Excercise 3
Implementing a deliberative Agent

Group №: 11: Marcel Dubach, Maxime Gardoni

December 8, 2020

1 Model Description

The implemented search algorithms perform a graph search on the graph defined by the problem statement. The agent starts at the initial node which corresponds to the state where it has no task on board of its vehicle and several tasks that are to be picked up and delivered. It then decides which task it should pick up or deliver first, in order to have a minimal total path length to pickup and deliver all given tasks.

1.1 Intermediate States

Each node of the graph corresponds to a state of the vehicle. In our implementation the state contains the following information:

- Current City: the city in which the vehicle is currently located
- Tasks on board of the vehicle: the set of tasks that have already been picked up but not yet delivered
- Available tasks: the set of tasks that are yet to be picked up (and delivered).
- Action of the parent node: Each node stores the action taken at the preceding node.
- Cost: the cost to get from the initial node to the current node
- Parent State: For each state we store the information about the preceding state. This allows us to calculate heuristic and cost in a recursive way.

1.2 Goal State

There are several goal states in the graph. They correspond to states where all tasks have been delivered, meaning that both, the set of tasks that are on board and the set of available tasks, are empty. Note that in the present scenario, there are an identical number of actions to be taken to get to any of the goal states (in fact, for n available tasks, there are $2 \cdot n$ actions to be taken, that are "pickup" and "deliver" for each of the tasks). This means that all goal states are in the same layer of the graph.

1.3 Actions

At the beginning, as long as none of the n tasks has been delivered yet, the agent has exactly n choices that it can take: either pick up one of the new tasks or deliver one of the tasks that it has already picked up. Later on, the number of possible choices that the agent can take, will decrease and still correspond to the number of tasks that have not yet been delivered.

2 Implementation

2.1 BFS

The set Q of nodes to visit is implemented using a LinkedList of State objects, in order to treat the nodes in an ordered way. The set C of already visited nodes is declared as an ArrayList of State objects, as the order no longer matters. The algorithm treats the first node in Q, removes it from Q and, if necessary, the current node is added to C and its respective child nodes will be appended to Q.
In order to prevent cycles, the algorithm checks on two conditions: Child nodes of the currently treated node will be enqueued only if the current node has either not yet been explored or if the found path leading to it is better than the previous one. In order to make the algorithm more efficient, the program actually checks if the same child nodes (with a different parent state) are already contained in Q. If this is the case, the parent state and action (of the enqued child nodes) are updated according to the better path (and the same chlild node will not be added twice).
As the algorithm gets to a goal state (meaning a state that has empty task sets and no further child nodes), it will add the corresponding state to an ArrayList of goal states. The algorithm stops only when Q contains no more elements, which means that that the tree has been entirely explored. Once all the tree has been explored, we loop over all possible goal states to find the one with the least cost. From that node we will backtrack the optimal path using the information about the parent state and parent action of each of the nodes on the optimal state trajectory. Finally, the optimal sequence of actions is obtained reversing the order of actions, such that the trajectory leads from the initial state to the optimal goal state.

2.2 A*

The A*-algorithm uses similar structures as the BFS. However, the Q object is now a PriorityQueue object, meaning that it will sort all enqueued nodes according to the sum of cost (to get from the initial node to the current node) plus heuristic (to get from the current node to a final node). This algorithm therefore relies not only of the cost to get to each of the states , but also on the heuristic estimation of the cost to get from each state to a goal state. The conditions on the update and addition of child nodes in Q remain similar to the BFS algorithm. which allows to avoid cycles during exploration. However, unlike the BFS algorithm, the A*-algorithm does not explore the entire tree but it will stop as soon as it has found a final node. Indeed, a good choice of the heuristic function should guide the exploration of the tree in a favorable direction.

2.3 Heuristic Function

The heuristic function for a given node n is chosen as follows:

$$h(n) = \max \begin{cases} d(P_i, D_i) \cdot k, & \forall i \in V \\ d(C, P_j) + d(P_j, D_j) \cdot k, & \forall j \in A \end{cases} \tag{1}$$

where P_i and D_i are pickup and delivery city of task i respectively, C is the current city of the vehicle, $d(x, y)$ is the distance between two cities x and y, V is the set of tasks that have been picked up but not yet delivered and A is the set of tasks that have to be picked up and delivered. k represents the cost per kilometer of displacement.
Recall that this function needs to underestimate the remaining cost from node n to any of the goal states.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

The settings used are the default ones. Only the number of task was changed. - Topology: switzerland.xml - randomSeed value is 2345 - cost per km is 5 - capacity is 30 kg.

3.1.2 Observations

Optimality: both methods find the same cost. **Efficiency** : from 5 tasks and more, ASTAR is more efficient. BFS can compute a plan for up to 5 task below 1min, A* can do up to 6 in less than 1min.

N. Tasks	Type	N. Iter	Cost	Time(ms)
1	BFS	3	2700	8
1	A*	3	2700	8
2	BFS	13	3900	8
2	A*	6	3900	7
3	BFS	55	4300	9
3	A*	8	4300	8
4	BFS	208	4300	11
4	A*	2	4300	10
5	BFS	784	5200	38
5	A*	36	5200	13
6	BFS	2620	5200	85
6	A*	25	5200	24
7	BFS	9073	5750	700
7	A*	490	5750	82
8	BFS	28837	5750	7044
8	A*	461	5750	303
9	BFS	97444	7400	119841
9	A*	22377	7400	7272

Table 1: Performance summary of BFS and A*

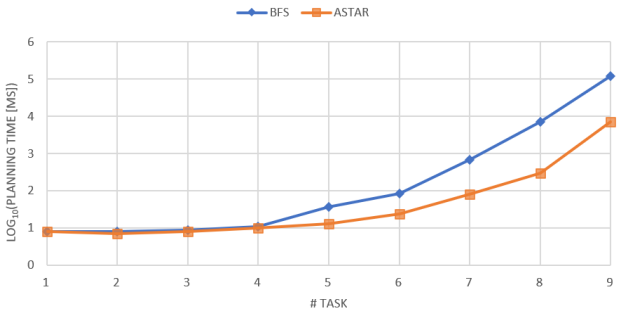

Figure 1: LinLog plot of the computation time

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

Same settings as previous experiment, but multiple vehicle are deployed. All the vehicles have the same cost per kilometer.

3.2.2 Observations

One can observe that the average reward per km decreases with increasing number of vehicles. Some vehicle will fail to gain the planned reward since one or several tasks may have been picked up by another agent already.

nVehicle	nTasks	Average reward per Km
6	1	359
6	3	148
8	1	450
8	3	213

Table 2: Multi vehicle Performance