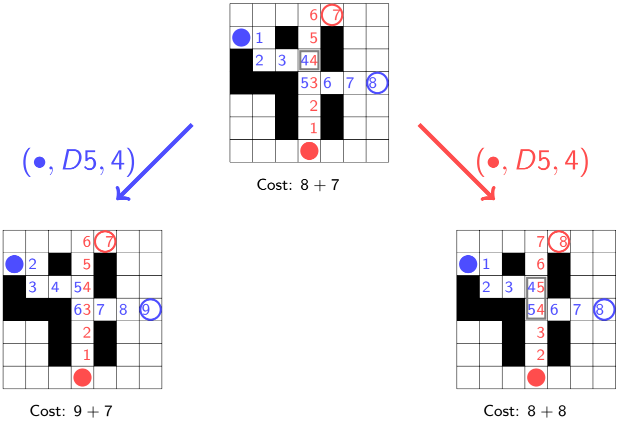
Edge conflict



Follow conflict



Solution 4: Conflict based search



Solution 4: Conflict based search

Cost functions:

$$\textsc{Flowtime}(\pi) = \sum_{i=1}^{k} |\pi_i|$$

$$\textsc{Makespan}(\pi) = \max_{i=1}^{k} |\pi_i|$$

Solution 4: Conflict based search

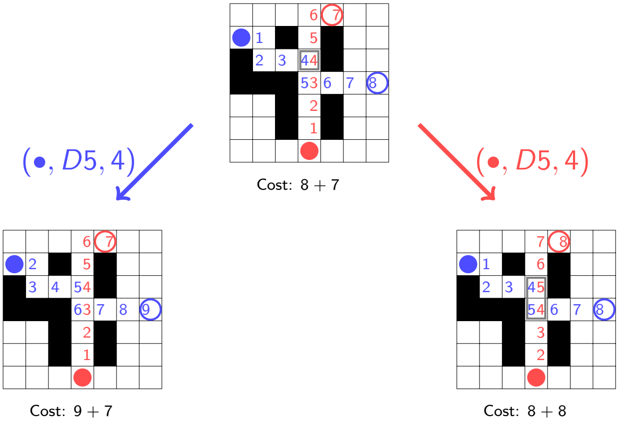
Cost functions:

$$\textsc{Flowtime}(\pi) = \sum_{i=1}^{k} |\pi_i|$$

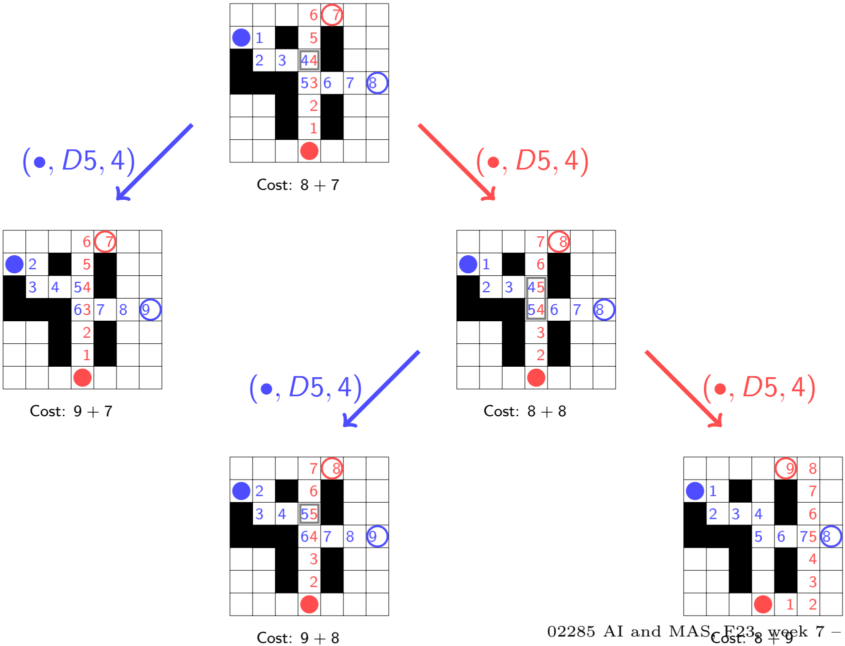
$$\textsc{Makespan}(\pi) = \max_{i=1}^{k} |\pi_i|$$

