ARTIFICIAL INTELLIGENCE FOUNDATION AND APPLICATIONS (AI61005) : ASSIGNMENT 1

Multi Agent Path Finding problem implementation using Conflict Based Search (CBS)

Introduction

Multi Agent Path finding (MAPF) computes a set of collision-free paths for multiple agents connecting their respective start and goal locations while optimizing a scalar measure of paths. Variants of MAPF have been widely studied in the robotics community over the last few years.

Multi Agent Path finding (MAPF) has vast applications in the field of:

1. **Robotics and Autonomous Vehicles:**
    - In robotics, multi-agent pathfinding is crucial for coordinating the movement of multiple robots or autonomous vehicles in environments like warehouses, factories, and transportation systems.
    - It ensures that robots do not collide with each other while navigating, optimizing efficiency and safety.
2. **Traffic Management**
    - In transportation and traffic management systems, multi-agent pathfinding can optimize traffic flow, reduce congestion, and prevent accidents.
    - It helps in coordinating the routes of vehicles in real-time, especially in scenarios like autonomous vehicles sharing the road with human-driven vehicles.
3. **Drones and UAVs:**
    - For applications such as surveillance, delivery, and search and rescue, multi-agent pathfinding is essential to coordinate the movements of multiple drones.
    - It ensures that drones avoid collisions and efficiently cover the desired areas.
4. **Multi-Robot Systems:**
    - In fields like agriculture, multi-agent pathfinding can be used to coordinate the movements of multiple agricultural robots for tasks like planting, harvesting, and monitoring crops.

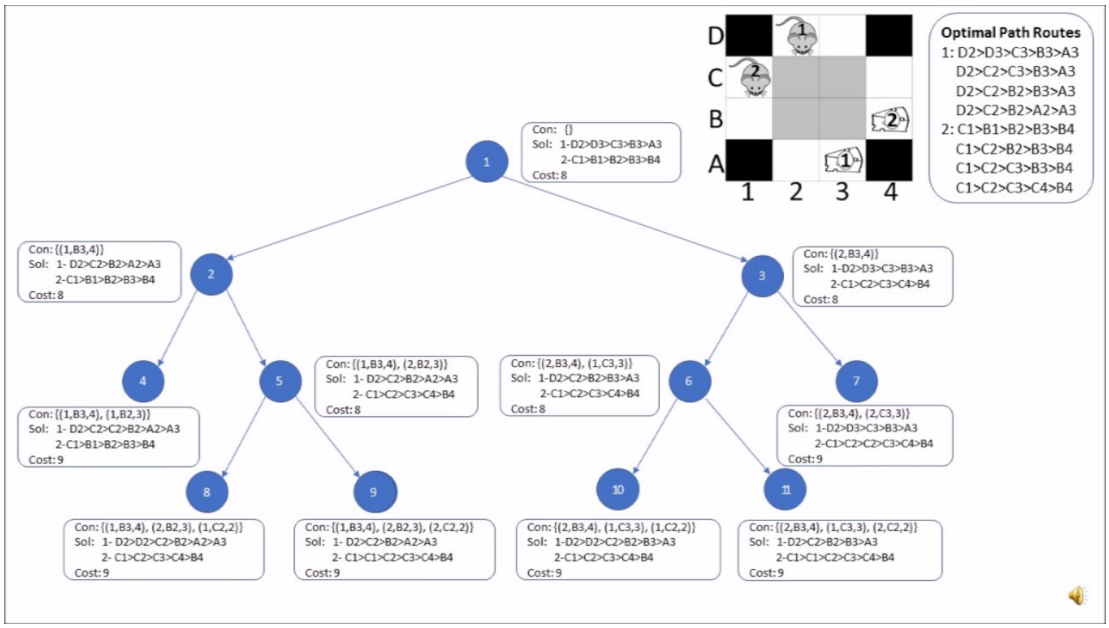
    Multi-agent pathfinding algorithms range from centralized approaches that find a global solution to decentralized methods where agents cooperate or negotiate their paths. These algorithms consider factors like agent priorities, dynamic changes in the

environment, and real-time constraints to provide practical solutions for various applications.

