Excercise 3
Implementing a deliberative Agent

Group №24: Quentin Juppet, Andrea Piccione

October 22, 2019

1 Model Description

1.1 Intermediate States

A state of the deliberative model is generally represented by three variables:
- *currentCity*: the city where the agent is;
- *tasksTaken*: the set of tasks that the agent has already picked up and need to deliver (an agent can carry only a set of tasks whose total weight doesn't exceed the capacity of the vehicle of the agent);
- *tasksLeft*: the set of tasks that the agent has not picked up yet and that are spread in different cities of the simulation.

1.2 Goal State

A goal state is a state where both the sets of *tasksTaken* and *tasksLeft* are empty. In other words, every task has been picked up and delivered.

1.3 Actions

A transition from one state to another can be caused by one of the two possible actions that an agent can take:
- deliver a task by moving to the deliver city of that task;
- pick-up a task by moving to the pick-up city of that task.

Each action causes a change in the state, i.e. it modifies at least one of the variables of the state, and leads to a different state.

2 Implementation

2.1 BFS

The Breadth-first search algorithm has been implemented by using two basic data structures:
- a *Queue* (specifically an *ArrayDequeue*) which stores all the states that need to be processed, one for each step of the algorithm;
- a *Map* (specifically an *HashMap*) which allows to keep track of the states that have been already visited and the least distance to get to it, it is essential to avoid the risk of cycles by stopping if we can get to the same state faster.

The core part of the algorithm is the method used to get all the possible successors of a state. In this case, successors are retrieved by looking at both the *tasksLeft* and the *tasksTaken*. For each task, the full path from the current city of the state (included) to either the delivery city (in case of a task belonging to the *tasksTaken*) or the pickup city (in case of a task belonging to the *tasksLeft*) is computed.

2.2 A*

The A* algorithm has been implemented by using the same search method used for the BFS algorithm. However, the A* needs to sort the states according to a cost function f(n) (f(n) = g(n) + h(n)) and, therefore, a *PriorityQueue* with a defined *State Comparator* has been used in order to keep the states of the queue sorted and process the best solutions first. In a similar way, the A* algorithm needs to check that the same state is not processed twice and has a lower distance, otherwise it would cycle.

2.3 Heuristic Function

The cost function used simply relies on the cost of travelling between cities: the cost of a state is the distance travelled by the vehicle until that state multiplied by the cost per kilometer of the vehicle. In order to compute an estimate the cost, it is necessary to have the total distance travelled from the initial state to the current state n plus an under-estimate (heuristic function) of the distance left to reach the goal state, i.e. the estimate of the cost of the best path from the initial state to the goal state passing though that state n. The chosen heuristic function computes an under-estimate of the future distance by simply taking the maximum distance that we must travel: - The distance to the delivery city for a *tasksTaken* - The distance to the pickup city for a *tasksLeft* plus the minimal distance from the pickup city to the delivery city This heuristic function obviously computes a distance shorter than the one that will be actually travelled and this guarantees admissibility and optimality since h(n) under-estimates the true cost of the path to the goal state.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We tested different numbers of tasks on the Switzerland topology in order to analyze the performance and the limitations of the two algorithms implemented. Table 1 summarizes the configurations tested and the results obtained.

Num. of tasks	Reward per km	BFS	A*
		plan time (ms)	plan time (ms)
5	172	71	27
6	185	584	119
7	201	4901	798
8	229	65875	7889
9	268	-	38627

Table 1: Plan computation time for BFS and A* with different number of tasks in Switzerland topology (seed = 1571680600621).

3.1.2 Observations

The different configurations have been run a sufficient number of times, even with other seeds, and every time we obtained the same outcome: both the algorithms always find the same optimal solution. In terms of efficiency, the A* algorithm has better performance than the BFS algorithm as we expected. In Table 1 we can see that A* is at least 5 times faster than BFS for most of the number of tasks tested. We noticed from the multiple experiments carried out that, up to a certain limit in the number of tasks, both the algorithms take relatively short time to compute the plan. However, after that limit, the time for computing the plan drastically increases unit by unit. This applies mostly for BFS but also A* has a similar behavior and this is due to the number of all possible states that need to be processed, which significantly increases even with by one unit of task in the simulation.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

We tested an increasing number of agents on the Switzerland topology for both the algorithms, A* and BFS, with a fixed number of tasks (6). The goal is to understand the interaction between agents, the impact on the performance and how the re-computation of the plan affects the results.

Num. of agents	BFS		A*	
	plan time (ms)	plan changes	plan time (ms)	plan changes
1	584	0	119	0
2	625	2	139	2
3	824	4	121	4

Table 2: Plan computation time and number of plan changes for BFS and A* with different number of agents in Switzerland topology (number of tasks = 6, seed = 1571680600621).

3.2.2 Observations

As we can see from Figure 1 and Figure 2, the interaction between agents generally leads to a lower reward per kilometer for each agent and it looks like the more the agents, the lower the total reward per kilometer for every agent involved in the simulation. As we also expected, having a higher number of agents leads to more re-computations of plan since there are more occasions to affect the plan of each agent. The plan computation time doesn't change a lot instead, it slightly increases but the difference is not significant.

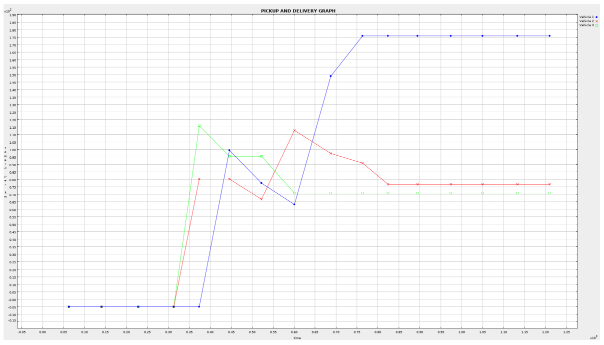

Figure 1: Pickup and delivery graph for three agents using BFS in the Switzerland topology.

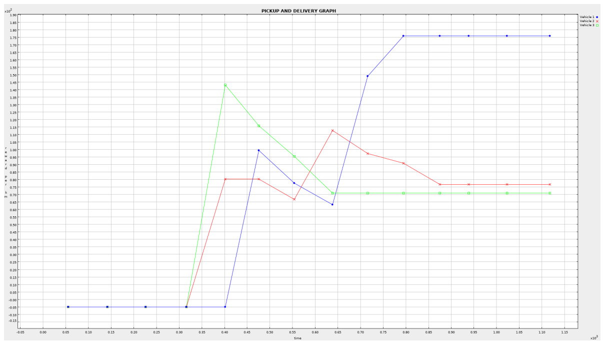

Figure 2: Pickup and delivery graph for three agents using A* in the Switzerland topology.