

# **Path Planning for Automatic Vehicle Parking Problem Based on MAPF**

**Jialing Li,**

jialing.li@studio.unibo.it

**Molin Zhang,**

molin.zhang@studio.unibo.it

## **Abstract**

The integration of autonomous driving technology has brought about a new level of convenience in how we move and park vehicles. However, as the number of self-driving cars increases, the task of parking multiple vehicles simultaneously becomes a more intricate challenge. This project aims to tackle this challenge by developing a solution for the multi-agent path planning problem in automatic vehicle parking. The objective is to create a system that allows multiple autonomous vehicles to find the optimal path from their starting location to their designated parking spot while avoiding collisions with each other. To achieve this, the project proposes a combination of A\* algorithm and Conflict-Based Search (CBS) techniques to ensure efficient path planning and safe navigation.

## **1. Introduction**

The Multi-Agent Path Finding (MAPF) is a problem in multi-agent planning that involves finding a collision-free path for a group of agents from their starting positions to designated goals. The goal is to find an optimized path that minimizes the number of time steps taken

for all agents to reach their final destinations. This problem is a multi-agent generalization of the traditional pathfinding problem and is closely related to graph theory principles.

The MAPF problem, in our case, is defined by an undirected and unweighted graph  $G = (V, E)$  and a set of  $k$  agents, each with a starting and a goal location. Time is divided into discrete time steps, and each agent can either move to an adjacent vertex or wait at their current location. The objective is to find a set of conflict-free paths that allow all agents to reach their goals while minimizing the total time taken. Conflicts arise when two agents occupy the same vertex or traverse the same edge in opposite directions. The solution must avoid these conflicts and find the optimal path for each agent.

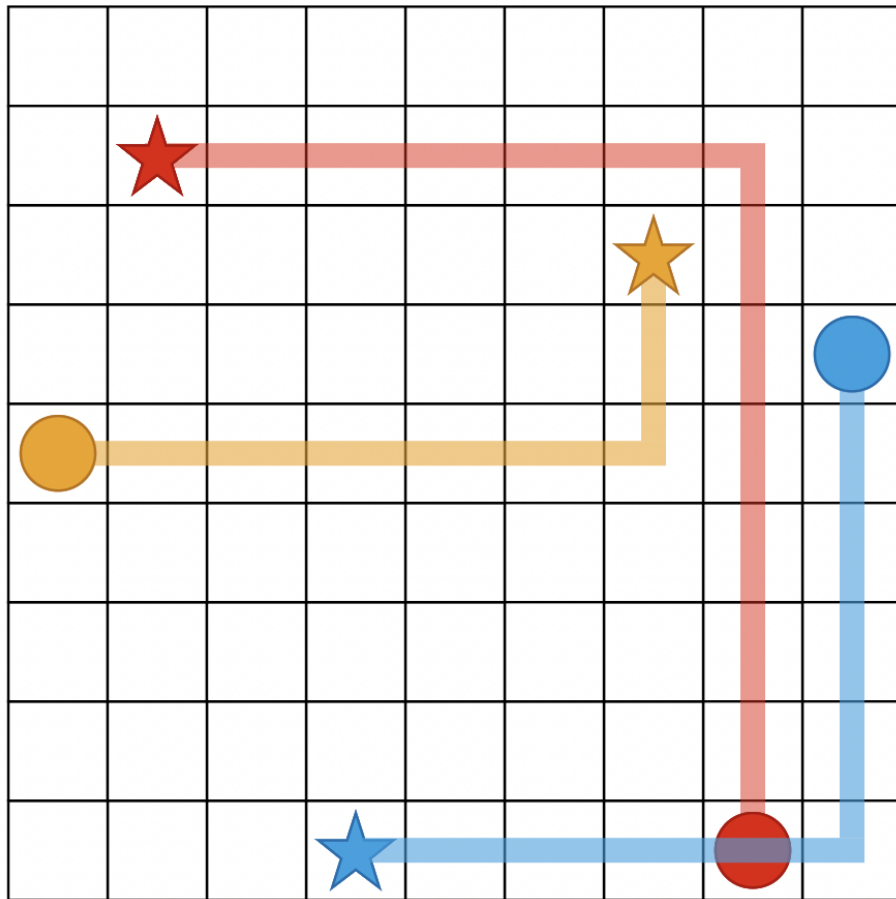


Figure 1

Example of Multi-Agent Path Finding environment

## 2. Theoretical Research

### 2.1 Dijkstra algorithm

The Dijkstra algorithm is a popular and widely used algorithm for finding the shortest path between a source node and a target node in a graph. It works by starting at the source node and visiting the nodes in the graph in order of their increasing distance from the source node. At each step, the algorithm selects the node with the smallest distance that has not yet been visited and updates the distances of its unvisited neighbors.