Conflict Based Search (CBS)

CBS is a two-layer search algorithm. The bottom search finds an optimal path for each agent. When a conflict occurs between paths, the algorithm uses a high-level algorithm to split the conflict between the two known paths, and adds constraints to the agent to avoid conflicts. The working principle is to decompose the problem into a large number of single agent pathfinding problems. The algorithm is divided into two levels: high-level (search constraint tree) and low-level (constrained single agent path finding).

- **High Level:** The high-level search of CBS is a best first search (BFS) that uses the costs of the high-level nodes as f-values. The cost of a high-level node is the sum of the path costs of its paths. When CBS expands a high-level node N, it checks whether the node is a goal node. A high-level node is a goal node if and only if none of its paths are in collision. If N is a goal node, then CBS terminates successfully and outputs the paths of the goal node as solution
- **Low Level:** When CBS generates a high-level node N, it performs a low-level search for each agent independently. The low-level search for agent j is a (best-first) A* search that ignores all other agents and finds a minimum-cost path from agent j's start vertex to its goal vertex that is both feasible and respects the constraints of N that involve agent j.



Problem Statement

A certain shop floor is represented as a 2D grid if size n * m. In the shop floor, robots are employed to perform shipment tasks, where a task involves shipping a component from a designated pick-up cell location to a destination cell. There are ‘k’ robots and ‘k’ shipment tasks. Each robot performs a single task. A robot can move at most one cell (either vertically or horizontally) at a time step. A cell cannot be simultaneously occupied by more than one robot. Black coloured cells are obstacles and cannot be traversed through. All robots start at the same time. The ith robot starts from cell location R_i, picks up one the components (say) at location P_j, delivers it in cell E_j and finally moves to cell D_i. The objective is to choose the appropriate task for each robot and determine its corresponding travel path, so that overall completion time for the work schedule involving all tasks is minimized.

Step-wise Depiction of the solution approach

Here's an explanation of the **program's functionality in stages:**

1. **Grid Setup:** The grid is represented as a 2D array where 0s indicate accessible cells, and 1s represent obstacles.

2. **Product Assignments:** There are three products, each with pickup and drop locations. The program explores different assignments of these products to robots to find the most efficient solution.
3. **Low-Level Pathfinding (A* Search):** For each product assignment, the program uses the A* search algorithm to find paths for individual robots from their starting positions to pick up the product, from the pickup location to the drop location, and finally from the drop location to the goal. The a_star_pathfinding function calculates the shortest path for a single robot using the Manhattan distance heuristic.
4. **High-Level Programming (Conflict Resolution):** After finding paths for each robot, the program constructs an initial CBSNode that contains these paths and their associated costs. It uses a CBS-based approach to handle conflicts between the paths of different robots. CBS iteratively resolves conflicts and updates the paths until no conflicts remain. The resolve_conflicts function detects conflicts and modifies the paths to resolve them. The CBS algorithm prioritizes nodes with lower costs, attempting to minimize the overall cost while considering conflicts.
5. **Optimization:** The program explores different assignments of products to robots, trying to find the assignment that results in the lowest overall cost.It maintains the minimum cost and the associated solution throughout this exploration.

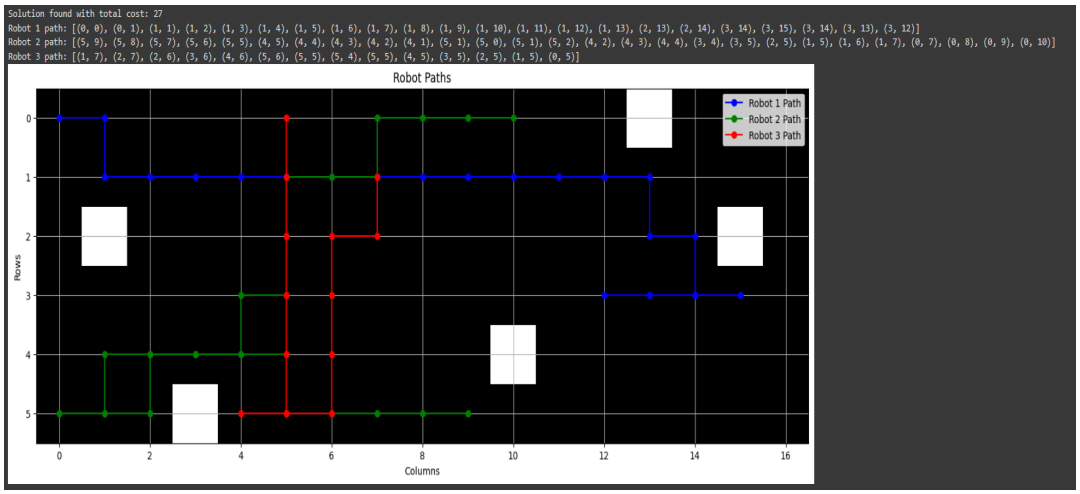
6. **Output and Visualization:** The program prints the found solution, including the total cost and the paths for each robot. It also provides a visual representation of the paths on the grid using matplotlib.