$\textsc{Fuel}$: Like $\textsc{Flowtime}$, but Noop is free

Solution 4: Conflict based search



Solution 4: Conflict based search



Solution 4: Conflict based search

```
def ConflictBasedSearch(problem, agents, cost):
    root.constraints = []
    for a in agents:
       root.solution[a] = AStar(problem[a])
    root.cost = cost(root.solution)
    frontier = PriorityQueue(root, lambda n: n.cost)
    while not frontier.empty():
        node = frontier.get()  # get lowest cost
        if IsValid(node.solution):
            return node.solution
        (a1, a2, v, t) = FindFirstConflict(node.solution)
        for a in [a1, a2]:
            m = n.copy()
            m.constraints.append((a, v, t))
            m.solution[a] = AStar(problem[a],
                                  m.constraints)
            m.cost = cost(m.solution)
            if m.cost < infinity:  # solution is found
                frontier.put(m)
```

Solution 4: Conflict based search

- CBS is complete and optimal because we're exploring all combinations of constraints to find a shortest solution for each agent

Solution 4: Conflict based search

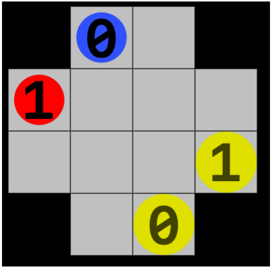
- CBS is complete and optimal because we're exploring all combinations of constraints to find a shortest solution for each agent
- Is CBS always better than MBA*, ID etc.?

Solution 4: Conflict based search

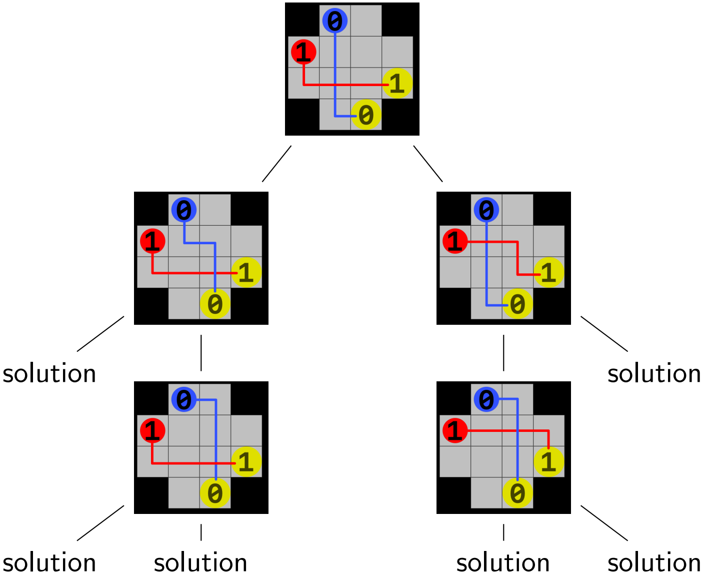
- CBS is complete and optimal because we're exploring all combinations of constraints to find a shortest solution for each agent
- Is CBS always better than MBA*, ID etc.?
- No, not if two agents are *strongly coupled*, i.e., have many conflicts among them

Solution 4: Conflict based search

- CBS is complete and optimal because we're exploring all combinations of constraints to find a shortest solution for each agent
- Is CBS always better than MBA*, ID etc.?
- No, not if two agents are *strongly coupled*, i.e., have many conflicts among them
- In an open $n \times n$ grid there are $2^{n/2}$ optimal paths between opposite corners



