Exercise 3
Implementing a deliberative Agent

Group №17 : Gervaise Lara, Khatib Karim

October 23, 2019

1 Model Description

1.1 Intermediate States

Our states are defined according to 7 properties :
- The current *position* of the agent.
- The *taskStatus* of all the tasks : 0 = waiting for pick up, 1 = in the vehicle for delivery and 2 = delivered.
- The *totalWeight* the agent is carrying. (If ¿ the capacity, the agent will not pick up additional tasks)
- The total travel *cost* of the agent until the current city. (What we try to minimize)
- The *parentState* is the previous state.
- The *task* identifies which one the agent is picking up or delivering.
- The *pickUp* variable is true if the agent is on its way to pick up a new task or false if it's going to deliver.

1.2 Goal State

The goal state corresponds to when all the tasks have been delivered. With our state representation, it means that all the characters of the string *taskStatus* = 2 in the state.

1.3 Actions

As the agent can carry multiple tasks, we have 2 possible actions : going to a new city to pick up a new task or to deliver a carried task.
The choice depends on the *taskStatus*. The properties of the new state will be decided as follow : the *parentState* is the current state, the *cost* is the previous one plus the one between the current position and the next city, this next city being the new *position* and corresponding to the pick up or delivery city. Finally *pickUp* is directly impacted by the choice and if a task has been picked up or delivered, the *totalWeight* is updated.

2 Implementation

For both algorithms, when an agent realizes that his environment has changed (e. g. another agent has picked a task he could have picked up), we store the tasks it has picked up but not yet delivered in the variable *carriedTasks* and then we execute the plan method again with the remaining unpicked tasks plus the carried tasks.

2.1 BFS

This algorithm is based on linked lists. We start by initializing the task status and the tree. Then, if all the tasks have been delivered, we check if we got a better (lower) cost than the best one obtained

previously. Then from the current state, we create the child states according to the task status. We do that until we explore all the layers of the tree.

2.2 A*

The A* algorithm is based on a predictive exploration of the environment as it relies on a heuristic function in order to decide whether or not it is useful to explore a given state. This means that the final state can be found without necessarily having to explore all of the nodes. It allows faster results and requires less memory than the BFS algorithm which searches the entire node tree before deciding which step to take.
In the case of a deliberative agent, we use a PriorityQueue that sorts its elements based on a comparator that integrates our heuristic function which will be discussed in the next section. This means we always have the least costly state at the top of our list, and when exploring neighboring nodes, we start by exploring those of the top elements. We also use a HashSet in the loop that goes over all the states. Afterwards, extracting a state consists of taking all its elements as presented in the first section and computing its cost.

2.3 Heuristic Function

In our A* algorithm, we implement the cost function of state n f(n) = g(n) + h(n) with g(n) being the cost of state n of the agent so far. h(n) is defined as follow : For each unpicked up task, we compute the sum of the distance from the current city to the pick up city and the distance from the pick up city to the delivery city as candidate heuristic. For each carried task, we do the same but to the delivery city. Finally, we choose the biggest one from all candidates as h(n).
To prove that our algorithm preserves the optimality, we can assume the most extreme case in which all unpicked tasks and carried tasks share an overlapped path, thus the remaining future cost will be exactly the same as our h(n). Any other heuristic functions greater than our h(n) will overestimate in such extreme case. Thus, our heuristic can guarantee optimality as well as satisfactory compute time.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

The experiment has been realized in Switzerland with tasks ranging from 6 to 10.

3.1.2 Observations

Task Number		6	7	8	9	10
BFS	Iteration Number	199957	1837128	11205132	Time out	Time out
	Execution Time (ms)	459	3208	24157	Time out	Time out
	Minimum Cost	6900	8050	8550	NAN	NAN
A*	Iteration Number	3910	29384	161571	522935	3221301
	Execution Time (ms)	124	379	1671	6588	49562
	Minimum Cost	6900	8050	8550	8600	9100

Figure 1: Performance comparison

We can see that the iteration number and execution time of BFS grow exponentially with the number of tasks, while theses numbers grow much slower for A*. We noticed that the maximum number of tasks for which we can build a plan in less than one minute is 10. When the number of tasks is 10, running our A* algorithm to construct the best plan costs 49 s.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

The experiment has been realized with 5 tasks in Switzerland.

3.2.2 Observations

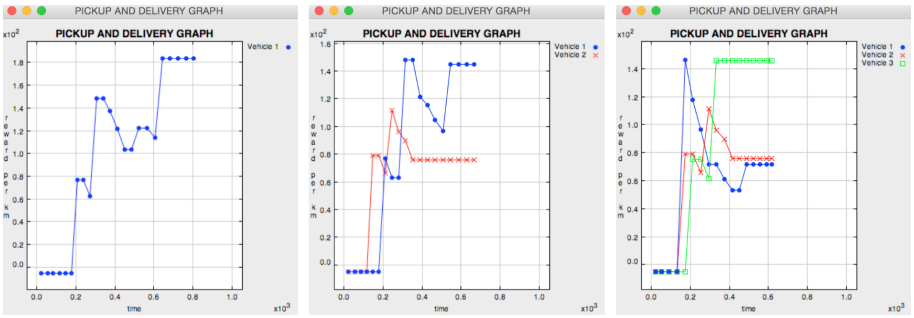

Figure 2: 1, 2 3 agents

On the second graph of **Figure 2**, we can see that agent 2 (red) has picked up some tasks agent 1 (blue) planned to pick up, leading to the decrease of the agent 1 reward.
On the third graph, we observe that agent 3 (green) "steal" some tasks to agent 1 while agent 2 is not impacted as its position is relatively far.
In conclusion, in order to reduce the cost, tasks are usually picked up and delivered by the nearest agent, and other agents just stop considering tasks that have been delivered. Thus having multiple agents increase the efficiency.