To summarize, this program demonstrates a combination of low-level A* pathfinding for individual robots and high-level CBS-based conflict resolution to efficiently solve a multi-agent pathfinding problem in a grid-based environment. It explores different product assignments to find the optimal solution with the lowest overall cost.

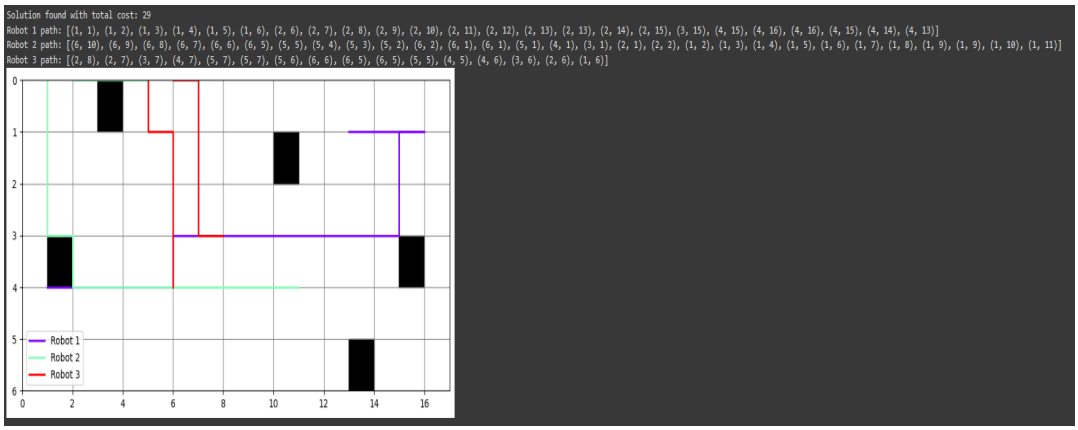
Test Case Analysis

- **1st Test Case**

CBS

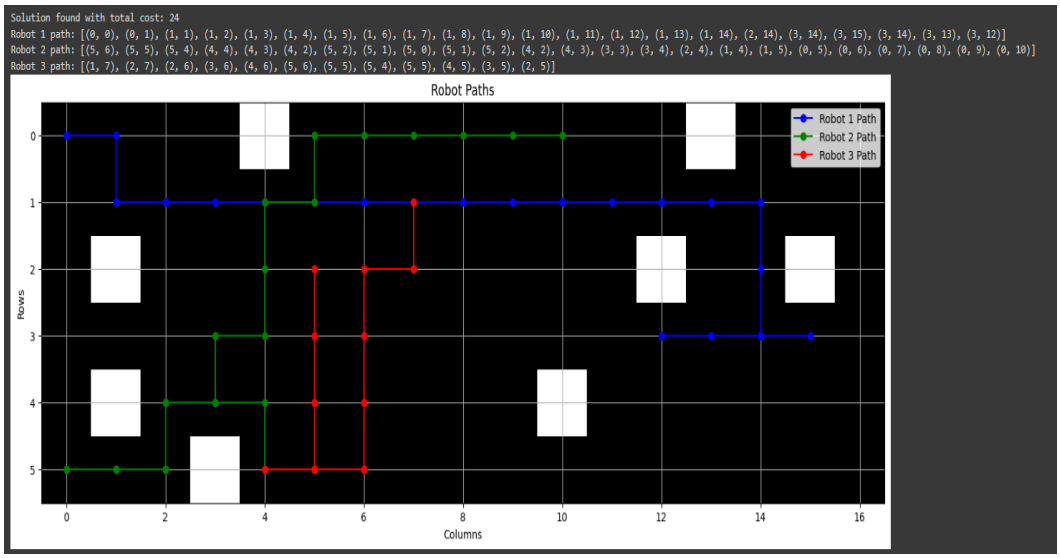


A* (A-Star)