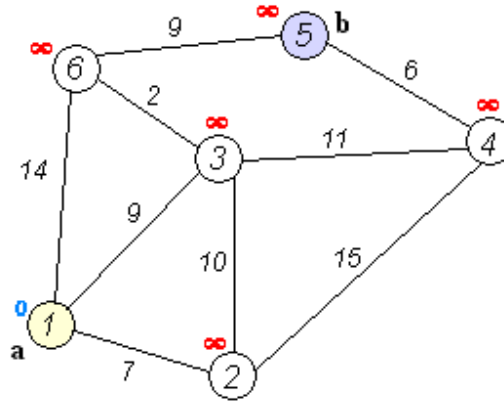


Figure 2 Example of Dijkstra's algorithm

Figure 2: Dijkstra's algorithm to find the shortest path between a and b. Mark visited (set to red) when done with neighbors.

In our project, we apply Dijkstra's algorithm to the MAPF problem by using it to find the shortest path from the starting point of each agent to its goal point. We use the algorithm to traverse the entire map and record the step length required to take the shortest path from any location on the map to any target location. By doing so, we can determine the optimal paths for each agent that minimize the total time taken for all agents to reach their goal points.

## 2.2 A\* algorithm

The A\* search algorithm is a widely used method in best-first search research. Proposed by Peter Hart and others, A\* is based on Dijkstra's algorithm. Dijkstra's algorithm explores the nodes around the starting node one by one until the target node is found, while A\* incorporates a more direct heuristic, which is the estimated distance between the current node and the target node, to guide the search towards the target. A\* evaluates each node by combining the cost  $g(n)$  of reaching the node with the cost  $h(n)$  from the node to the target.

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

Also, A\* is frequently used for multi-agent path planning in dynamic environments. Agents plan their paths based on the available map information, but may need to replan if the actual environment information detected by their sensors differs from the map. The replanning process involves generating a new path from the current position to the target, and repeating until the target is reached or deemed unreachable. This process ensures that the agents navigate the environment effectively and avoid any obstacles or changes in their path.

## 2.3 Conflict Based Search (CBS)

The use of path planning and motion control by individual agents can effectively avoid obstacles in their environment, however, it creates a new challenge when multiple Automatic Vehicles are present in the same scene. With an increase in the number of vehicles, congestion, and deadlocks can easily occur, leading to road traffic issues. To address this, the Conflict-Based Multi-Agent Path Search method was introduced as a solution. This approach aims to resolve conflicts between multiple agents and ensure efficient and safe navigation in shared environments.

The CBS (Conflict-Based Multi-Agent Path Search) process consists of two search processes: a bottom-level search process and a high-level search process. The bottom-level process finds an effective path for each robot while the high-level process checks for

conflicts between paths, generates new branches, and selects the branch with the lowest cost value for the bottom-level search process.

The high-level search process has two main functions:

1. Check for conflicts between paths and generating new branches,
2. Select the branch with the lowest cost value for the low-level search process.

Conflicts between paths can occur when two paths occupy the same node at the same time or change positions at the same time.

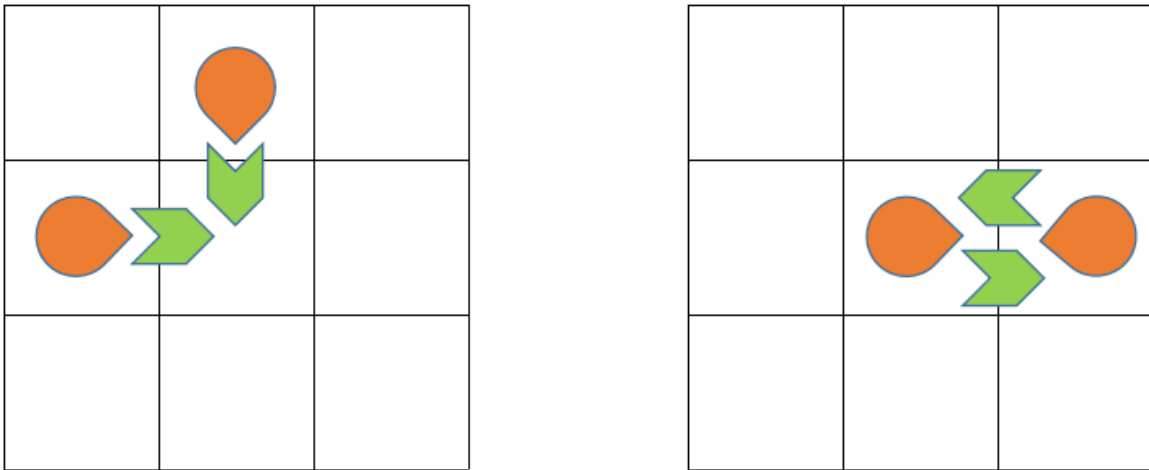


Figure 3 Two types of conflict

In case of a conflict, two branches need to be generated: one where the first robot cannot enter the node at time  $n$  and another where the second robot cannot enter the node at the same time. The branch with the lowest cost value is then selected for the low-level search process.

