Excercise 3
Implementing a deliberative Agent

Group №: Bejjani Jean Marc, Bonnesoeur Maxime

October 23, 2018

1 Model Description

1.1 Intermediate States

We define a state in our model as the city our agent is located in and the status of each tasks it needs to fulfill. A task can be : *NOT_PICKED*, *PICKED* and *DELIVERED*.

1.2 Goal State

.

The goal of our model is to compute the best route in order to deliver a set of task while minimizing the cost of transportation. This means our goal state is reached when all the tasks are delivered. Hence, a goal state is defined by all the tasks being in the status *DELIVERED*.

1.3 Actions

The actions of our model are :

- Moving from one city to its neighbor
- Picking up a task if the sum of the weight of the task and the actual weight carried by the vehicle is inferior to the maximal capacity of our vehicle
- Delivering a task

2 Implementation

Each node of our model is defined by an object of the class *State* which is used for both the A* and the BFS algorithm. This class describes a state with two variable : the city our vehicle is actually in, *myCity* and a table *taskStatus* which is giving us the current status of each task.
This class has functional attributes like the previous state of the current state : *pState*, the total distance to reach our current state : *travelCost*, the heuristic value of the state that will be used in the A* algorithm, *heuristic*, the total weight carried by our vehicle : *weight* , and an id to identify our states *id*.
Finally, this state function implement a method to compute the heuristic of the state, the plan generator which computes the Plan from the best node, and the method descendant which returns all the next states of our current state.

2.1 BFS

For the Breadth First Search algorithm, the implementation is the following one. First we create our initial state with all the tasks in the status NOT_PICKED. Then, we are calling the function *descendant*() to generate the queue. We also added a function that analyze if a node already computed will be a next State of our System. If this is the case, we will compare the *travelCost* to reach this state with the one already stocked in memory. If the travel cost is superior, we will not generate the descendant of this state. If the travel cost is inferior, we will replace our old state in memory with our current state

2.2 A*

The A* defines a TreeMap object *scoreMap* that links a queued State with a *double* key defined by the score of this state. The score is the sum of the heuristic value and the travel cost of the State from the initial state. This TreeMap sorts the queue so we can get extract the State with the lowest score to discover its descendants. We also define a HashMap for storing the explored states. We explore the search space by defining a head state that will move across the states. We iterate in a while loop that breaks in the condition to have all the tasks delivered. In this loop, we start by putting the head state on the top of the queue and removing the corresponding state from the queue. We then test if the head has not been explored yet and if so we add the descendents of this state to the queue that will take care of sorting them. We then check if the head state has all the tasks delivered and if so we break the loop and generate the plan from the final State. Else we add the head state to the HashMap and we reiterate.

2.3 Heuristic Function

In order to have the optimal result for the A* algorithm, we need to find an heuristic that will guide us to the optimal path.
We decided to pick the following heuristic :

$$heuristic = \frac{\sum_{n=1}^{NOT_PICKEDtasks}(distanceToPickupcity + distanceToDeliveryCity) + \sum_{n=1}^{PICKEDtasks} distanceTodeliverycity}{Numberoftasks}$$

With this heuristic, we are able to determine which state will be the closest to a large number of task to pick and/or to deliver. This way, we will be able to do a large number of tasks in a restrained zone and hence, to reduce our travel distance.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We are running our programs (A* and BFS) with the same seed (rngSeed : 5667).

3.1.2 Observations

As expected, our programs are way more efficient that naive plan and are able to deliver the optimal plan for 3 and 5 tasks. The BFS is always giving us the optimal path but the A* can deliver up to 9 tasks and is using much less memory than the BFS.

Number of tasks	Time (ms)		Number of states evaluated		Reference (naive plan)	Optimal cost
	A*	BFS	A*	BFS		
3	4.0	3.0	42	271	1270.0	1070.0
5	12.0	4204.0	305	326011	1740.0	1150.0
7	25.0	-	1102	-	3160.0	1430
9	40.0	-	1802	-	4230.0	1640.0

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

We are using the same configuration as before (rngSeed : 5667) while generating 3 agents at the same time while using the A* algorithm.

3.2.2 Observations

We implemented the function *plancancelled*() which is used if a task is not available anymore. In this program, we are just saving the current carriedTask and we are using it to recompute the initialization of the system.

number of tasks	Agent 1	Agent 2	Agent 3	Total distance	Total distance for 1 agent	Naive agent
3	1070	900	1070	1960	890.0	1590.0
5	1150	940	1110	3200.0	1220.0	3090.0
9	1640	1670	1300	3250.0	1710.0	4420.0