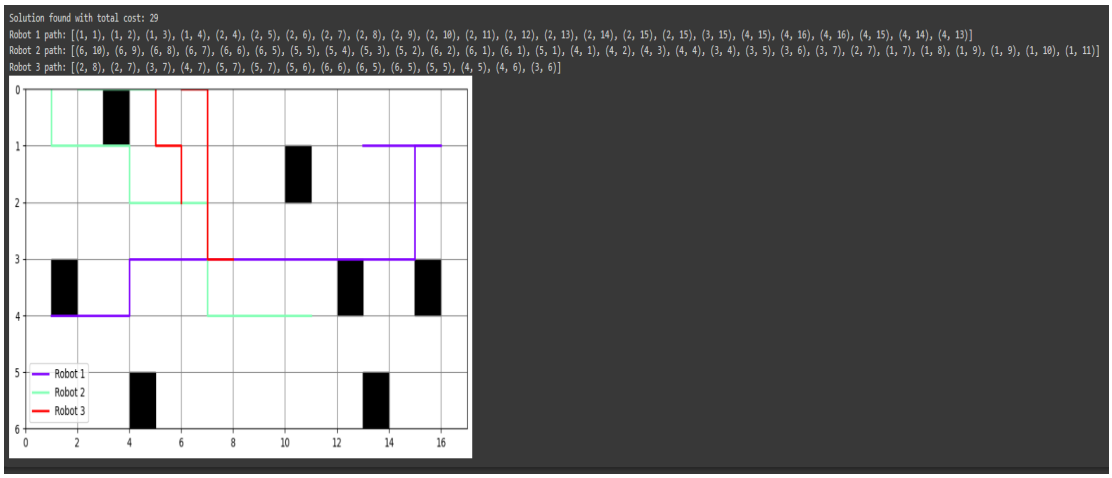


- **2nd Test Case**

CBS

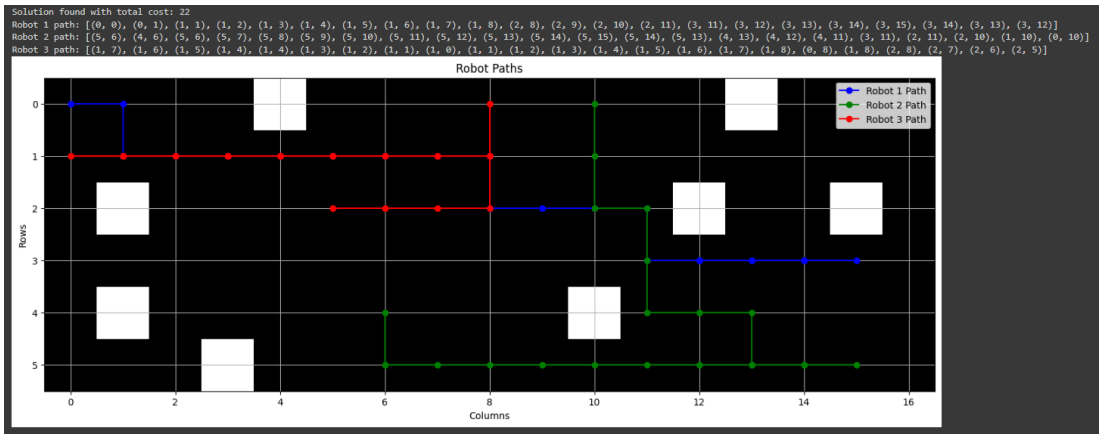


A* (A-Star)