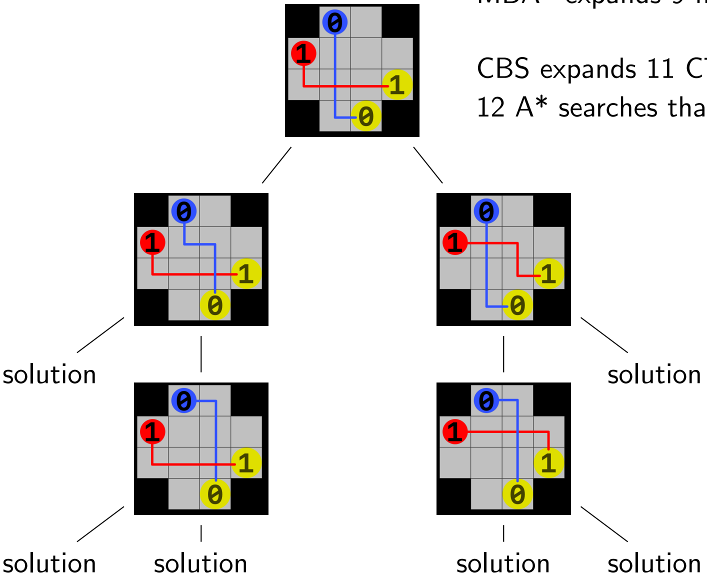
Solution 4: Conflict based search



Solution 4: Conflict based search

MBA* expands 9 nodes

CBS expands 11 CT nodes:
12 A* searches that expand 7 nodes



Solution 4.5: Meta-agent CBS

- Improvement: Meta-agent CBS (MA-CBS)
- If there are more than x conflicts between a_i and a_j, merge them into one *meta-agent*
- Same idea as ID and the Martha project

Solution 4.5: Meta-agent CBS

- Improvement: Meta-agent CBS (MA-CBS)
- If there are more than x conflicts between a_i and a_j, merge them into one *meta-agent*
- Same idea as ID and the Martha project
- We can tune the parameter x:
  - MA-CBS(∞) $=$ CBS
  - MA-CBS(0) $=$ ID

Solution 4.5: Meta-agent CBS

```
def ConflictBasedSearch(problem, agents, cost, x):
    conflicts = dict()
    root.constraints = []
    for a in agents:
       root.solution[a] = AStar(problem[a])
    root.cost = cost(root.solution)
    frontier = PriorityQueue(root, lambda n: n.cost)
    while not frontier.empty():
        node = frontier.get()  # get lowest cost
        if IsValid(node.solution):
            return node.solution
        (a1, a2, v, t) = FindFirstConflict(node.solution)
        if conflicts[(a1,a2)] > x:
            m = merge(a1, a2, constraints)
            m.solution[a] = AStar(problem[(a1,a2)]
                                      m.constraints)
            m.cost = cost(m.solution)
            if m.cost < infinity:  # solution is found
                frontier.put(m)
        else:
          conflicts[(a1,a2)] += 1  # then do as before
```

Summary

	Pros	**Cons**
MBA*	It works!	But inefficient

Summary

	Pros	**Cons**
MBA*	It works!	But inefficient
STA*	It's efficient!	But not complete, not optimal

Summary

	Pros	Cons
MBA*	It works!	But inefficient
STA*	It's efficient!	But not complete, not optimal
ID	Complete and optimal!	But corridors degrade performance

Summary

	Pros	Cons
MBA*	It works!	But inefficient
STA*	It's efficient!	But not complete, not optimal
ID	Complete and optimal!	But corridors degrade performance
CBS	Corridors, no problem!	But some agent merging again plz?

Summary

	Pros	**Cons**
MBA*	It works!	But inefficient
STA*	It's efficient!	But not complete, not optimal
ID	Complete and optimal!	But corridors degrade performance
CBS	Corridors, no problem!	But some agent merging again plz?
MA-CBS	Have some merge!	But what's a good x?

Now what?

- See (Silver, 2005) for STA*
- See (Standley, 2012) for ID
- CBS was introduced in (Sharon et al., 2015) and compares it to other state-of-the-art algorithms
    - Benchmark against other algorithms
    - Focus on completeness and optimality

Now what?

- See (Silver, 2005) for STA*
- See (Standley, 2012) for ID
- CBS was introduced in (Sharon et al., 2015) and compares it to other state-of-the-art algorithms
  - Benchmark against other algorithms
  - Focus on completeness and optimality
- This won't solve the hospital domain, but hope you can take inspiration from the analysis and comparisons of algorithms

References

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2014.11.006. URL https://www.sciencedirect.com/science/article/pii/S0004370214001386.

David Silver. Cooperative pathfinding. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, page 117–122, 2005.

Trevor Scott Standley. Independence detection for multi-agent pathfinding problems. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.