Exercise 3
Implementing a deliberative Agent

Group № 38: Andrej Janchevski 309143, Orazio Rillo 313423

October 20, 2020

1 Model Description

1.1 Intermediate States

Let S be the set of all the possible states of our representation, C the set of the cities in the topology and T the set of all the tasks to pickup and deliver before the simulation is launched. We say that a vehicle v has state:

$$s \in S \iff s = (c_L, T_P, T_D), \quad c_L \in C,\ T_P, T_D \subseteq T$$

where c_L represents the current location of v, T_P and T_D are the sets of tasks that v has to (respectively) pickup and deliver. With a combinatorial argument and the binomial theorem it can be shown that:

$$|S| = |C| \sum_{p=0}^{|T|} \binom{|T|}{p} 2^{|T|-p} 1^p = |C|(2+1)^{|T|} = |C|3^{|T|}.$$

1.2 Goal State

In our representation a *goal state* is any state $s = (c_L, T_P, T_D)$ s.t. $c_L \in C$, $T_P = T_D = \varnothing$.

1.3 Actions

Consider an agent having state $s \in S$; call C_N^s the set of the neighbor cities of c_L, $T_P^s \subseteq T_P$ the set of the tasks available to be picked up in c_L and $T_D^s \subseteq T_D$ the set of the tasks carried by the agent that can be delivered in c_L. In state s the agent can choose among $|C_N^s| + |T_P^s| + |T_D^s|$ possible actions i.e. a $moveTo(i)$ action for each $i \in C_N^s$ plus a $pickup(t)$ action for each task $t \in T_P^s$ and a $deliver(t)$ action for each task $t \in T_D^s$.

Concerning transitions, let a be the action taken in state s, leading to s'. If $a = moveTo(i)$, $i \in C_N^s$, then $s' = (i, T_P, T_D)$. If $a = pickup(t)$, $t \in T_P^s$, then $s' = (c_L, T_P \backslash \{t\}, T_D \cup \{t\})$. If $a = deliver(t)$, $t \in T_D^s$, then $s' = (c_L, T_P,\ T_D \backslash \{t\})$.

2 Implementation

2.1 BFS

To implement the BFS algorithm, we employed a tree structure, by introducing a new class: `BFSNode`. It is the abstract representation of a generic planning step. Each node contains information about a particular state reached by the agent, the total cost g paid for reaching the node, the action that has been chosen in that state and, of course, the reference to the parent node. The g cost is updated by taking into account the cost of the parent, adjusted by the cost of the last performed action. Only move actions give a positive contribution, while delivery actions result in a negative contribution: the obtained reward.

In each state, all the possible actions are explored and for each of them a new node is added to the tree only if either (1) the corresponding state has not already been visited or (2) the state has already been

visited and the cost of the new node is lower. This is checked using a `HashMap` mapping visited states to their corresponding nodes. Whenever a goal node is reached, that branch of the tree is interrupted, the goal node is added to a `PriorityQueue` ordered by the g cost of the nodes and the computation continues for the other branches. Eventually, the node at the top of the `PriorityQueue` corresponds to the goal node of the path having minimum cost. From this, we finally infer the optimal plan.

2.2 A*

In the A* algorithm the procedure is very similar. An `AStarNode` has the same attributes of a `BFSNode` plus the h cost, which is an estimate of the minimal cost to pay to reach a goal node. About the algorithm, the only difference is the fact that the nodes of the exploration tree are kept in a `PriorityQueue`, ordered by their f cost, which is the sum of their g cost plus their h cost. In addition, the computation is interrupted as soon as a goal node is found and, indeed, the optimal plan is inferred starting from it.

2.3 Heuristic Function

We defined three different heuristic functions. Let us suppose that $s = (c_L, T_P, T_D)$ is the current state of the agent. Then:

1. The first heuristic is the difference between the cost paid for going from c_L to its nearest neighbor city and the total reward given by delivering all the tasks in $T_P \cup T_D$. This heuristic is admissible as, regardless of the chosen action and unless it is in a goal state, the vehicle would have to travel the distance at least to its closest neighboring city. However, this is not the optimal estimate since the nearest neighbor may not even be in the optimal path to complete the tasks.

2. The second heuristic we defined is the difference between the cost paid for going from c_L to the nearest city in which there is a task to pick up or to deliver and the total reward given by delivering all the tasks in $T_P \cup T_D$. This heuristic is also admissible, as we would in all cases have to travel to a pickup or delivery city in order to deliver all tasks in the end. We believe this approach gives an extremely good lower bound to h^* (true optimal cost to reach a goal node). Indeed, let us suppose that our agent is carrying the last available task, that has to be delivered in city c. In this case our estimate $h = h^*$. Hence, the bound is tight.

