MULTI-AGENT PATH DETECTION

Sr. No	Name	Roll number
1	ASHIT KUMAR	17IM30006
2	SHUBHAM KAKDE	17IM30022
3	YASH VATSA	17IM10032

Problem Statement

Consider a warehouse and a set of robots that pick up items from designated places, deliver those in desired locations, and finally, the robots go to the end location. Assume that the warehouse is represented as a 2-D grid of size n × m. Each location (Li) on the warehouse floor can be denoted by a pair of integers Li = (xi,yi). Each cell on the grid can be categorized in one of the following ways (see diagram below) - source location (P1-P3), destination location (D1-D3), temporary storage location (TS1-TS4), obstacle (black square), normal (rest of the cells). Source & destination denote pick-up and drop locations respectively.

Temporary storage location denotes the place where the robot can keep some items. Obstacles represent a location where no robot can move. The rest of the cells are considered normal cells.

Our Approach

Heuristics Approach

We construct a directed weighted graph G = (V, E) = R ∪ T where ri ∈ R is a robot and ti ∈ T is a task. We have been given initial (si) and final locations (ei) for each of the robots and the pick-up (pi) and delivery locations (di) of the tasks.

We assume that we have **m** agents(robots) and **n** tasks. Thus, in this case, we have a graph of **(m + n)** vertices and **(m+n-1)!** Hamiltonian Cycles. Each of these cycles corresponds to task assignment to the robots.

Key Aspects of the algorithm

1) Every cycle can now be considered to be made of sub-paths starting with a robot and followed by a list of tasks or no tasks at all
2) All possible hamiltonian cycles hence correspond to all possible assignment and consequent scheduling of tasks where each of said subpaths have a weighted path length that corresponds to the total time taken by the robot to accomplish all tasks that are assigned to it
3) Optimization problem becomes to find the optimal hamiltonian cycle where the maximum weighted path length amongst all such sub-paths becomes minimized

This is however **not** an optimal solution although it always returns a solution i.e. it is a complete heuristic algorithm. The complexity is **exponential** since brute force approach involves traversing all (m+n-1)! cycles to find the optimal assignment.

The **heuristic assumption** is that time taken for an agent (robot) to complete all tasks without colliding with other robots in a multiple-agent environment has been assumed to be the same as the time taken in a single-agent environment.

For sparse graphs i.e. small number of robots and tasks compared to warehouse size and reasonably well-distributed pickup, delivery, start and end locations, the heuristic algorithm performs quite well.

Optimal Approach

Hamiltonian cycles

A Hamiltonian path or traceable path is a path that visits each vertex of the graph exactly once. A graph that contains a Hamiltonian path is called a traceable graph. A graph is Hamiltonian-connected if for every pair of vertices there is a Hamiltonian path between the two vertices.

A Star

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance traveled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes

$f(n)=g(n)+h(n)$
where n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. A* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be extended.

Conflict Based Search
In the multi-agent pathfinding problem (MAPF) we are given a set of agents each with respective start and goal positions. The task is to find paths for all agents while avoiding collisions. Most previous work on solving this problem optimally has treated the individual agents as a single ‘joint agent’ and then applied single-agent search variants of the A* algorithm.

Now we use TCBS(Task Conflict-Based Search) to generate a conflict tree. We then apply a modified A* algorithm to find the optimal solution.

The heuristic function returns a list of heuristics for each of the robots. For each task it assigns the task to the closest robot. At the end of all assignments for each robot it adds the cost to travel from its last delivery location to its end location. Distance returns the **manhattan distance** between two nodes.

Heuristic pseudo code:

```
Input: node s
cost = 0
heuristics = []

for t ∈/ s.т do
heuristics[closest agent] += Distance (closest agent, start(t)) +
heuristics[closest agent] += Distance (start(t), goal(t))

for each agent:
    heuristics[agent] += Distance(last delivery location, home location)
return cost
```

Now for optimal solution,

Inputs:
g = array of cost so far s for each agent
h = array of heuristics for each agent
f = array of total cost estimates, fi = gi + hi
f* = array of total actual costs

G = max(g)
F = max(f)
F* = max(f*)
H = F - G
H* = F* - G

Since each of the tasks are assigned to the robots closest to them at a particular location,
We have **h(n) ≤ h*(n)**

So f(n) ≤ f*(n) ∀ agent n

H = max(f) - max(g) ≤ max(f*) - max(g) = F* - G = H*

Hence **H ≤ H***

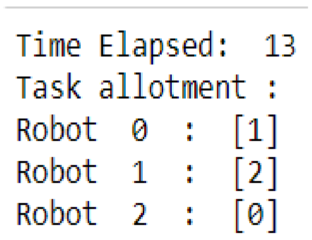
So we have an admissible heuristic. Hence applying A* on the conflict tree will give us the optimal solution!

Input test case format

```
5 12
1 1 1 1 1 1 1 1 1 1 2 1
1 1 1 1 1 1 1 2 1 1 1 2
1 0 1 1 1 1 2 1 1 1 2 1
1 1 1 2 1 1 1 0 1 1 1 1
1 1 1 1 1 1 2 1 1 0 1 2
3
0 0 3 4
1 2 3 5
2 1 1 5
3
2 0 3 2
4 1 2 3
2 5 4 8
```

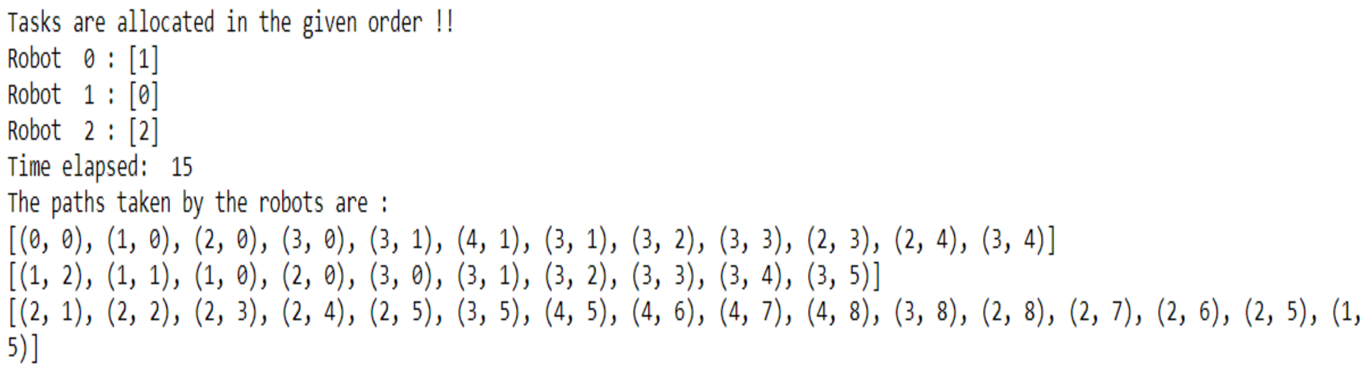
Output

Heuristics



```
Time Elapsed:  13
Task allotment :
Robot  0  :  [1]
Robot  1  :  [2]
Robot  2  :  [0]
```

Optimal



```
Tasks are allocated in the given order !!
Robot  0 : [1]
Robot  1 : [0]
Robot  2 : [2]
Time elapsed:  15
The paths taken by the robots are :
[(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (3, 1), (3, 2), (3, 3), (2, 3), (2, 4), (3, 4)]
[(1, 2), (1, 1), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5)]
[(2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 5), (4, 5), (4, 6), (4, 7), (4, 8), (3, 8), (2, 8), (2, 7), (2, 6), (2, 5), (1,
5)]
```