Excercise 3
Implementing a deliberative Agent

Group №49: Timur Lavrov, Tim Lebailly

October 22, 2019

1 Model Description

1.1 Intermediate States

In our model we represented states with the attributes outlined below:

- **city**: the current city in which the agent finds itself
- **availableTasks**: the set of available tasks in the world
- **loadedTasks**: the set of tasks currently loaded in the truck
- **parent**: the parent state that the agent found itself in before reaching the current state
- **costDistance**: cost that the agent incurred by getting to the current state from the parent state
- **capacityLeft**: remaining capacity available in the truck
- **vehicle**
- **heuristic**: heuristic cost associated with the current state (only used in A*)

These fields are kept for the purpose of running the different algorithms and being able to compute the optimal plan afterwards, but a state is identified only using 3 fields: city, availableTasks and loadedTasks. It is also worth noting that all costs were calculated taking into account the vehicle's cost per km, when actually this can all be done using only the distance measures (omitting the vehicle's cost per km).

1.2 Goal State

The objective of the agent in our simulation is to reach a goal state in which there are no more *availableTasks* and no more *loadedTasks*. In other words, a goal state is where all tasks have been delivered by the agent.

1.3 Actions

An agent in our simulation has the choice to either pick-up a task in the environment (if it fits into his truck), or to deliver a task that he has loaded. Hence an action naturally incurs a modification in either the list of *availableTasks* (if the agent picks up) or the list of *loadedTasks* (if the agent delivers). Therefore, the successors of a state are those states where a task has been picked-up or delivered. It is worth noting that the *city* can remain the same when transitioning to a new state. This can occur in the situation where a task is available in the initial city the agent finds itself in at the start of the simulation. This is because at the start all tasks are available, and subsequently the agent will pick-up the task that is in the current city and as a result transition to a new state. A similar situation occurs when a task (that the agent can fit into his truck) is available in the delivery City of a task he is currently delivering.

2 Implementation

2.1 BFS

For our BFS implementation we used the pseudo-code given in the lecture slides as a starting point and made multiple modifications. First of all, to ensure that the algorithm finds the optimal solution we keep track of the goal states we come across and retain the optimal one (with the smallest cost) and have the algorithm terminate only once the queue is empty. Additionally, we keep track of the states that we have already visited in a HashMap that maps a unique combinations of *availableTasks*, *loadedTasks*, and *city* to a corresponding state. The given subset of attributes was chosen as a key for this map since we consider two states to be the same if they share identical values in these. Therefore, when a state is popped from the queue, the algorithm verifies whether it has visited it before. If it hasn't, the state's children are added to the queue and a new key/value mapping is added to the HashMap. If it has, the state in the HashMap is updated with the new version only if this new version of the state has a lower cost than the one in the mapping. Since this state was previously visited, its children are already in the queue. If a lower cost was found when revisiting the same state, the cost of its children in the queue need to be updated. Because of this, the first verification that the algorithm performs when a state is popped from the queue is to check whether its parent is in the HashMap, and if it is, it updates the state's parent state to the one found in the HashMap. This ensures that states are updated with their lowest possible costs.

2.2 A*

Our A* implementation also follows the pseudo-code given in the slides. As opposed to BFS, the queue is kept in an sorted order based on the states' heuristics (outlined below). As a result of this ordering, the algorithm can now return the first goal state it reaches as it is optimal (principle idea behind A*). Similarly to our BFS implementation, we also keep track of visited states. The current state's children are added to the queue only if it is a newly visited state or if it has a lower cost than the similar state that was previously visited.

2.3 Heuristic Function

The heuristic function computes the maximum costing task out of all *availableTasks* and *loadedTasks* in a given state. The cost of an available task is the cost of getting from the state's current city to the pick up city of the task plus the cost of getting from the pick up to the delivery city of the task. The cost of a loaded task is simply the cost of going from the state's current city to the delivery city of the task. We believe that taking the maximum out of all of these possible costs is the best option since it would represent the tightest lower bound (from the given set of task costs) to the actual remaining cost until the goal state. This preserves optimality since the chosen task must be delivered before reaching the goal state in any case. Experiments have shown that it is indeed faster to take the maximum out of all of the possible costs rather than taking the first, even though it involves looping over all the possible costs.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We ran simulations with varying number of tasks for both BFS and A*. Only the number of tasks and algorithms were changed accordingly to the test case. All other parameters were left as the default (i.e. topology, seed, probability, reward, weight, etc.).

3.1.2 Observations

# Tasks	BFS (s)	A* (s)	Total Distance
6	0.019	0.009	1380
8	0.143	0.075	1710
10	1.735	0.533	1820
11	8.529	1.784	1820
14	>60	29.846	1820

Figure 1: BFS vs. A*

From the above table, we can observe that A* significantly outperforms BFS in terms of run time for computing the plan. This is expected due to A*'s use of a heuristic to order the queue. As a result, A* visits a smaller number of states in order to reach the optimal goal state. This improvement exceeds the overhead incurred by keeping the queue sorted. The maximum number of tasks for which the algorithms can build a plan in under a minute is 11 for BFS and 14 for A*. Additionally, we observed that in each case both algorithms returned a plan of identical total distance. This is a good indication that both algorithms are returning the optimal plan. Note that here we displayed the total distance of the plan and not the total cost (i.e. we omitted the costPerKm).

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

We run the simulation using 4 agents all using A*, with 10 tasks available and using the default capacity of 30 for each vehicle. We could have used BFS, it does not make a difference since both are fast enough to handle this amount of tasks.

3.2.2 Observations

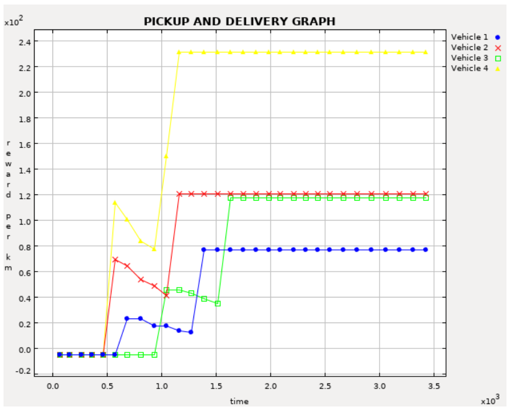

Figure 2: Multi-agent simulation

The results are shown in figure 2. First of all, we can observe that the reward per km of each agent is lower than that of the single-agent simulation. This makes sense since agents will essentially destroy each other's plan in a way that agents will travel to pickup tasks that have already disappeared of the map. In this setting (default seed), the plan was recomputed twice for agent 1, twice for agent 2, once for agent 3 and twice for agent 4. In total, we thus had to compute 7 extra plans as well as the original 4 plans (1 per agent). As such, using multiple agents not only decreases the reward per km (profit of the company), it also is more intensive computationally wise.