Intro to AI,
Autumn, 2025

MULTI-AGENT PATH FINDING:
TWO ROBOTS ROUTING

Group 4

Đào Ngọc Hiền	- 11247288
Nguyễn Thị Nhiên	- 11247337
Lê Ngọc Anh Thư	- 11247355
Trần Khải Vân	- 11247368

Table of contents

01

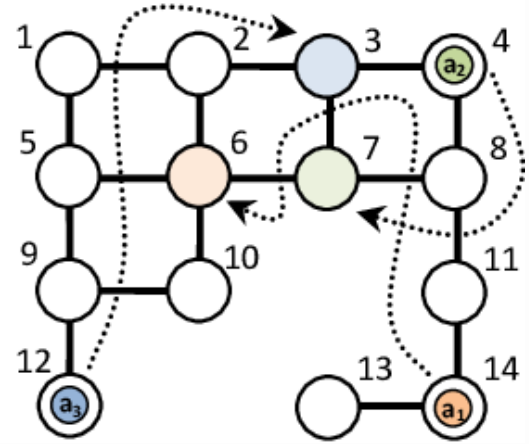
INTRODUCTION

PROBLEM STATEMENT AND REQUIREMENTS

What is MAPF?

Multi-Agent Path Finding (MAPF): the case of a number of agents navigating a **shared environment**

- MAPF asks multiple agents to move on a graph from given start nodes to goal nodes **without colliding**
- Time is discrete; at each step, an agent can **wait or move** to an adjacent node



In this report, we focus on ***2 Robots Routing*** only

PROBLEM STATEMENT AND REQUIREMENTS

What is MAPF?

Multi-Agent Path Finding (MAPF): the case of a number of agents navigating a **shared environment**

- MAPF asks multiple agents to move on a graph from given start nodes to goal nodes **without colliding**
- Time is discrete; at each step, an agent can **wait or move** to an adjacent node

In this report, we focus on ***2 Robots Routing*** only

Objectives in MAPF

Makespan

Minimize the latest finishing time

Sum of Costs - SOC:

Minimize the sum of the agents' path lengths/costs

In this report, we focus on ***Makespan*** only

REAL-WORLD PROBLEM

Warehouse Logistics Bots

Delivery robots sharing hallways; coordinate together to avoid jams

Autonomous Vehicles

Self-driving vehicles sharing road; make real-time driving decisions

Rail Shunting

Two locomotives share single-track segments; prevent face-to-face conflicts on the same edge.

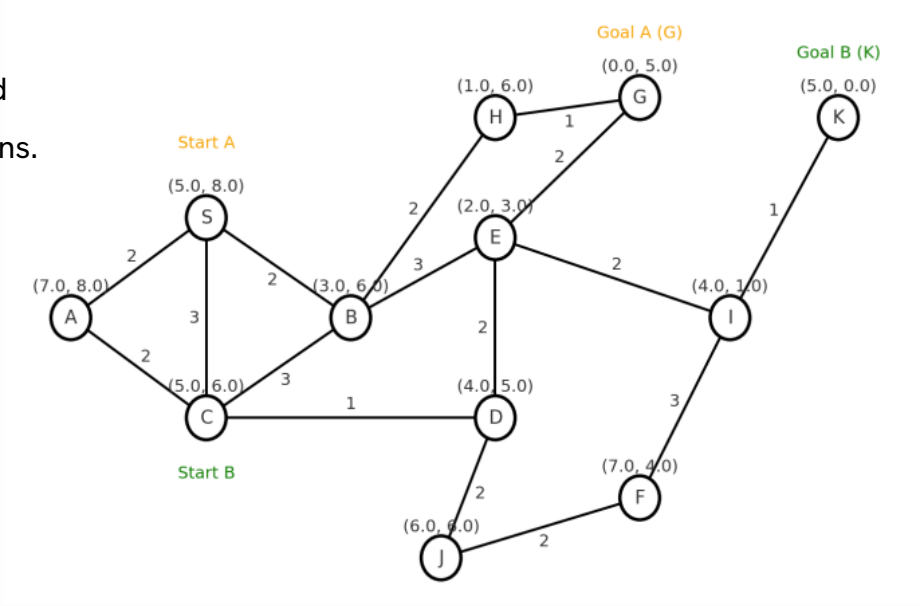
02

Problem Formulation

Map Simulation

Description:

