Excercise 3
Implementing a deliberative Agent

Group №70: Jelena Banjac, Stratos Triantafyllou

October 22, 2019

1 Model Description

A deliberative agent has an explicit model of the world. Starting from an initial state, we systematically search through all possible sequences of actions and intermediate states, aiming to reach an optimal goal state. In the subsections that follow, we describe our representation of the problem.

1.1 States and transitions

In our model, a state represents a point of the action sequence, where the vehicle is located at a specific city (*currentLocation*). In each state, there is a list of tasks that the vehicle has picked up but has not yet delivered (*tasksToDeliver*), as well as a list of the tasks that it still has to pick up (*tasksAvailable*).

At each state, we consider two types of successor states: moving to a city where one of the tasks being currently carried can be delivered, or a city where some of the still available tasks can be picked up. Notably, a vehicle does not have to deliver a task before picking up another, as far as the total weight carried does not exceed the capacity of the vehicle. We do not explicitly consider states in neighbor cities, although this is automatically addressed by specifying moves to neighbor cities using the shortest paths to delivery/pickup points.

The sequence of actions are recorded in a list (*actions*), that is used to construct the plan later. The cost of the movements to reach that state is also recorded and used to determine the optimal final state in the end.

1.2 Goal States

As goal states we consider states where all of the carried tasks have been delivered and there are no other tasks available in the world. As such, a goal state has no successors.

The optimal goal state is the one where the cost spent to get to this state is minimal in comparison to other goal states.

2 Implementation

2.1 BFS

We use a regular queue-based implementation of the breadth-first search algorithm. In order to avoid cycles, we use a list of unique states (based on the current location of the vehicle, the tasks currently carried and the tasks still available in the world). For each generated successor state, we check if it already exists in that list with a smaller cost to reach it. In the end, we consider the final state with the minimum cost of reaching it.

2.2 A*

The A* implementation was done according to the algorithm we initially received. We implement the sorting of the queue in the `State` class where it is done by comparing two heuristics, the current one and the next one. It is used as a callable which was overriden from the Comparable class.

2.3 Heuristic Function

The main idea of the heuristic function was to find the highest cost depending on that whether the task was available or supposed to be delivered. In short, it just finds where the cost is maximal and when it finds it, it will be additionally increased by the current cost of the state.

2.4 Multiple Agents

When there are multiple agents operating in the world, it can happen that an agent reaches a city to pick up a task that has already been picked up by another agent. In that case, *planCancelled()* is called and then the plan is recomputed. For the recomputation, we consider a new initial state which corresponds to the current city of the vehicle and only addresses tasks that have not been picked up or delivered yet.

3 Results

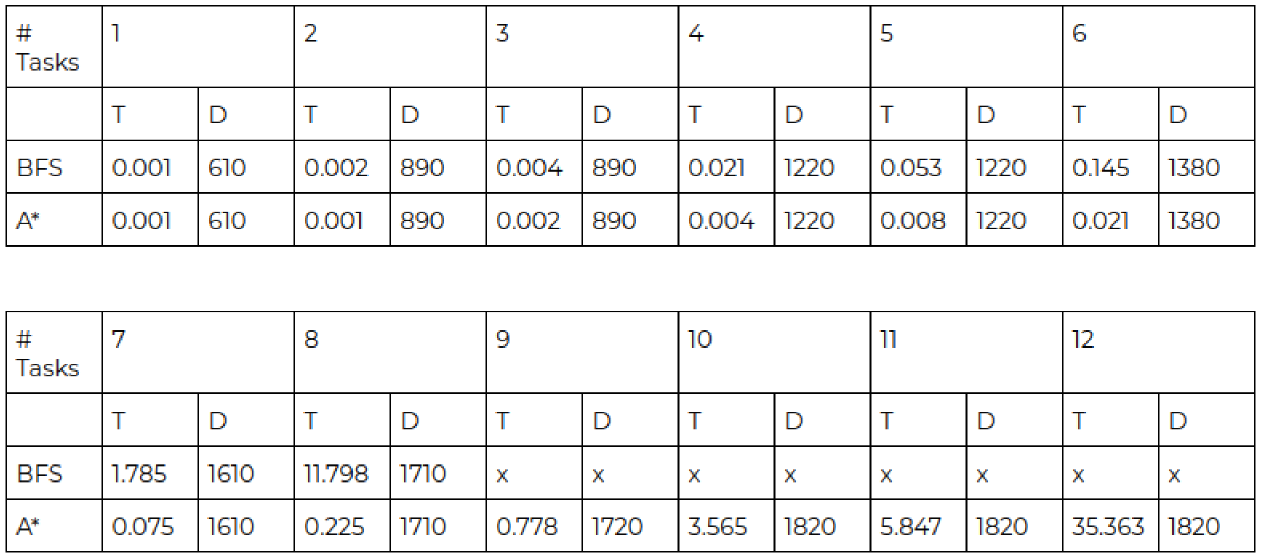
3.1 Experiment 1: BFS and A* Comparison

In this experiment we will compare the two algorithms in terms of: optimality, efficiency, limitations.

3.1.1 Setting

The settings used are the ones that were initially received. Only the number of tasks was varied in order to compare the performances of these algorithms. - Topology: switzerland.xml - Initial city is Lausanne - rngSeed value is 2345 - cost per km is 5 - capacity 30 kg

3.1.2 Observations



# Tasks	1		2		3		4		5		6	
	T	D	T	D	T	D	T	D	T	D	T	D
BFS	0.001	610	0.002	890	0.004	890	0.021	1220	0.053	1220	0.145	1380
A*	0.001	610	0.001	890	0.002	890	0.004	1220	0.008	1220	0.021	1380

# Tasks	7		8		9		10		11		12	
	T	D	T	D	T	D	T	D	T	D	T	D
BFS	1.785	1610	11.798	1710	x	x	x	x	x	x	x	x
A*	0.075	1610	0.225	1710	0.778	1720	3.565	1820	5.847	1820	35.363	1820

Figure 1: Performance depending on different number of tasks for each of the algorithms

From the table above we can see that the BFS algorithm manages to find the optimal solution in 1min constraint for maximum of 8 tasks. Whereas, the A* manages to calculate the optimal distance in 1 minute for 12 tasks. It takes the algorithm around 36 seconds.

3.2 Experiment 2: Multi-agent Experiments

In this experiment we will observe the multi-agent performance.

3.2.1 Setting

The settings were following: - 2 agents (one BFS second A*) - 6 tasks - Topology: switzerland.xml - Initial city for BFS agent is Lausanne - Initial city for A* agent is Zürich - rngSeed value is 2345 - cost per km for both of them is 5 - capacity for both of them 30 kg - speed for both of them is 90

3.2.2 Observations

From the observed results we can see that the computation time increases a bit. From experiment 1 we see that the BFS needs around 0.145 seconds to determine the optimal plan, whereas, now it takes 0.15 seconds. This is because of the additional time it takes to recompute the plan after a clash. The optimal plan distance was 1380 and it stayed the same now. Regarding the A* algorithm (note, now it starts from Zurich) it takes around 0.001 seconds to get the optimal plan of 1190 km. But in the previous experiment, it took around 0.021 seconds to calculate it.