Excercise 3
Implementing a deliberative Agent

Group №1 : Célia Benquet - 271518, Artur Jesslen - 270642

October 20, 2020

1 Model Description

1.1 Intermediate States

We implement a **State** class to represent the states. It is defined by the city in which the agent is, the tasks available that are either already picked-up or awaiting to be delivered and the corresponding plan, cost and heuristic at the stage of the state. States internally store the corresponding plan used to get to them. The cost and heuristic are re-computed when needed, based on the total travelled distance of the plan at the time and the tasks that are not delivered yet. Between each action, a new intermediate state should be defined. However in our case, we considered as proper states only the states where we pickup or deliver a task. Picking up and delivering tasks on the way are not considered as proper states as it has the same effect to directly add the action to the next state. Hence, the corresponding actions (picking up tasks and delivering tasks) are added to the computed plan that defines the next state.

1.2 Goal State

A goal state is reached when there are no more task to pick up and no more task to deliver in the environment. To verify this, we check that both the **pickedUpTasks** and **awaitingDeliveryTask** TaskSets from the state are empty.

1.3 Actions

There are 3 types of actions. The agent can either move from one city to a neighbouring city, pick up a task or deliver a task. Hence, moving from a city to another could result in a sequence of moving actions from one city on the path to the next one.

2 Implementation

For both algorithms, the implementation is similar and thus, we grouped them under an abstract superclass **Algo**. They both override the main algorithm method **computePlan** and the computation of the successors takes place in **computeSuccessors**. We go through the different layers of the tree until the best plan with respect to the chosen algorithm is found.

2.1 BFS

The BFS algorithm is implemented in the class **BFSAlgo**. The layers are browsed one by one by adding the successors of a given node at the end of the list of nodes to check. Hence, we are sure that the first possible plan we find is the shortest one. The method **computePlan** computes the BFS algorithm by checking if the current node is a final node. **computeSuccessors** determines the successors of the current state and

add them to the list in order to be checked later in the main algorithm. To find the successors, we browse all the awaiting tasks to compute the path to pick them up and picked up tasks to compute the path to deliver them. For all the steps on the way (i.e. intermediate cities), we consider the intermediate states by checking if tasks could be delivered or picked up during the journey.

2.2 A*

The A* algorithm is implemented in the class **AStarAlgo**. The nodes are checked by starting with the most promising, as the list of nodes to check is sorted with respect to their total cost (i.e. sum of current cost and heuristic cost). Hence, we are sure that the first possible plan we find is the one with the lowest cost. Both override methods have a similar rationale than for the BFS algorithm but the notions of cost and heuristic are added.

2.3 Heuristic Function

The total cost $f(n)$ of a state is implemented as the sum of the heuristic $h(n)$ and the cost $g(n)$ of the plan to arrive to this state: $f(n) = h(n) + g(n)$. $g(n)$ is stored as an attribute of the state similarly to the plan to reach this state and corresponds to $travelledDistance \times cost/km$. The heuristic gives an information about the future cost to get to the goal state. The heuristic we choose consists in $cost/km\times$ the longest distance to travel between:
- The longest distance to travel by the agent to pickup and deliver a task that is awaiting to be picked up.
- The longest distance to travel by the agent to deliver a task that has already been picked up.

The computed heuristic can be considered as optimal. Indeed, there are no scenario where the heuristic function would be overestimated as the distance taken as heuristic would be the minimum travelled distance even if all the tasks are on the same path.

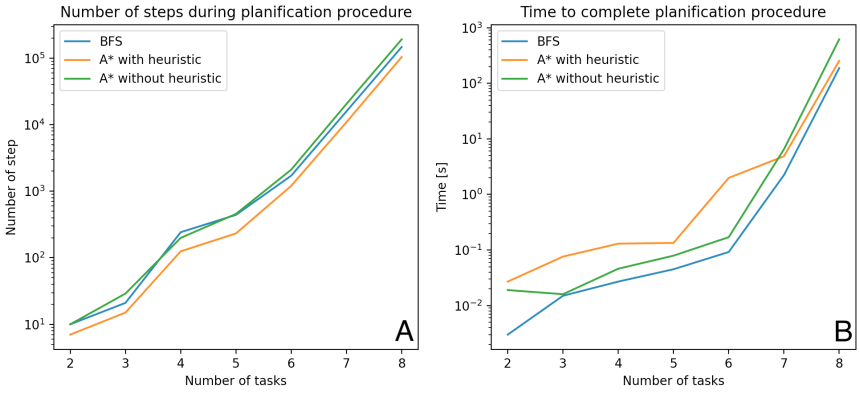

Figure 1: **A.** Evolution of the number of steps required to compute the optimal plan depending on the number of available tasks. **B.** Evolution of the time needed to compute the optimal plan depending on the number of available tasks.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We performed the same experiment in the Switzerland topology on 3 deliberative agents planning respectively with the BSF algorithm, A* algorithm with no heuristic and A* algorithm with heuristic. The vehicles had

a capacity of 30 and a cost per km of 5. We made the number of tasks vary from 2 to 8. Results were tested for different seeds in order to have a random location of the tasks.

3.1.2 Observation 1: BFS and A* Comparison

We observed that whether our deliberative agent applies the BFS or the A* algorithm, it finds the same optimal plan. However, even if it takes more time to be computed (Fig. 1B) as it needs to sort the list of nodes to check at each step, we see that the A* algorithm is always more efficient than the BFS (Fig. 1A), as it has to check less nodes in the queue before finding the optimal plan. From Figure 1B, for both algorithms, we can see that the time to compute the plan goes exponentially with the number of tasks. It means that the algorithms are limited by the number of layers to go through in the tree of possible states.

3.1.3 Observation 2: A* Heuristic Comparison

A* without an heuristic presents results that are highly similar to the results of the BFS algorithm. This proves that our heuristic is admissible and efficient.

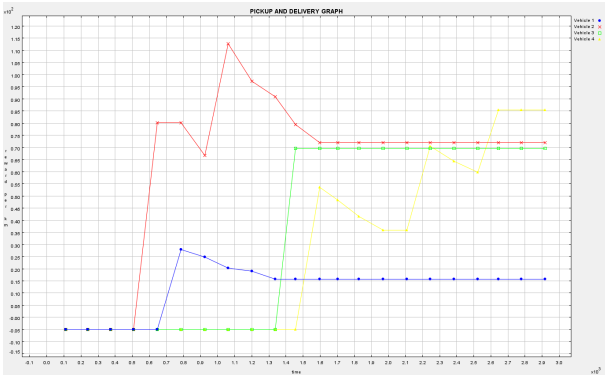

Figure 2: Evolution of the reward per km through time for 4 deliberative agents using A* algorithm for find the plan.

3.2 Experiment 3: Multi-agent Experiments

3.2.1 Setting

We used the Switzerland topology with 7 tasks for 4 agents using the same A* search algorithm and the same heuristic. Vehicles had a capacity of 30 and a cost per km of 5.

3.2.2 Observations

We see that for a multi-agent experiment, some agents are interfering in the plan of others to pick up a task by picking it up before them. It means that some of them have to re-compute their plan multiple times, which consequently makes them lose a lot of time with respect to the others. For the specific experiment we performed, Vehicle 4 (yellow) starts by picking up multiple tasks before delivering them (Fig. 2). As a consequence, it double crosses other agents as it picks up 3 out of 7 tasks while the others only have time to pick up one task and deliver it. Once the other agents are done with their first delivery, they realize that all the tasks are gone already, while Vehicle 4 still delivers its tasks.