- The graph with 12 vertices labeled A through K and two Robots with different Start and Goal destinations.
- At each vertex, there is a heuristic of the form (h_G, h_K), in which:
    - h_G: the shortest number of steps from that vertex to the Goal of Robot A (G)
    - h_K: the shortest number of steps from that vertex to the Goal of Robot B (K)



Map Simulation

Requirements:

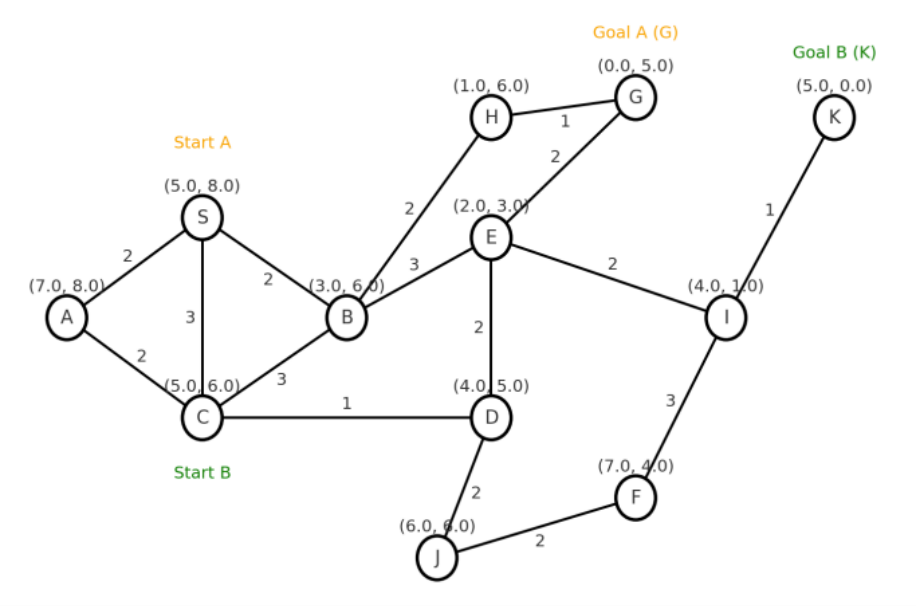
Find the path for two robots such that:

- The goal is the least number of latest finished time steps from two Robots
- At each time step, both Robots make a decision: move or wait

Constraints:

Collision avoidance:

- No duplicate vertices after each move
- No edge swapping from two Robots



Map Simulation

Input:

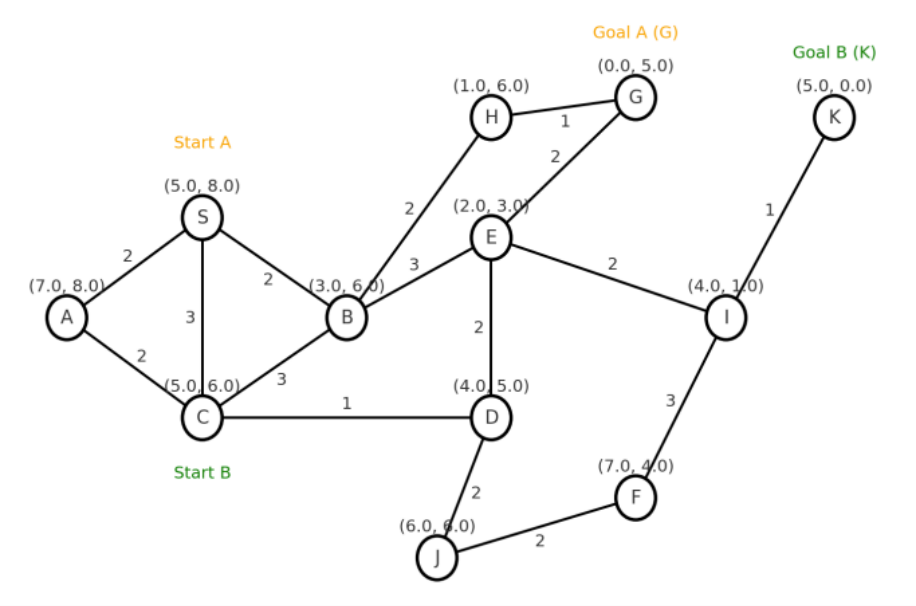
Start and goal vertices for each agent:

(StartA, StartB) → (GoalA, Goal B)

E.g: (S, C) → (G, K)

Output:

- A schedule of moves for both robots is given as a vertex sequence from Start to Goal.
- The cost values (makespan) of the returned solution



03

Algorithms

Breadth-First Search (BFS)

Working Principle

1. Start from the initial node and place it into a queue.
2. Explore all neighbors of the current node before moving to the next level.
3. Use a FIFO queue to maintain the order of exploration.
4. Mark visited nodes to avoid infinite loops in cyclic graphs.

```
BFS(Graph, start):
  create an empty queue Q
  enqueue start into Q
  mark start as visited
  while Q is not empty:
    current = dequeue(Q)
    if current is goal: return SUCCESS
    for each neighbor of current:
      if neighbor not visited:
        mark neighbor as visited
        enqueue neighbor into Q
```

BFS Pseudocode

A* Search

Working Principle

A* uses an evaluation function:

$$f(n) = g(n) + h(n)$$

- g(n) = cost of the path from the start node to n
- h(n) = heuristic estimate of the cost from n to the goal.

For A* to be optimal, the heuristic h(n) must satisfy:

- Admissibility: h(n) never overestimates the true cost.
- Consistency: *h(n) ≤ c(n, n′) + h(n′)* for every edge *(n, n′)*

```
A*(start, goal):
    open = {start}
    closed = {}
    g(start) = 0

    while open is not empty :
        n = node in open with lowest f ( n )
        if n is goal :
            return PATH

        remove n from open
        add n to closed

        for each neighbor of n :
            if neighbor in closed: continue
            tentative_g = g(n) + cost(n, neighbor)
            if neighbor not in open or tentative_g < g(neighbor):
                g(neighbor) = tentative_g
                h(neighbor) = heuristic(neighbor, goal)
                f(neighbor) = g(neighbor) + h(neighbor)
                if neighbor not in open :
                    add neighbor to open
```

A* Search Pseudocode

General Solution

- Assume *L* is the agent with the longer individual shortest path of timestep *d_L*;
  S is the agent with the shorter one *d_S*.

- The lower bound of Makespan value on shared environment is:

$$LB = max(d_L, d_S) = d_L$$

- Slack is the amount of time agent *S* can wait while still maintaining the validity of the plan:

$$Slack = LB - d_S = d_L - d_S$$

General Solution

Step 1

Use specified searching methods to find the shortest individual path for each agent and calculate *d_L, d_S* and *LB of makespan*

Step 2

Calculate *Slack* = *d_L* – *d_S*

Step 3

Choose one shortest path for L. Fix the schedule of L in the shared environment: each step moves 1 edge, no waiting allowed.

