Exercise 3
Implementing a deliberative Agent

Group №34: Carl Hatoum, Raphaël Linsen

October 20, 2020

1 Model Description

1.1 Intermediate States

A state contains the following fields:

- The current city of the vehicle.
- A list of the tasks the vehicle is carrying.
- A list of the tasks the vehicle still has to pick up and deliver.

1.2 Goal State

The goal is to deliver all the tasks, thus the goal State corresponds to the state with the lowest cost among the states where both the carried tasks list and the remaining tasks list are empty. The cost corresponds to the total distance travelled by the vehicle. Indeed since all tasks need to be delivered, the reward from completing them is the same for all final states, therefore minimizing this cost is sufficient.

1.3 Actions

There are 3 possible types of actions from a state to another:

- **Pickup**: The vehicle picks up a given task of the current city. The task is then moved from the remaining tasks list to the carried tasks list. This action is available only if the vehicle has enough capacity left to carry the task. This capacity can be computed from the list of carried tasks.
- **Deliver**: The vehicle delivers a given task to the current city. The task is then removed from the carried tasks list.
- **Move**: The vehicle moves from the current city to a neighbouring one. The only thing that changes is the current city of the state. This is the only action that has a cost, which corresponds to the distance between both cities.

2 Implementation

2.1 BFS

For the Breadth First Search algorithm, we start by generating the initial state node, with the agent's current city, and the list of all the tasks to deliver. We implemented a queue with a linked-list, and for cycle detection, a HashMap (for fast lookup times) of the visited nodes with the lowest cost.

If a node with a goal state is found (with both tasks lists empty), we generate a plan with all the actions that led to this state by going backward from parent to parent, and comparing states to find which action led to each node. We then add the plan as well as the total associated cost to a *plans* HashMap. The node's children are then added to the queue.

When the whole graph have been searched, we may have found multiple plans, to make sure to pick the optimal one, i.e. with the lowest total cost, we search the *plans* Hashmap and return the one with the lowest cost.

Concerning cycle detection, for both BFS* and A*, we check whether the next node is in the list of visited nodes, and if it is, we check whether the cost of the new node is better for reaching the same state. If it is, we replace in the list the old node by the one with the lower cost (or estimate for A*).

2.2 A*

The queue of nodes to be processed is implemented using a PriorityQueue, a structure which is always sorted and sorts added elements automatically. This greatly improves performances comparing to sorting every time. They are sorted using a Comparator, which compares according to the total estimate = previous cost + heuristic.

The first final state found has to be the lowest cost one since the heuristic is admissible. The plan is then reconstituted in the same manner as BFS.

2.3 Heuristic Function

If there are tasks left to be picked up and delivered, the heuristic is defined, among all tasks, to be the longest distance that needs to be travelled to complete the task. This is then equal to the distance from the current city to the pickup city of the task, plus the distance of the task.

If the only ones left are the one carried by the vehicle, the heuristic is the length of the path to the furthest delivery city among the carried tasks.

This heuristic is admissible since the vehicle will always need to travel at least this distance. Indeed to get to the goal state, he needs to deliver all the tasks, including the longest one.

A better estimate might be difficult since this one would already be exact in the cases where the others tasks can be delivered and picked up along the path of the longest task, thus without additional cost.