The low-level search process is similar to other path planning methods, such as Dijkstra or A\*, but it considers additional constraints such as conflicts generated in the high-level search process and the situation of waiting in place. Time is also added as a dimension to the search process, with the search area expanding each time the time increases by one unit length.

By continuously running the high and low search processes, better results can be obtained in a shorter amount of time, especially when the problem complexity is low.

### 3. Implementation

The multi-agent grid map pathfinding system consists of the following components, as illustrated in Figure 4:

This is the main code for the car parking problem. It uses the A\* algorithm and CBS (Conflict-Based Search) algorithm to find the solution to the problem.

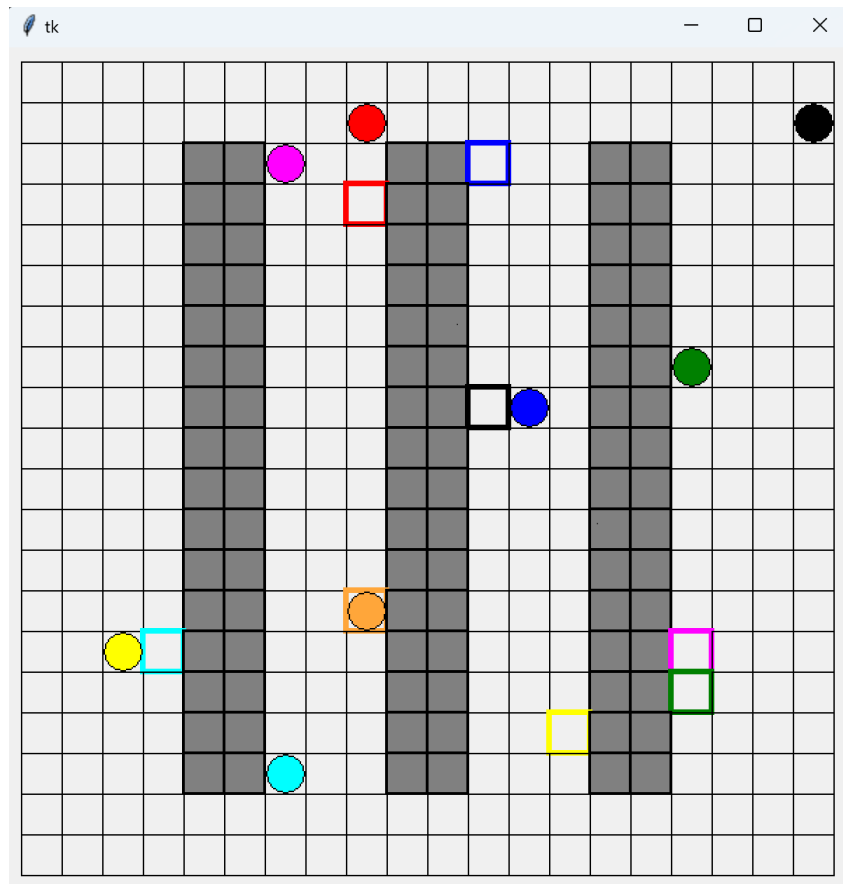


Figure 4 Initial simulation state

### 3.1 Simulation Structure

**Initialization of the environment:** The simulation in this project uses the grid method to model the environment. This method divides the agent's path-planning environment into 2D grids, each grid serving as a unit with unchanging properties. There are three types of grid units in the grid method: obstacle grids (*OBSTACLES*), free grids (*FREE\_SPACE*), and grids occupied by the car.

**Visualizing the maze:** An Interface class is used to visualize the maze. The *draw\_world* method of the Interface class is used to draw the map.

**Initializing the cars:** The Car class is used to initialize the cars. The *car\_init* method of the **Car** class is used to set the starting position and final destination for each car. The positions of the starting and final destinations for each car are saved in separate lists.

### 3.2 Path-finding unit

The pathfinding unit provides the core algorithm of pathfinding for our entire system. It solves the problem of finding paths for multiple agents from their initial positions to a common goal location while ensuring that the paths do not interfere with each other, i.e. there are no conflicts.