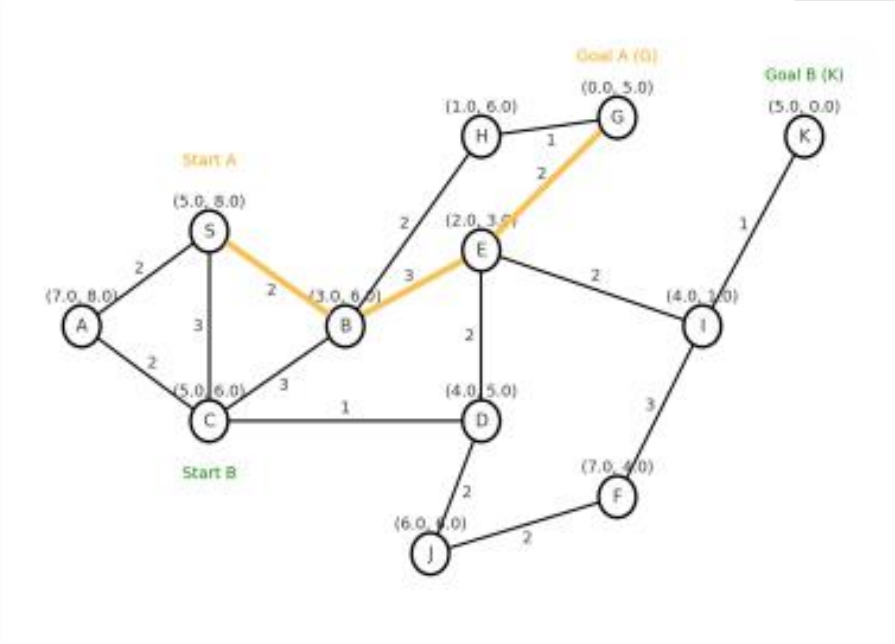
Step 4:

Create a time-space graph in starting from t = 0 for S (waiting allowed) with the constraint of avoiding collisions with the trajectory of L
If the path cannot be found in *LB* steps, try the next shortest path of *L* and repeat from step 1 until the final state is reached in *LB*

BFS for Robot A

Step	u	edge(u)	OPEN
0	–	–	S
1	S	B, A	B, A
2	B	E, H, C	A, E, H, C
3	A	–	E, H, C
4	E	G, I, D	H, C, G, I, D
5	H	–	C, G, I, D
6	C	–	G, I, D
7	G	–	–

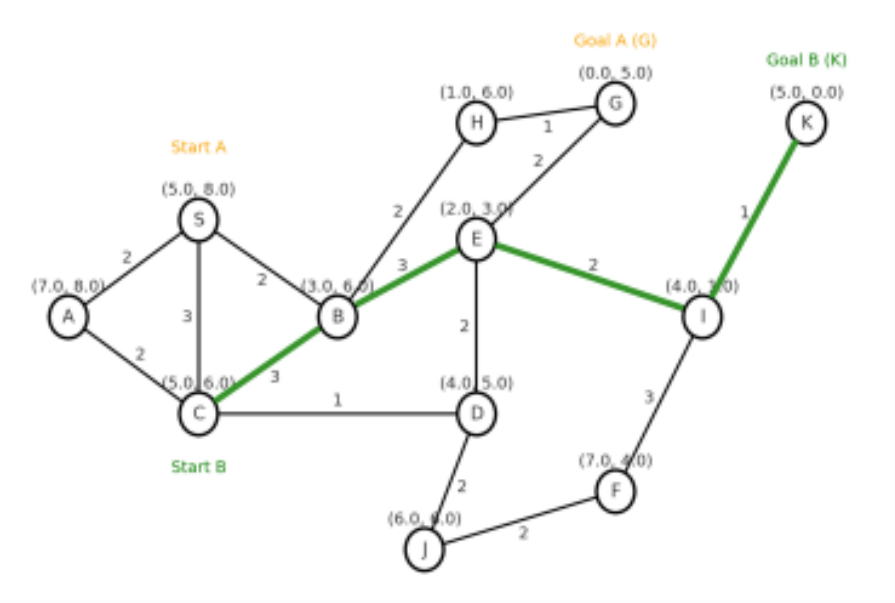
3 step: $S \rightarrow B \rightarrow E \rightarrow G$



BFS for Robot B

Step	u	edge(u)	OPEN
0	–	–	C
1	C	B, S, A, D	B, S, A, D
2	B	E, H	S, A, D, E, H
3	S	–	A, D, E, H
4	A	–	D, E, H
5	D	J	E, H, J
6	E	I, G	H, J, I, G
7	H	–	J, I, G
8	J	F	I, G
9	I	K	G, K
10	G	–	K
11	K	–	–

