Excercise 3
Implementing a deliberative Agent

Group №89: Wenuka Gunarathna, Sofia Dandjee

October 20, 2020

1 Model Description

1.1 Intermediate States

The agent is fully aware of its environment. A state therefore comprises of the agent current city, a `logist.topology.Topology.City`, the tasks it is carrying as well as the tasks it has not yet picked up nor delivered, two `List<logist.task.Task>`. A state is also aware of the actions that has led to it with a `List<logist.plan.Action>` and the cost of the resulting plan.

1.2 Goal State

A goal state is a state where there are no more tasks to pick up nor to deliver, that is that the lists of carried and unhandled tasks are empty. This final state can be reached at any city depending on the last task that has been delivered and can result of a different plan depending on the order in which tasks have been delivered.

1.3 Actions

The possible actions of the model can be one of the following depending on the state: deliver / deliver and move / deliver, pickup and move / pick up and move / just move. From a given state, the agent automatically delivers every task he is carrying if any of the task's delivery city corresponds to the current city, as the delivery is cost-free. Then, he can either move to a neighbour city or pick up one or several tasks if the capacity of its vehicule is not exceeded. After having picked up and delivered all possible tasks, he can again move to a neighbour city.

2 Implementation

2.1 BFS

The Breadth First Search (BFS) was implemented with cycle detection. The initial node is built with the vehicle's current city (`getCurrentCity()` method) and current tasks (`getCurrentTasks()` method). We created a Queue `Q` for the nodes to visit and a HashSet `C` for nodes that been visited. Before entering the loop, we add the initial node to `Q` and set the optimal cost to $-\infty$. While `Q` is not empty, we visit the node which is popped from `Q`. If this is a final node, we check that the plan that lead to this state has a lower cost than the one we stored before. If it is, we update the optimal cost and plan. Afterwards, we add the visited node to `C` and add all the states that can be reachable from it to `Q`. That way, the graph is created dynamically during the algorithm. This saves us computational time compared to if we generated every possible states and its successors beforehand. This derived implementation of BFS leads to a full tree search.

2.2 A*

The A* algorithm was also implemented with the Heuristic functions described in the subsections 2.3 to find a final node. With this, we achieve a significant reduction in the number of iterations to find a final node as we explore the most prominent points first. To facilitate some Heuristic functions we came up with a static *TopologyStats* class which holds some statistics about the given topology such as the minimum distance neighbour of each city using a Singleton design pattern in Java.

2.3 Heuristic Function

We have used different heuristic functions to support our A* algorithm as below. For comparison purposes, we are suggesting a naive approach, an inferior approach and two better approaches. In all cases we have to make sure to not overestimate the cost, making the search end at a sub optimal point. In all of the following heuristic functions, it ended up in the best possible plan for the Switzerland topology and the number of visited nodes it took to find the first solution is depicted in Table 1.

Heuristic Function	Iteration count to solve
Naive	6941
Two Step Away	12867
Min Average	3095
Max Distance Between Tasks	1653

Table 1: Iteration count to find the first solution for Switzerland Topology with one Agent and 6 tasks

2.3.1 Naive Heuristic Function

In the Naive approach, we are estimating the cost proportionately to the number of task to be delivered in a given state. To match the figure with the current cost, we multiply it by the minimum distance between two cities which was 40km for the Switzerland topology.

2.3.2 Two Step Away Heuristic Function

This is an extension of the naive approach where we output the exact cost if the state is two steps away from the final state, or else the naive value. However this approach was not performing well as shown in table 1 taking almost 8 more times steps to find a solution. This is mainly because we are using two scales for each scenario, resulting A* search getting mislead about the expected cost. This clearly shows the importance of selecting an appropriate heuristic function.

2.3.3 Min Average Heuristic Function

For this heuristic function, for all not-handled task (not picked up nor delivered), we are taking the average between the minimum neighbour distances of the delivery city and the pickup city of each remaining task and then sum it up to the estimated cost. On top of that, we add the minimum neighbour distances of the delivery city of the carrying tasks.

2.3.4 Max Distance Between Tasks Heuristic Function

This function estimates the cost of a state as the maximum distance from pick up city to delivery city among the remaining tasks to pickup plus the minimum neighbour distances of the carrying tasks deliver cities.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We used the topology of Switzerland. The $MaxDistanceBetweenTasks$ heuristic function was used in the A* for the comparison.

3.1.2 Observations

In both the algorithms, the agent was able to find the same optimal plan with a cost of 6900 (total distance of 1380km) and Figure 1 shows the per km profit of the agent. Therefore to compare the different algorithms, we are looking at the number of iterations (visited nodes) it took to find a result. For BFS it took 12888 steps to find the best optimal solution where A* algorithm with the best heuristic function does also find the same optimal plan in 1653 steps. This is a 780% increase of the efficiency.

Regarding the number of tasks for which we can build a plan in less than one minute, for BFS it was 6 tasks where it took 3 seconds while for 7 tasks it took 285 seconds. For A* with the best heuristic function we found, it took 18 seconds to find a plan for 8 tasks, while for 9 tasks it took 126 seconds (Intel Core i5-5250U). Here we can observe the complexity of the solution increases exponentially with the number of task available and that the A* algorithm significantly reduces the computational time while finding the same optimal plan than BFS.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

Switzerland topology was used with the best A* algorithm with one, two and three agents and 6 tasks.

3.2.2 Observations

In this experiment, agents cannot communicate with each other and have to recompute their plan according to what the other agents have already picked up. Therefore, agents make unnecessary moves as they think there are tasks to pick up whereas it has already been picked up by another agent, which leads to extra costs. With a single agent, the reward per km converges to 180 as shown in Figure 1. However with 2 agents, the average reward per km per agent is lower (80 and 150 for each agent). With three agents, this leads to an even less efficient situation. In our experiment, two agents follow almost exactly the same plan as they start in close cities while the third agent useless. This leads to an even reduced average reward per km per agent (80, 150 and 0 for each agent), as shown in Figure 2. There can be some agent with zero rewards as well due to this extra uncertainty present in the environment.

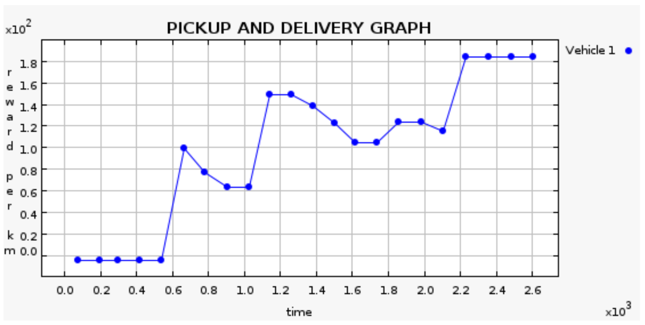

Figure 1: Reward per km function for one deliberative agent present at Switzerland topology

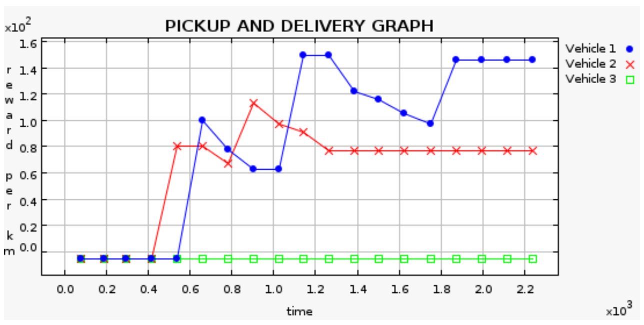

Figure 2: Reward per km function with three deliberative agents at Switzerland topology