Excercise 3
Implementing a deliberative Agent

Group №: Bruno Wicht, Adrian Valente

October 24, 2017

1 Model Description

1.1 Intermediate States

The important factors that the agent has to take into account are the distribution of tasks (those that have not been picked up, and those that have already been picked up) and its current position. Thus our state representation will be:

$$(p, (t_1, \ldots, t_n), (t'_1, \ldots, t'_m))$$

where p is a `City` representing the current position, $(t_1, \ldots, t_n)$ is a `List<Tasks>` representing the tasks that haven't been picked up yet, and $(t'_1, \ldots, t'_m)$ is also a `List<Tasks>` representing the tasks that the agent is currently transporting. To our class `State` we have added as helper variables the list of actions took from the root to the current state, and the total cost of that path.

1.2 Goal State

A goal state is simply a state where the two lists of tasks are empty (there can be several goal states, depending on the city where we end up).

1.3 Actions

For the actions, we use the representation given by Logist. There are three types of actions: Move (to a neighbor), Pickup and Delivery.

The actions that can be taken at a given state are: moving to one of the neighbors, picking up any combination of the available tasks, and delivering a packet. However, since delivering a packet nevers causes an overcost, we

always do it as soon as possible. Finally, there is no need to compute all combinations of packets to pick up: we simply create one pick up branch for each packet, and the combinations will naturally appear during the next branchings of the tree.

2 Implementation

2.1 BFS

The implementation of BFS is really straightforward: it suffices to visit a node, list all its children wich we add to a queue, and then move to the next node in the queue. Since it is possible to visit a same state several times, which can lead to cycles, we check at each step if we have already visited the current state.

2.2 A*

The algorithm is similar to BFS, except that this time we visit first the nodes which have the smaller `d()` value (that is, the sum of the cost of past actions and of the heuristic). In order to enforce that visiting order, nodes are added to a `PriorityQueue`.

2.3 Heuristic Function

Our heuristic function is defined by the *maximal distance that has to be covered to deliver a task*. That distance is the distance between the current city and the delivery city for a task that we are delivering, and the distance between the current city and the pickup city plus the distance between the pickup and the delivery city for a task that we haven't picked up yet.

Clearly, that function is smaller than the actual cost of the smaller path between the current state and a goal state. Indeed, the agent will have to cover at least the distance defining the heuristic. Thus, our heuristic function is admissible, ie. preserves optimality of the search algorithm.

However, one could imagine more precise heuristics. For example we have thought of using the length of the shortest path going through all cities in which a task has to be delivered or picked up. That heuristic is also admissible, and estimates better the value of a state. Indeed, with our heuristic, getting closer to a task that does not maximize the heuristic does not bring any apparent improvement. But the cost of computation clearly outweighs the advantages of that other heuristic.

3 Results

3.1 Experiment 1: BFS and A* Comparison

First, let us note that the BFS algorithm does not always compute the optimal solution in terms of kilometers covered. Indeed, if we give it a single task to carry from Lausanne to Bern, the BFS algorithm will carry it through Neuchtel, which is slightly more expensive than going through Fribourg.

The difference in efficiency is staggering: while the BFS algorithm usually needs around 8 seconds to handle computations for 6 tasks, the A* algorithm finds the optimal plan in a little more than 0.5 seconds. The BFS algorithm always needs more than one minute for 7 tasks, while the A* algorithm needs a few seconds. The A* algorithm needs more than one minute for 9 tasks (tests carried on MacBookPro 2010, Intel Core 2 Duo, random seeds=1,2,3).

3.1.1 Setting

We have carried our test on the Swiss topology, with multiple task configuration (several random seeds and numbers of tasks).

3.1.2 Observations

As we have said, the BFS algorithm does not always find the optimal plan, thus the number of kilometers covered is always smaller with the A* algorithm.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

We will use the Swiss topology, with 2,3 and 4 tasks, and 2 and 6 agents.

3.2.2 Observations

We observe that the lack of cooperation results in inefficiencies. Agents constantly travel to cities where the tasks have already been picked up. Sometimes, random distribution of tasks cause one agent to do almost all the work. However, when adding a lot of agents, they learn at the first recomputation of the plan that all tasks have been picked up, and there are somehow less inefficiencies.