The algorithm consists of three main steps:

1. **Preprocessing:** A heuristics table is computed using Dijkstra's algorithm that stores the cost of moving from a cell to the goal location.
2. **Constraint checking:** A constraint table is built that stores constraints between agents, such as the timing and location of their movements.
3. **Searching:** A search algorithm is used to find the paths that satisfy the constraints and minimize the sum of the costs of moving from each agent's initial location to the goal location.

Our code uses several data structures, such as lists, dictionaries, and heaps (priority queues) to store and manipulate the data needed for the algorithm. The `heapq` module provides functions for working with heaps. The `collections` module provides alternative implementations of some built-in types.

### 3.3 A\* solver

**Initialization:** The algorithm starts by putting the starting node into the closed list, which is a list of nodes that have already been expanded or checked.

**Checking Adjacent Nodes:** The algorithm then checks the 4 adjacent child nodes of the starting node. These nodes are potential candidates for the path.

**Putting Nodes into the Open List:** The child nodes are then put into the open list, which is a list of nodes that need to be expanded or checked. The nodes are arranged in order of their F value, which is a cost metric calculated as the sum of the cost of reaching the node (G value) and the estimated cost of reaching the goal from that node (H value).

**Selecting the Next Target:** The node with the smallest F value in the open list is then selected as the next target for expansion.

**Expansion:** The selected node is removed from the open list and compared with the end node. If it is not an end node, it is put into the closed list and its adjacent child nodes are put into the open list in order of their F values.

**Repeat:** This process is repeated until the end node is found or the open list is empty.

**Path Search Complete:** If the end node is found, the path search is complete, and the algorithm calculates all the nodes on the path and returns the result.

### 3.4 CBS Solver

The algorithm works as follows:

**Initialization:** The CBS algorithm starts by creating a tree of subproblems, where each subproblem involves a subset of the agents. Each subproblem is assigned a priority based on the number of conflicts it contains

**Prioritization:** The subproblems are explored in a prioritized order, with the subproblem with the highest priority being explored first.

**A\* Search:** For each subproblem, the A\* algorithm is used to find paths for each of the involved agents that do not conflict with each other. The A\* algorithm maintains a priority queue of nodes to visit, and uses the estimated cost of reaching the goal from each node to determine the priority.

**Update Constraints:** The CBS algorithm updates the constraints for each subproblem based on the paths found by the A\* algorithm. This ensures that the paths found for each agent do not conflict with each other.

**Repeat:** The CBS algorithm continues to explore subproblems, updating the constraints for each subproblem and using A\* search to find paths for the involved agents, until a solution is found or all subproblems have been explored.

**Solution:** When a solution is found, the CBS algorithm returns the paths for each of the agents, which represent a feasible solution to the multi-agent pathfinding problem.

## 4. Results

In the simulation, 8 cars were placed in a parking lot environment with random starting positions and target parking spots. To replicate real-life parking scenarios, it was specified that the cars could only park on either side of the static obstacles represented by the grid. The outcome of the simulation included the starting coordinates and target coordinates for each car. Also, we compute the total cost taken by all the agents. The results were visualized for better understanding and analysis.