4 step: $C \to B \to E \to I \to K$



04

Experiments & Results

Joint BFS

LB: Makespan ≥ max(stepA, stepB) = max(3, 4) = 4

Slack = LB – stepA = 4 – 3 = 1

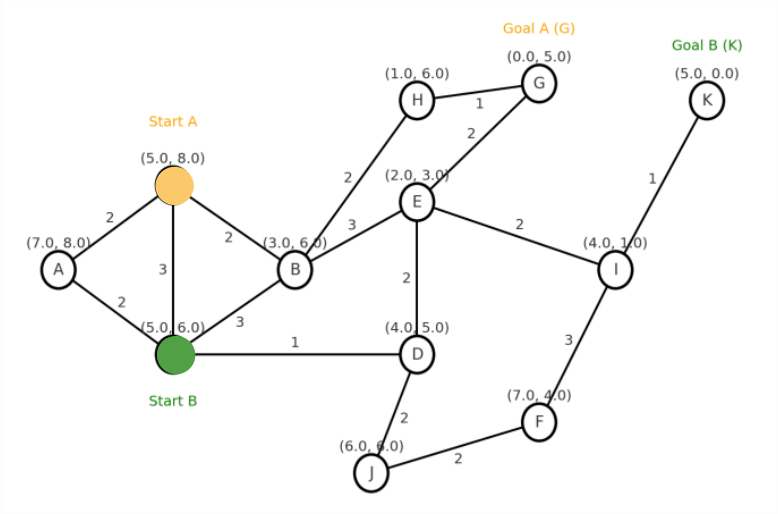
➔ To achieve the optimum in the shared environment:

- Robot B must move continuously
- Robot A can wait at most 1 step

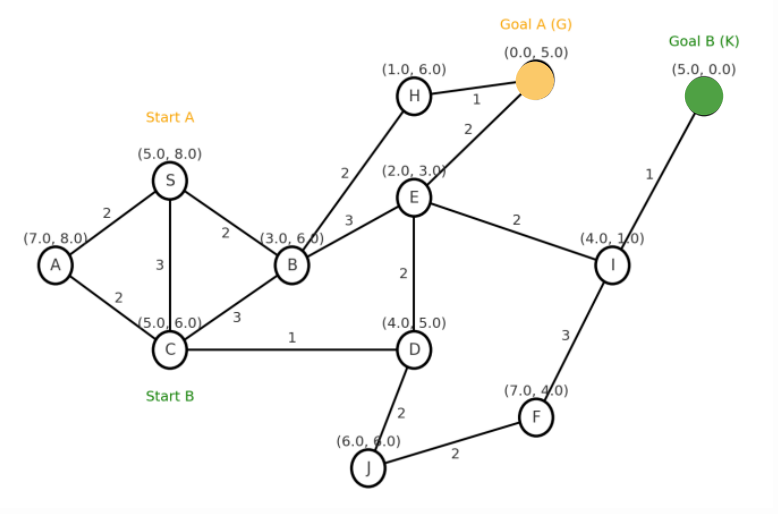
Input Data/Test cases

Example Input:

