Multi Agent Path Finding

In multi-agent pathfinding problems(MAPF) we are given a set of agents each with respective start and goal positions. The goal is to find collision free paths from the start to goal and optimise it.Solving the MAPF problem has many applications, including improving traffic at intersections, search and rescue, formation control, warehouse applications, and assembly planning.

Problem Definition

The *multi-agent pathfinding* (MAPF) problem consists of a graph and a number agents, Here we are considering a warehouse represented as a 2-D grid of size n × m and a set of Robots, each with a unique source state to pick up items from these designated places, and a unique destination state to deliver those in these desired locations, and finally the robots go to the End location. Each location (Li) on 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 -
Source location (P1- P3) : Source denotes pick-up location
Destination location (D1-D3) : destination denotes drop location
temporary storage location (TS1-TS4) : Temporary storage location denotes the place where robot can keep some items
Obstacle (black square) : Obstacle represents a location where no robot can move
Rest of the cells are considered as normal cells.
A robot can move at most one cell (either vertically or horizontally) at a time step, a normal cell can be occupied by at most one robot. Source, destination, temporary storage locations can accommodate multiple robots simultaneously.
Let there be k number of robots and r number of tasks.
The task is to develop a work schedule or find paths for all Robots from their source states to their End states through the Destination state where they have to deliver the items that **minimizes the time to complete all tasks**, under the above constraints.

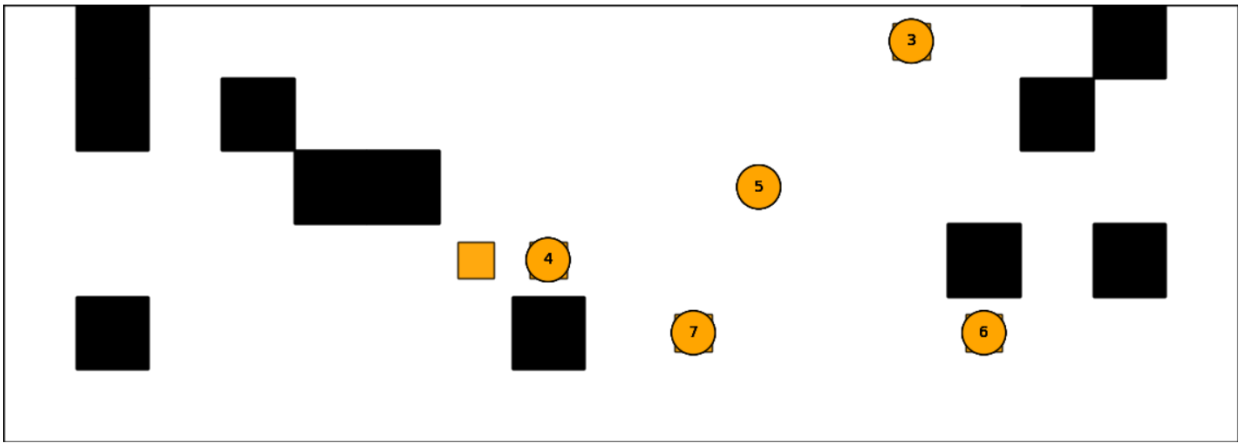
CBS Algorithm

The conflict based search algorithm(CBS) solves MAPF by decomposing it into a large number of constrained single-agent pathfinding problems. CBS decomposes the problem into a large number of constrained single-agent pathfinding problems.
CBS solves the problems by finding paths that satisfy the constraints. If any point in the path is occupied by more than one agent, it is a conflict. CBS resolves conflicts by adding new constraints.

CBS searches a tree called the constraint tree (CT). A CT is a binary tree. Each node N in the CT consists of:

1. A set of constraints which belong to a single agent. The root of the CT contains an empty set of constraints. The child of a node in the CT inherits the constraints of the parent and adds one new constraint for one agent.
2. A set of paths, one path for each agent which must be consistent with the constraints of the agent.
3. The total cost of the current solution. This cost is referred to as the f-value of node N.

A node in the CT is a goal node when the set of paths for all agents has no conflicts. The high level performs a BFS on the CT where nodes are ordered by their costs. In this implementation, ties are broken in favor of CT nodes whose associated solution contains fewer conflicts. Further ties were broken in a FIFO manner.
The low-level search of CBS is implemented through A* which handled the constraints as follows. Whenever a state is generated where v is a location and t a time step and there exists a constraint in the current CT node, this state is discarded. The heuristic used is the shortest path in the spatial dimension, ignoring other agents and constraints.
The low level search returns the shortest path for each agent. Once the path is found it is



Conclusions

We conclude that the performance of each of the known algorithms depends greatly on problem features such as: density, topology of the map, the initial heuristic error and the number of conflicts encountered during the CBS solving process.In this paper, we introduced MAPF, an approach that makes use of a simple temporal network to postprocess the output of a MAPF solver from AI in polynomial time to create a plan-execution schedule that can be executed on actual robots. Many interesting extensions of our approach are possible. For example, we intend to make it works for 483 user-provided safety distances. We also intend to extend it to take additional kinematic constraints into account, such as

the maximum accelerations important for heavy robots. Finally, we intend to exploit slack for replanning (which has not been implemented yet) and creating a hybrid between online and offline planning. For example, it is especially important to monitor progress toward locations that are associated with vertices in the simple temporal network whose slacks are small. Robots could be alerted of the importance of reaching these bottleneck locations in a timely manner. If probabilistic models of delays and other deviations from the nominal velocities are available, they can be used to determine the probability that each location will be reached between its earliest and latest entry time and trigger replanning only if one of these probabilities become small.