Exercise 3
Implementing a deliberative Agent

Group 10 : Rebecca Cheng, Eddie Wang

October 23, 2018

1 Model Description

1.1 Intermediate States

We represented each state as a class with many properties and methods. The properties of each state include: the current city that a vehicle is at (***City*** *currentCity*), available tasks left for pickup (***TaskSet*** *availableTasks*), task set of items already picked up and to be delivered (***TaskSet*** *tasksToDeliver*), total cost so far (**double** *costSoFar*), capacity left (**double** *capacityLeft*), previous state (**State** *prevState*), heuristic cost (which was only used for A*) (**double** *heuristicCost*) and the action taken at the previous state to get to the current state(***Action*** *actionToGetHere*). However to check if two states are equal, all properities are taken in consideration except for *actionToGetHere, heurisiticCost, prevState,* and *costSoFar*.

1.2 Goal State

We represented the goal state by checking if both *availableTasks* and *tasksToDeliver* are empty. If so, that means that there are no more tasks left to pick up and no more tasks left to deliver, so the vehicle has reached its goal! Since we kept the *actionToGetHere* property at each state, at the final state we loop through each preceding state to get the sequence of actions used to come up with a final plan.

1.3 Actions

The *getSuccessorStates* method in the State class determines the successor states and their transition types from each state. At each state, there are three possible actions: Pickup, Deliver, or Move. If the vehicle performs a Pickup at a state for task t, then we create a new state at the same city with $availableTask_{new} = availableTask_{old} \backslash \{t\}$ and $tasksToDeliver_{new} = tasksToDeliver_{old} \cup \{t\}$. If the vehicle performs a Delivery at a state for task t, then we create a new state at the same city with $tasksToDeliver_{new} = tasksToDeliver_{old} \backslash \{t\}$. If the vehicle performs a Move to city c, then we create a new state at city t with no changes to *availableTasks* or *tasksToDeliver* but we update the *costSoFar* according to the distance to the new city.

2 Implementation

2.1 BFS

To implement BFS, we first create a queue and store an initial state in the queue. Every time a State is dequeued, we check that the queue is non-empty, and the dequeued State has not been visited to prevent cycles.
If the State is a goal state, a plan with a shortest list of legal actions would be returned. Otherwise, successors of that States would be added to the queue. This process is repeated until a goal state is reached, or the queue is empty which means all possible states were explored and no goal state was found. This implementation follows the pseudo-code on the BFS with cycle detection slide.

2.2 A*

The A* algorithm was implemented in a similar manner mentioned in the exercise slides; by keeping a priority queue that was sorted by the heuristic function, sorting successor states by heuristic, and merging the (sorted) priority queue with (sorted) successor states. A HashMap *map* was also created to check if a state (based on equality as defined above) has already been visited to prevent infinite cycles.

2.3 Heuristic Function

For our heuristic function, we let $g(n) = costSoFar$ and $h(n) = max(deliverCost, pickupAndDeliverCost)$ where $deliverCost$ is calculated by looping through all the $tasksToDeliver$ and obtaining the max delivery task distance from the current city. $pickupAndDeliverCost$ is calculated by looping through all the $availableTasks$ and obtaining the max distance to pickup and delivery a task from the current city. If there are no tasks left to deliver or pickup, h(n) = 0. Thus, $f(n) = g(n) + h(n)$.

The heuristic function is optimal because it underestimates the true final cost, since the cost to deliver or pickup and deliver the furthest task possible out of a set of tasks. Therefore it is always upper bounded by the true sum of costs of every task in a task set. Thus by theorem, the heuristic function is optimal.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

For this experiment, we set the topology to be Switzerland, random seed to be 23456, and home city as Lausanne. Note that in all our tests, we use Macbook Pro 2018 2.2 GHz Intel Core i7.

3.1.2 Observations

Tasks	Total Time (BFS)	Total Cost (BFS)	Total Time (A*)	Total Cost (A*)
6	0.023s	6900	0.105s	6900
7	0.067s	8150	0.361s	8050
8	0.219s	8550	1.095s	8550
9	0.741s	9050	3.236s	8600
10	2.445s	9250	10.87s	9100

We observe that the BFS algorithm does not always arrive to the optimum solution unlike A*. This is because in our implementation, BFS only searches for the shortest plan length that reaches the final state, but it may not be optimal. However, if BFS were configured to find an optimal solution as well, we anticipate that A* will find the solution faster. This is because A* ensures that the next visited node is the one "closest" to the final goal state, so it will evaluate nodes that are "further" away from the goal state with less priority.

Overall, since our BFS only finds the shortest solution (that may not be optimal) but A* has to search through more paths to find the optimal solution, BFS performs better time-wise compared to A*.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

For this experiment, we set the topology to be Switzerland, random seed to be 23456, and home city as Lausanne, Zurich for 2-agents case and Lausanne, Zurich, Bern for 3-agents case.

3.2.2 Observations

BFS Agents	A* Agents	Tasks	Total Distance	Total Cost	Time
2		6	2170	10850	0.03s
2		9	2790	13950	0.859s
3		6	2850	14250	0.035s
3		9	3160	15800	1.172s
	2	6	2170	10850	0.032s
	2	9	2740	13700	0.833s
	3	6	2420	12100	0.035s
	3	9	3150	15750	0.863s

We can see that performance in terms of time and cost does not necessarily increase with multiple agents. This is because each deliberative agent has no information about other deliberative agents (and what they picked up/delivered) in the map. Thus, one agent may go to a location to pick up a task that has been already picked up by another agent and waste travel costs.