Excercise 3
Implementing a deliberative Agent

Group 45 : Lüscher Jérôme, Delisle Ludovic

October 20, 2020

1 Model Description

1.1 Intermediate States

We define the state as a tuple containing the current city of our agent, the tasks that the agent currently carries, the tasks that are still left to pick-up in the environment and the cumulative cost. So the state is : $\langle current\_city, current\_tasks, available\_tasks, current\_cost\rangle$. We made this choice because we think that this tuple is the minimum of necessary information to uniquely identify a state.
A small comment on comparing states: we define comparable states as follows: if the current city, the tasks currently carried and the tasks left to pick-up are the same for two states, they are comparable. Furthermore, the best state among multiple comparable states, is the one with the smallest accumulated cost. Having the smallest cost to arrive at a same "checkpoint" means that the sequence of action was more efficient.
We found it useful to store a list of actions leading to a given state, that way, it is easy to access the actions that led to a specific desired goal state and construct the corresponding plan.

1.2 Goal State

A final state can be defined as a state where the agent has accomplished all the tasks. Thus there is no more task available for pickup and the agent has no more task to deliver. The goal state can be defined as the best state among the final states which is the final state with the lowest accumulated cost.

1.3 Actions

We defined three types of actions: move, pickup and delivery. They are what makes up the plan which is a sequence of actions that the agent will do in order to reach the goal state. For each state, we find all the actions that the agent can do and the algorithm chooses the best one among them. When an agent is in a city where it can deliver a task that it is carrying, the only possible action is delivery. When an agent is in a city where it can pickup a task, it can chose to move to another city or to pick it up, it will depend on the remaining capacity of the vehicle and the future reward prospects.

2 Implementation

2.1 BFS

Given the starting state of the agent, this algorithm finds the best sequence of actions to reach the goal state. First, we find every actions that the agent can make. For each of these actions, we find the state in which the agent will be after doing it. We keep track of the states that were seen by putting them in a map. For every state in the queue, we take them out one by one and apply the same process that was done on the starting state. By always taking the state that is at the front of the queue, we perform the breadth first search layer by layer. The algorithm ends when the queue is empty which means that every possible sequence of actions have been tested.

2.2 A*

The implementation of the A* algorithm is very similar to BFS. The main difference is that we keep a collection of states that is ordered according to their accumulated costs, this is translated in our implementation by the `PriorityQueue` data structure. Each time we add new states to our queue, they are automatically ordered by increasing cost. This means that is we take the first element of the queue, it is necessarily also the one with the lowest cost. This means that we are always exploring the sequence of actions which seems the most promising at this time. Note that when the cost increases above the second element's cost, they switch places in the queue and we explore another path. This is done until we reach a goal state. However, with this method, we may reach the same state multiple times, which increases spatial complexity. To disambiguate, we always check which of the two copies has the lowest cost and we only add this one. This algorithm is thus also optimal.

2.3 Heuristic Function

The A* algorithm can be further improved by a heuristic function which, if it is well suited for our purposes, can greatly improve the algorithm's performance. The heuristic function computes a part of the total cost. The complexity of the heuristic function has to be extremely low in order not to increase the algorithm's running time. We choose a very simple approximation of the remaining cost: $h\_cost = dist * costPerKm * (nb\_tasksAvailable + 2 * nb\_tasksRemaining)$. The Heuristic function preserves optimality, we indeed observe that the result is unchanged.
More complicated and precise options were explored, but we found that even adding a simple for loop over the remaining tasks increased the running time of A* heuristic and thus defeats the purpose of having a heuristic.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

All experiments were performed on the Switzerland topology, with 6 tasks unless otherwise specified.

3.1.2 Observations

We observe that both BFS and A* yield the same optimal plan which is 1380km long for the Switzerland topology with the agent starting in Lausanne. While BFS and A* were able to make a plan for 11 tasks in less than a minute, heuristic A* was able to come up with a plan in much less than a minute for up to 100 tasks. Beyond 100 tasks heuristic A* couldn't always find a plan in less than 60 seconds. Also, to give an idea of performance, heuristic A*, A* and BFS found the optimal plan for 12 tasks in 79s, 2s and 72s respectively. BFS is faster than naive A* because it is a simpler algorithm in its conception. However, we see that A* clearly performs better as soon as we add a heuristic.

3.2 Experiment 2: Multi-agent Experiments

During experiments including multiple agents, we observe that their respective plans have to be recomputed a great number of times. Indeed, as soon as an agent takes the task that another agent was planned to take, all the planing done until this moment becomes obsolete because the agent would simply head to a city where there is no task to pick up anymore. So all the plans have to be recomputed. The tasks and consequently their rewards are now split among them.

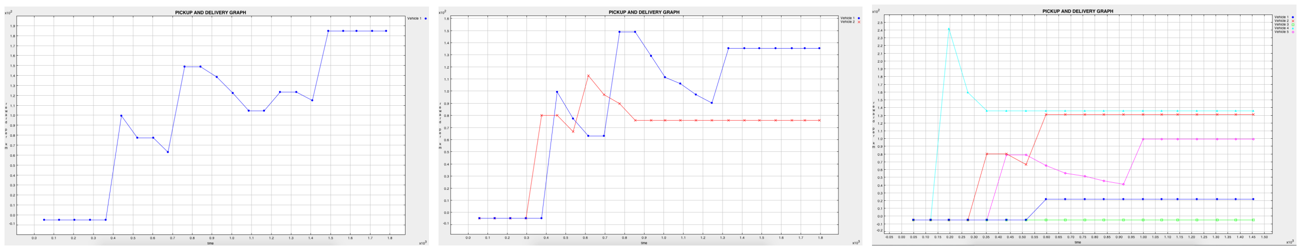

Figure 1: Profit per km plotted against epochs for 1 (left), 2 (middle) and 5 (right) agents respectively. The agents were modified to have the same speed.

3.2.1 Setting

All experiments were performed on the Switzerland topology. The experiments were done with 1 to 5 agents competing for the same tasks. Each agent started in a different city and their speed was set to the same value. The algorithm that we used for this plot was BFS but similar results were obtained with A*. We also tested assigning different speeds to the agents and surely the faster agents were at a large advantage.

3.2.2 Observations

We observe that the final reward per kilometer of the agents is reduced each time we add another agent which means that they loose in efficiency. Moreover, we also observe that the agents reach a final state earlier when they are more numerous.

When we look at the plot with five agents, we see that because the agents operate in a decentralized fashion, there is a lot of inefficiency. For example, the third agent couldn't pick up any task. This is the worst case as this agent now has a negative profit per km. This is an interesting observation, because there are 6 actions in the default configuration for the Switzerland topology, but due to the central initial position of vehicle 2 and 5, they were both able to deliver two tasks. In the end, the agent with the highest reward per kilometer is the fourth agent that picked up and delivered a task very quickly before reaching a final state before the others.

The total distance traveled by the agents is 2770 km, which is a lot higher than the optimal path of 1470 km that a single agents chooses. Note that these figures are not absolute because they largely depend on topology, task distribution and starting cities. But a factor of almost two in the difference of total distance traveled is enough to conclude a large inefficiency in the multi-agent case.

The profit that a single agent makes is 184.5 per km, whereas the maximum profit per km for any single agent in the 5 agent case is 137. So we see that even the best performing agents do not match the single agent's efficiency.