- Robot A: S → G
- Robot B: C → K



Initial State

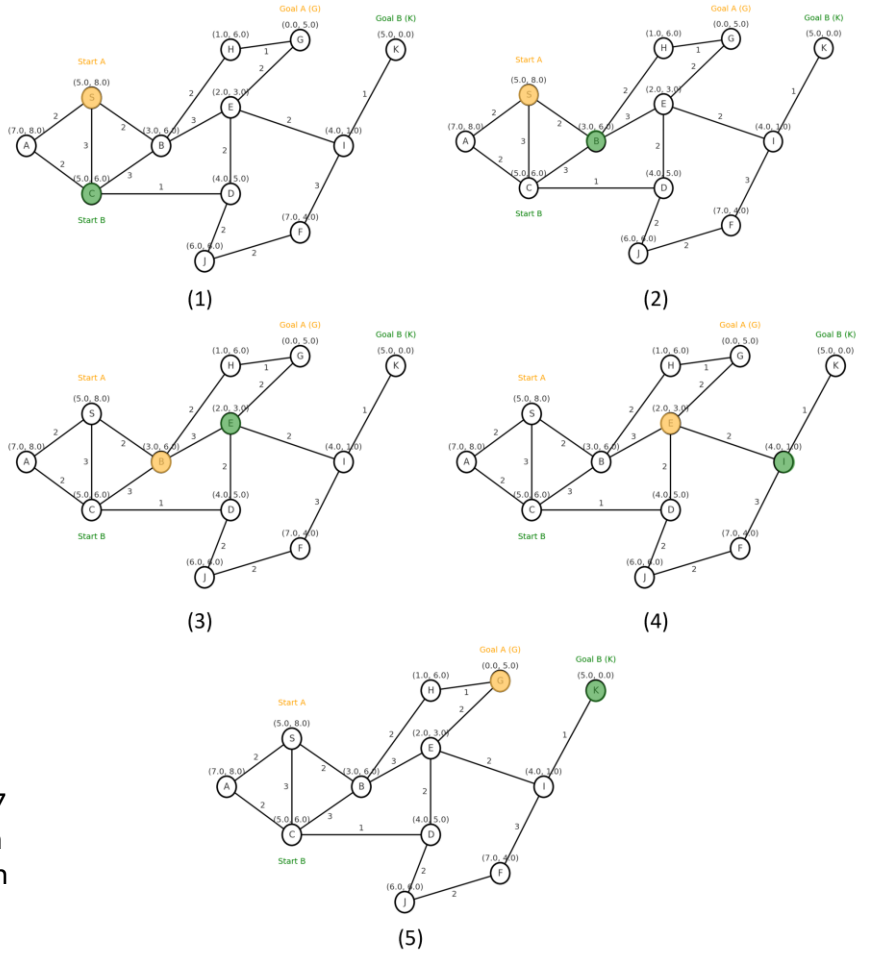


Final State

Joint BFS Final Path

t	(pos_A, pos_B)	A action	B action
0	(S, C)	start	start
1	(S, B)	wait	C→B
2	(B, E)	S→B	B→E
3	(E, I)	B→E	E→I
4	(G, K)	E→G	I→K

Makespan = 4



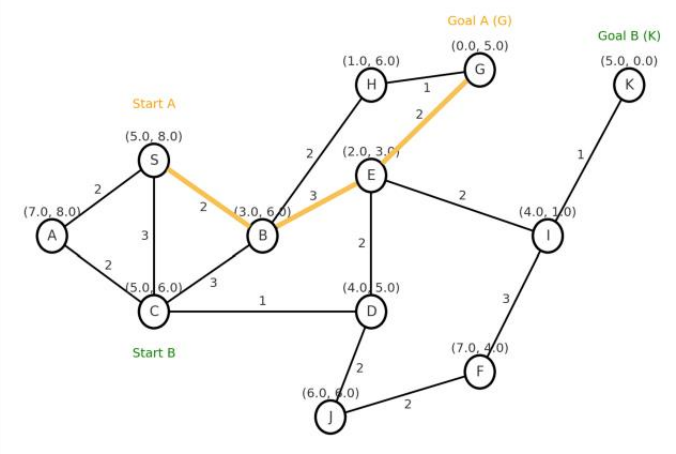
Note:

- The visualized graph and the execution table above are just **the final result** of the solution using BFS.
- The full tracking table would expanded over 107 states since **BFS does not decide next moves** on the spot: it explores the entire tree first and then extracts paths from the search tree.