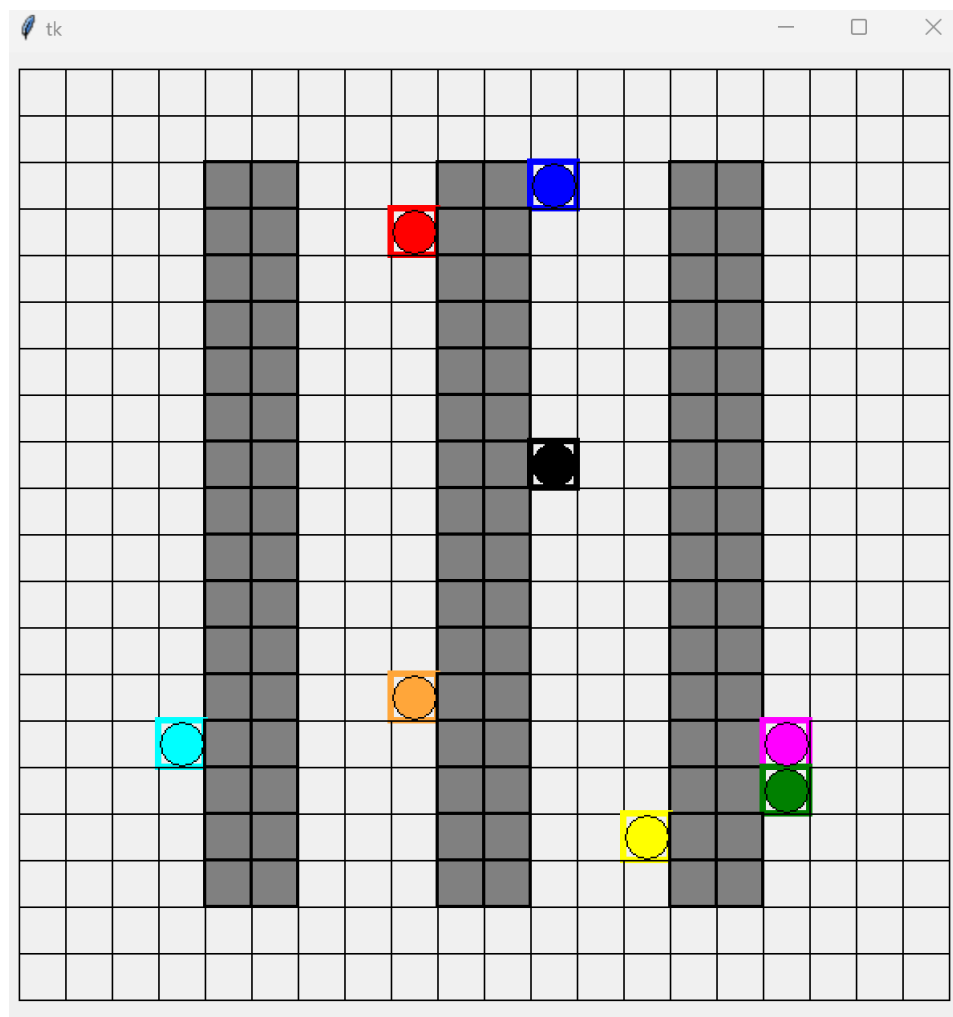


Figure 5 Final simulation state

## 5. Conclusion

The Multi-Agent Path Finding (MAPF) problem presents a major challenge in the development of automated vehicle parking systems. The solution proposed in this project, which integrates the A\* algorithm and Conflict-Based Search (CBS) techniques, offers a promising approach to overcoming this challenge.

In our project, the Car class employs the A\* algorithm to determine the optimal path from the starting location to the designated parking spot for each car, while the CBS-Solver uses the CBS algorithm to coordinate the paths of all vehicles and avoid collisions. This combination of algorithms ensures efficient path planning and secure navigation of multiple autonomous vehicles in shared parking environments. Further research and experimentation are required to validate and improve the performance of the proposed solution.

However, the method also has limitations such as slow performance in crowded or highly populated environments. In future developments, algorithms such as ICBS and CBSH could be employed to address these limitations.

## 6. Reference

[1] Ko-Hsin Cindy Wang and Adi Botea. Fast and Memory-Efficient Multi-Agent Pathfinding[C]. Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, 2008:380-387.

[2] Peter Hart,Nils Nilsson,Bertram Raphael.A Formal Basis for the Heuristic Determination of Minimum Cost Paths[J].IEEE Transactions on Systems Science and Cybernetics,1968,4(2):100-107.

[3] Ian Millington.Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology)[C].Morgan Kaufmann Publishers Inc.,San Francisco, CA, USA,2006.

[4] Silver D.Cooperative pathfinding[J].In Young, R. M.,and Laird, J. E., eds.,AIIDE,AAAI Press,2005:117-122.

[5] Jeremy,Joshua,Christopher,Natalya,Shopf and Barczak et al.March of the Frobnars: Simulation and Rendering Massive Crowds of Intelligent and Detailed Creatures on GPU[J].Advances in Real-Time Rendering in 3D Graphics and Games Course-SIGGRAPH,2008:52-101.

[6] Jansen R,Sturtevant N.A new Approach to Cooperative Pathfinding[C].In AAMAS'08: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems,2008:1401-1404.

[7] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

[8] H. Lee, J. Motes, M. Morales and N. M. Amato, "Parallel Hierarchical Composition Conflict-Based Search for Optimal Multi-Agent Pathfinding," in IEEE Robotics and Automation Letters, vol. 6, no. 4, pp. 7001-7008, Oct. 2021, doi: 10.1109/LRA.2021.3096476.

[9] S. Biswas, S. G. Anavatti and M. A. Garratt, "Particle swarm optimization based co-operative task assignment and path planning for multi-agent system," 2017 IEEE Symposium Series on Computational Intelligence (SSCI), Honolulu, HI, USA, 2017, pp. 1-6, doi: 10.1109/SSCI.2017.8280872.