3 Results

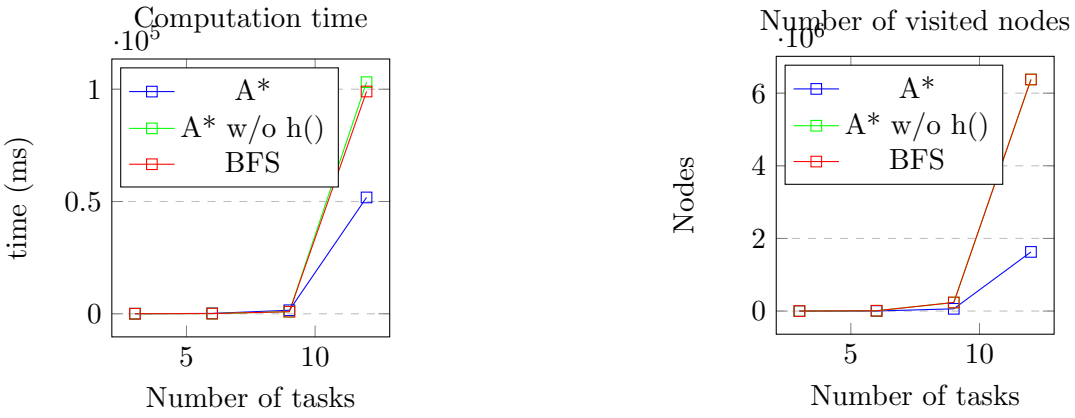
3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We ran both the BFS and A* algorithm on the Switzerland map, with the same seed(23456), for 3, 6, 9, and 12 tasks. Using A*, the plan for 13 tasks could be computed in 40s. For a larger number of tasks, the computation time takes more than a minute. A* without heuristic (i.e. Dijkstra's) is also plotted to show the benefit of the chosen heuristic.

3.1.2 Observations

As we can see, the computation time of plan grows exponentially with the number of tasks for both algorithms. When we plot the number of nodes as a function of tasks, we can clearly notice that the computation time is indeed correlated to the number of nodes, that represents states. An increasing number of tasks yields more possible states, in a combinatorial explosion.



We notice however that the A* algorithm performs better than the BFS as the complexity of the problem grows. For the BFS algorithm, we perform an exhaustive search that works reasonably well only for small problem sizes, but cannot be generalized.

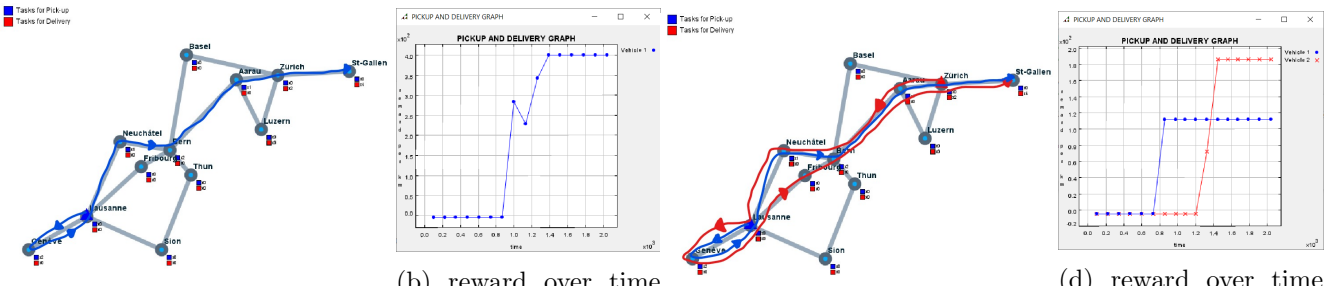
In A*, instead of visiting every single node, we are guided by the estimated cost using a heuristic evaluation, which reduces the time to compute the plan as we can see when we compare it to the performance of A* without heuristic.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

The experiment is conducted with 6 tasks, using seed 34567 on the Switzerland map, with 1, 2 then 3 vehicles, all using the A* algorithm.

3.2.2 Observations



(a) path for one agent

(b) reward over time for one agent

(c) paths for two agent

(d) reward over time for two agent

Figure 2

In this example, it takes one agent 1400 unit of time to complete all the tasks, while it takes two agents 1450. This is due to the fact that in the two agents case, vehicle 2 travels to Genève to discover that the tasks he wanted to pick up there have already picked up by vehicle 1 (see figure 2c), and only after that, he travels to St-Gallen to deliver the task he was carrying.

This illustrates one of the main issues of the fact that the agents do not communicate, and thus cannot coordinate their actions.

With 3 agents it takes 1150 units of time, we start to see an improvement, however this is very dependent on luck since the deliberative method is not optimal in this case.