3. The third heuristic is defined as follows. Call p_{min} (resp. d_{min}) the minimal cost to move to a city in which it is possible to pick up (resp. deliver) a task. Then, $h(s) = costPerKm \cdot (p_{min} \cdot |T_P| + d_{min} \cdot |T_D|) - rewardSum(T_P \cup T_D)$. This heuristic turns out to sometimes overestimate the cost to reach a goal state and, thus, it is not admissible. For instance, this occurs when the agent is located far away from all the pickup and delivery cities, but all of those cities are very close to each other.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

In this experiment, we wished to compare all of the available deliberative agent algorithms in terms of their optimality and efficiency. These include our BFS' implementation and three different A*'s implementations using the heuristic functions we defined.

The four agents used are defined in the `agents.xml` configuration file, having names `deliberative-bfs`, `deliberative-astar-h0`, `deliberative-main` and `deliberative-astar-h2`. All are configured as instances of the `DeliberativeAgent` class. In order to configure the heuristic choice for each A* agent, we add a `heuristic-id` parameter tag, which can be either 0, 1 or 2.

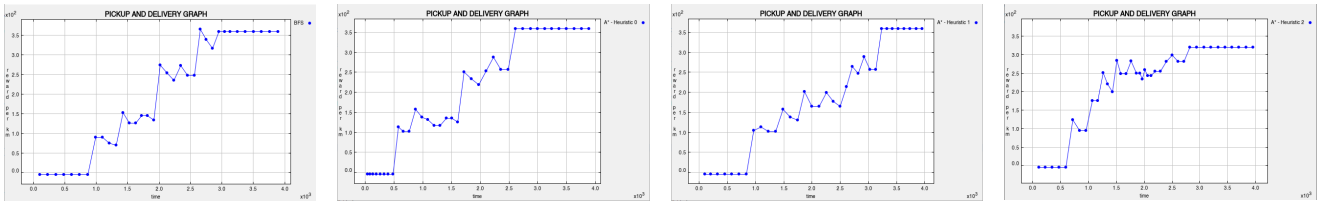

Figure 1: Plots displaying the average reward per km for the BFS and three A* agents on the Switzerland map and 12 available tasks. From left to right: BFS, A* with the first, second and third heuristic variant.

As topology for these simulations we selected Switzerland. Initially in order to estimate the time complexity of the BFS and A* plan construction algorithms, the number of initial spread-out tasks was steadily increased, by modifying the `number` property of the `tasks` tag in the `deliberative.xml` file. Afterwards, this parameter was fixed to the value of 12 to facilitate an equal comparison.

3.1.2 Observations

First, we will elaborate on the results from the efficiency tests. We observed that by limiting the plan construction time to a maximum of one minute, both the BFS and A* algorithms usually can handle iterating through the state space with at most 12 tasks initially available. The only slight exception being the A* agent utilizing the third heuristic, which can handle at most 13 tasks. These results are to be expected considering the exponential growth of the state space size in relation to the number of tasks.

The plots in Figure 1 roughly show that the BFS and A* agents using the first and second h functions do find the same optimal plan. However, the third heuristic is definitely suboptimal. In fact, it was included in order to practically show how easy it is for a non-admissible heuristic function to make the A* algorithm construct a non-optimal plan. As such, we have selected the second heuristic as our default choice, as previously we showed it is admissible and we proved it to lead to better results.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

In order to visualize the competitive and self-interested nature that each deliberative agent expresses, when placed into a multi-agent environment, we ran two different experiments. Firstly, we hoped to analyze the competition between three agents each running a different plan building algorithm. This was achieved by placing the naive, BFS and A* agent with the second heuristic into the Switzerland topology with 12 tasks to deliver in total. Secondly, another simulation was run with three agents utilizing the same A* algorithm and heuristic, competing to deliver 12 tasks across Switzerland.

3.2.2 Observations

A result from the first simulation is displayed on the left plot in Figure 2. When performing multiple runs of this multi-agent experiment, we observed that almost always the naive agent is the worst performer, as the others are understandably more aggressive in picking up and delivering tasks. Since both the BFS and the A* agent find the optimal plan, which one of them outmatches the other purely depends on the vehicle's starting city and the task distribution.

A result from the second simulation is displayed on the right plot in Figure 2. When performing multiple runs of this multi-agent experiment, we observed a much greater competitiveness compared to the previous setting, which is to be expected. Indeed, it was very common to observe agents following the same path behind one another.

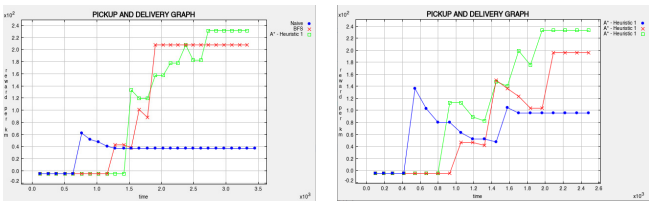

Figure 2: Plots displaying the average reward per km for three agents running competitively on the Switzerland map with 12 tasks. On the left: Naive vs BFS vs A* with `heuristicId` = 1. On the right: three A* agents with `heuristicId` = 1.