A* Search for Robot A

Step	u	v	$h(v)$	$g(v)$	$f(v)$	OPEN
0	–	–	–	–	–	S_5
1	S	A, C, B	$h(A) = 7$ $h(C) = 5$ $h(B) = 3$	$g(A) = 2$ $g(C) = 3$ $g(B) = 2$	$f(A) = 2 + 7 = 9$ $f(C) = 3 + 5 = 8$ $f(B) = 2 + 3 = 5$	B_5, C_8, A_9
2	B	H, E	$h(H) = 1$ $h(E) = 2$	$g(H) = 2 + 2 = 4$ $g(E) = 2 + 3 = 5$	$f(H) = 4 + 1 = 5$ $f(E) = 5 + 2 = 7$	H_5, E_7, C_8, A_9
3	H	G	$h(G) = 0$	$g(G) = 4 + 1 = 5$	$f(G) = 5 + 0 = 5$	G_5, E_7, C_8, A_9
4	G	–	–	–	–	–

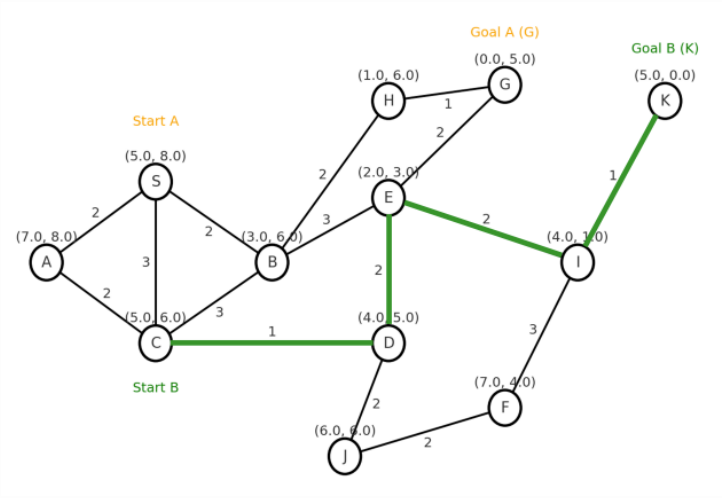
3 step: $C \rightarrow B \rightarrow H \rightarrow G$



A* Search for Robot B

Step	u	v	$h(v)$	$g(v)$	$f(v)$	OPEN
0	–	–	–	–	–	C_6
1	C	A, S, B, D	$h(A) = 8$ $h(S) = 8$ $h(B) = 6$ $h(D) = 5$	$g(A) = 2$ $g(S) = 3$ $g(B) = 3$ $g(D) = 1$	$f(A) = 2 + 8 = 10$ $f(S) = 3 + 8 = 11$ $f(B) = 3 + 6 = 9$ $f(D) = 1 + 5 = 6$	D_6, B_9, A_{10}, S_{11}
2	D	E, J	$h(E) = 3$ $h(J) = 6$	$g(E) = 2 + 1 = 3$ $g(J) = 2 + 1 = 3$	$f(E) = 3 + 3 = 6$ $f(J) = 3 + 6 = 9$	$E_6, J_9, B_9, A_{10}, S_{11}$
3	E	G, I	$h(G) = 5$ $h(I) = 1$	$g(G) = 3 + 2 = 5$ $g(I) = 3 + 2 = 5$	$f(G) = 5 + 5 = 10$ $f(I) = 5 + 1 = 6$	$I_6, J_9, B_9, G_{10}, A_{10}, S_{11}$
4	I	K, F	$h(K) = 0$ $h(F) = 4$	$g(K) = 5 + 1 = 6$ $g(F) = 5 + 3 = 8$	$f(K) = 6 + 0 = 6$ $f(F) = 8 + 4 = 12$	$K_6, J_9, B_9, G_{10}, A_{10}, S_{11}, F_{12}$
5	K	–	–	–	–	–