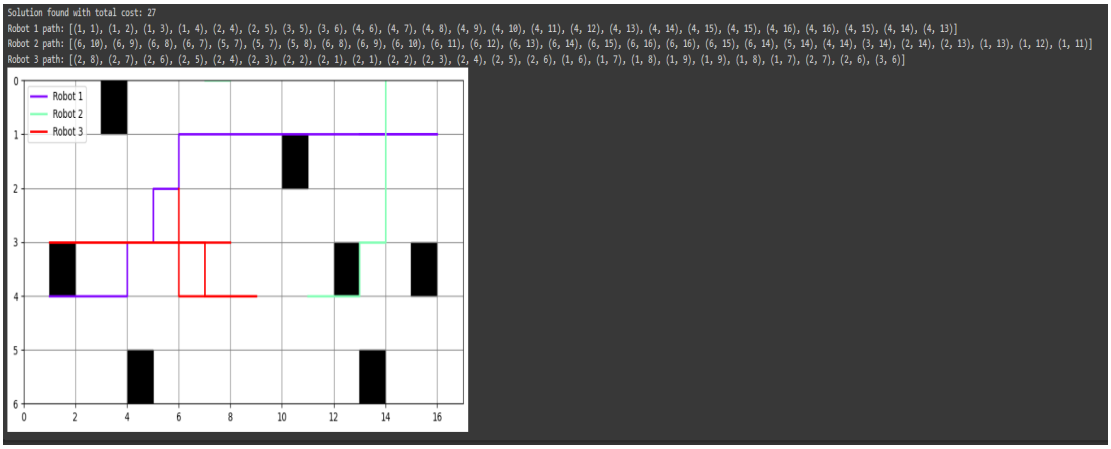


- **3rd Test Case**

CBS



A* (A-Star)



Time Overhead Comparison

TIME OVERHEAD COMPARISION						
					TOTAL COST	
		Ri	Di	OBSTACLES LOCATION	CBS	A* Search
TEST CASE 1	ROBOT 1	(1,1)	(4,13)	(3,2), (6,4), (5,11), (1,14), (3,16)	27	29
	ROBOT 2	(6,10)	(1,11)			
	ROBOT 3	(2,8)	(1,6)			
TEST CASE 2	ROBOT 1	(1,1)	(4,13)	(3,2), (6,4), (5,11), (1,14), (3,16), (1,5), (3,13)	24	29
	ROBOT 2	(6,10)	(1,11)			
	ROBOT 3	(2,8)	(3,6)			
TEST CASE 3	ROBOT 1	(1,1)	(4,13)	(3,2), (6,4), (5,11), (1,14), (3,16), (1,5), (3,13)	22	27
	ROBOT 2	(6,10)	(1,11)			
	ROBOT 3	(2,8)	(3,6)			
	**** Test case 2 and test case 3 contrast in terms of the pickup and drop-off locations for the three robots.					

Analysis

In this comparative analysis of the Conflict-Based Search (CBS) algorithm and the A* algorithm for solving multi-agent pathfinding problems, several key elements were taken into consideration. First, the execution time of each algorithm was measured for the identical problem instance. It determined that the CBS set of rules outperformed A* in terms of solution quality, generating a lower total cost(27 vs 29 in test case 1) and similarly in other test cases. This indicates CBS's effectiveness in coping with multi-agent conflicts and optimizing path lengths.Additionally, scalability and robustness tests were suggested for evaluating the algorithms' performance in more complex scenarios and varying obstacle layouts. Memory utilization needs to be monitored, especially for CBS, which may require greater memory due to conflict table renovation. The choice among CBS and A* relies upon on problem trends, with CBS being suitable for multi-agent problems with conflicts, on the same time as A* may additionally excel in single-agent tasks. Lastly, it is crucial to ensure that CBS constantly produces optimal solutions, as it goals for optimality.

Discussion:

The program in the ipynb file is designed to solve a multi-agent pathfinding problem, where multiple robots (agents) originating from different locations need to navigate through a grid-based environment to pick up products from certain locations and deliver them to specified destinations. The program uses the Conflict-Based Search (CBS) algorithm, which combines low-level pathfinding with high-level conflict resolution to find a solution.

References:

Sharon, Guni, et al. "Conflict-based search for optimal multi-agent pathfinding."ArtificialIntelligence219 (2015): 40-66.

Team Details:

1. Ajay Paliwal - 21EE30039

2. Gundeti Manoj - 21CY10014
3. Nikhil Mishra - 21CY10023
4. Shruti Srivastava - 21EE30025