Exercise 3
Implementing a deliberative Agent

Group 30 : Jinbu Liu, Mengjie Zhao

October 25, 2016

1 Model Description

1.1 Intermediate States

One state has following major properties:

Table 1: Properties of a state

Name of Property	Description
currCity	indicating current location of the agent.
tasks	containing all available tasks.
maxWeight	indicating the maximum weight the agent can carry.
disToGoal	distance to agent's goal.
disToInit	distance to initial city.
cost	sum of disToGoal and disToInit.
fromParent	information containing parental state of current state.
taskStates	indicating states of tasks.

1.2 Actions and Goal State

There are three types of actions:

move: move to a neighbor city.

delivery: delivery task to current city or a neighbor city.

pick-up: pick up at current city or a neighbor city.

We obtain next state by firstly get an copy of current state. After that, if the action type is move, we update current city of the copy. If the action type is delivery, we update the current city and task states then reduce weight of load. If the action type is pick-up, we update current city and task states then increase weight of load. So the copy is the next state of current state.

When all tasks of a state are delivered, then the state is considered as a final state. We define `isGoalState` method to check if all the tasks have been delivered.

2 Implementation

2.1 BFS

To implement BFS, we create a set and queue, and store the initial state in the queue. If the queue isn't empty and a goal state isn't found, we dequeue the first state in the queue, if the state is the goal state, then our goal is achieved, otherwise, we mark the state as a checked one, if the set doesn't contain the state, we add the state into the set and find the neighbor states of current state, and enqueue unchecked ones of these states into the queue. We will repeat the above execution until the goal state is found or the queue is empty which means no goal state exists.

2.2 A*

The procedure of A* algorithm is very similar to that of BFS, and the mainly difference is that before we dequeue a state from the queue, we need to sort the states in the queue by their cost, so the state with the lowest cost will return from the dequeue execution. The cost of states is computed when these states are generated, and the cost is the sum of two part. One part is the distance from the current state to the initial state and the other part is the distance from the current state to the goal state, which will be illustrated in the next section. Another difference between BFS and A* is that if the current checked state is already in the set and the state has lower cost comparing to the same state in the set, we also enqueue the neighbor states of the current state into the queue.

2.3 Heuristic Function

When implement A* algorithm, the most important part is to design a good heuristic function which is to estimate the distance from current state to the goal state in this lab. If the estimated distance is larger than the optimal distance, the algorithm runs fast but the optimal result isn't guaranteed. If the estimated distance is smaller than the optimal distance, the optimal result is guaranteed but the algorithm runs runs relatively slow. To balance the optimality of the result and search efficiency, we need to make the estimated distance close to the optimal distance, and we use the random method to achieve it. To reach to a goal state, some actions need to be applied on the current state, including many pickup and delivery actions. But we don't know the optimal order of these actions, so we shuffle the order of these actions to get many different orders and computed the distances to the goal state if the state transfers follow these order actions, then we save the shortest distance of these distances. Apparently, the this distance is always larger or equal than the optimal distance, so in order to obtain the optimal result, we use the half of this distance, which is expected to be smaller than but close to optimal distance, as the distance to that goal state. Using this heuristic, A* algorithm can run relatively fast and good enough result is guaranteed.

3 Results

3.1 Experiment 1: BFS and A* Comparison

- Optimality: Within one experiment, the optimality means that after delivering all tasks, the traveled distance should be the smallest. In BFS algorithm, we set the distances between current state to next states is the same. So BFS gives the smallest number of actions required to reach the goal state instead of the shortest distance, however, normally a small number of actions is roughly equivalent to small cost. So a good result can still be guaranteed. For A*, if we design a good heuristic, it is

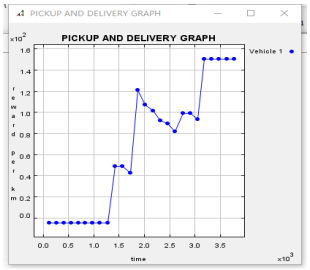


Figure 1: BFS bad

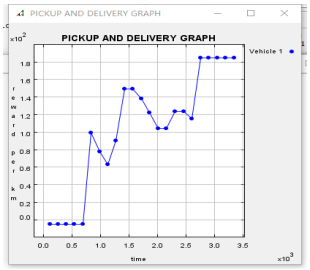


Figure 2: BFS good

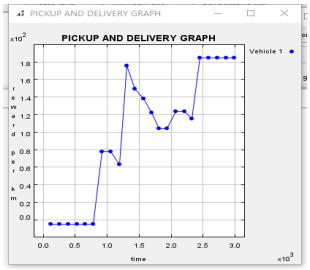


Figure 3: A*

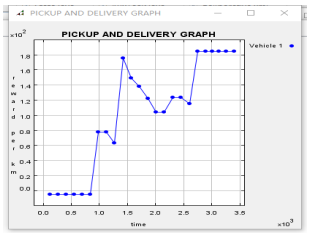
guaranteed that we will get our good enough result. As shown in above figures (# of tasks = 6), BFS can give good or bad results while A* always gives good enough or optimal result.

- Efficiency. BFS has much longer planning time than A*. When delivering 6 tasks, BFS consumes 49.151 seconds while A* consumes 0.878 seconds. In one minute, A* can finish 11 tasks while BFS can finish 6 tasks.
- Limitation: BFS is slower than A* and the optimal result may not be found, however, the implementation is relatively simple. Implementation of A* is more complex as it is not easy to find a good heuristic. But once we found a good heuristic, the result is good enough.

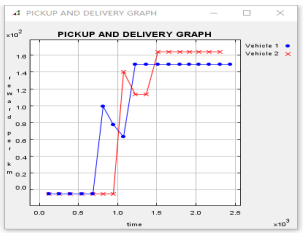
3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting and observations

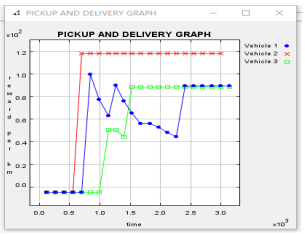
The figure in appendix shows the multi-agent experimental results of BFS and A* with total number of tasks is 6. For a multi-agent experiment, the agents may have conflicts among their plans. For example, it is possible that when a agent arrived at a city that anticipated possessing a task, however, the task is already pick-up by other agents. In this case, the original plan is canceled and the agent will firstly save all existing delivered tasks then re-generates a new plan, with current attribute currCity to be current city.



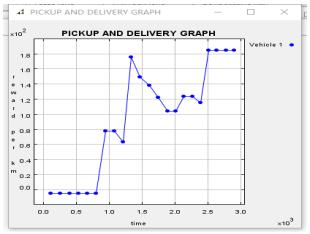
(a) BFS, 1 agent



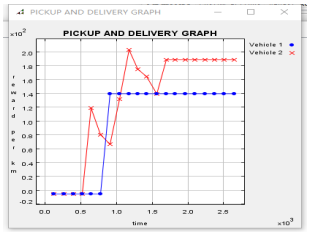
(b) BFS, 2 agents



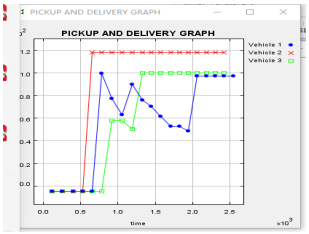
(c) BFS, 3 agents



(d) A*, 1 agent



(e) A*, 2 agents



(f) A*, 3 agents

Figure 4: Multi-agent experimental results