4 step: $C \to D \to E \to I \to K$



Joint A*

$$LB: Makespan \geq max(stepA, stepB) = max(3, 4) = 4$$

$$Slack = LB - stepA = 4 - 3 = 1$$

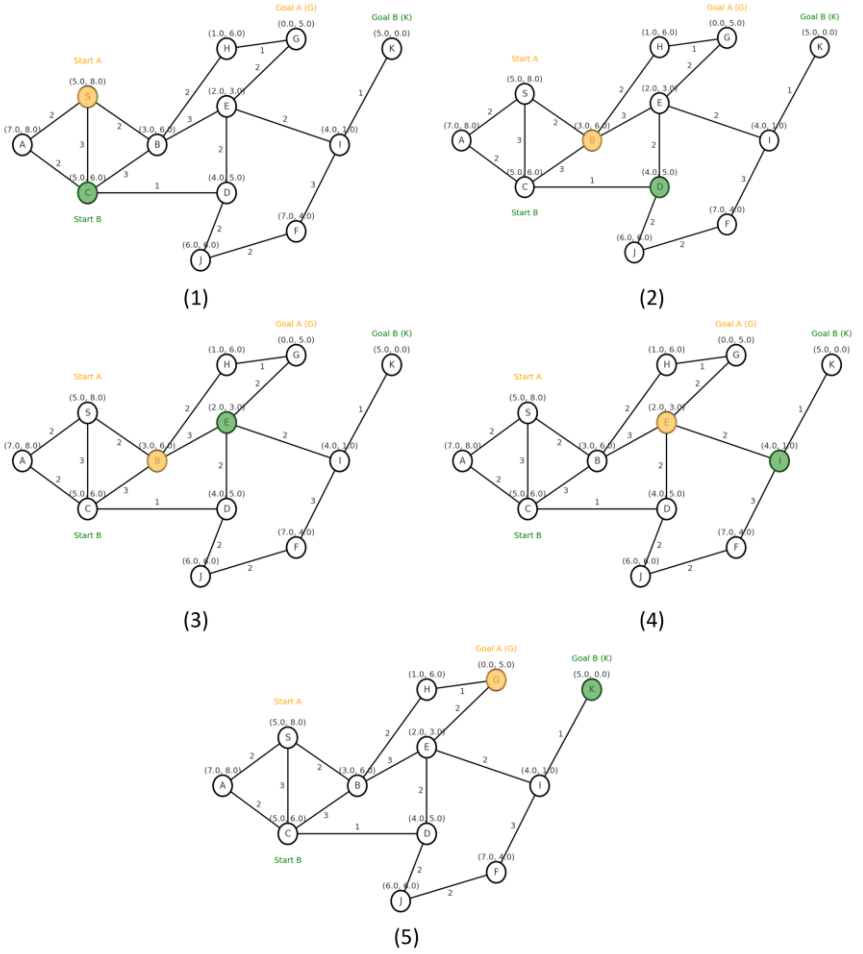
➔ To achieve the optimum in the shared environment:

- Robot B must move continuously
- Robot A can wait at most 1 step

Joint A* Final Path

t	(pos_A, pos_B)	A action	B action
0	(S, C)	start	start
1	(B, D)	S→B	C→D
2	(B, E)	wait	D→E
3	(E, I)	B→E	E→I
4	(G, K)	E→G	I→K

Makespan = 4



Note:

- The visualized graph and the execution table above are just **the final result** of the solution using A* Search.
- The full tracking table would expanded to 8 states due to positional permutation set but was pruned due to priority OPEN queue

Comparison table

Start -> Goal	BFS expanded	A* expanded	BFS time (ms)	A* time (ms)
(S,C) -> (G,K)	107	8	0.2973	0.1161
(A,B) -> (I,J)	120	15	0.3112	0.1499
(C,I) -> (I,C)	104	21	0.3174	0.2164
(K,G) -> (G,K)	117	13	0.2965	0.1002
(J,H) -> (S,A)	132	9	0.3253	0.0895

Observations:

- A* narrows the search range more aggressively
- A* typically executes twice as fast as BFS

➔ In most cases, A* outperforms BFS.

05

Discussion

The meaning in the real-world context

2 Robot routing problems shows up in many smart tasks today

Amazon Kiva robots

Volvo Drive Pilot

The meaning in the real-world context

- In fact, the real-world problem could be more complex due to larger number of agents, terrain complexity, etc. Besides, there are some better algorithms that can handle these issues.
- However, BFS and A* remain the basis for these other algorithms and are essential since many real systems reduce locally to **pairwise interactions.** It’s a clean step toward multi-robot coordination

Thanks

for listening!