Excercise 3
Implementing a deliberative Agent

Group №17 : Thibauld Braet, Nathan Greffe

October 23, 2018

1 Model Description

1.1 Intermediate States

The state representation is quite simple, it is composed of:

1. the city the vehicle is currently in
2. the list of tasks we still have to pick up
3. the list of tasks we have picked up but not delivered yet

We do not represent the already delivered tasks since we don't care about them anymore. Although the different combinations of these 3 define a lot of states, the amount of states we are interested in from a given starting state is relatively limited (see Section 1.3).

1.2 Goal State

A terminal state is achieved once there are no more task to pick up and we have delivered all our picked up tasks. The goal state(s) are the terminal states with the lowest cost (lowest total travelled distance).

1.3 Actions

The only sensible actions are to go to a city and pickup a task (without exceeding the vehicle capacity) or to deliver one of the tasks we have picked up. This means that for a state with N picked up tasks and M available tasks, we have at most $N + M$ children (bound on the branching factor) and are $N + 2 * M$ actions (bound on the depth of the tree) away of a terminal state (no matter which actions we choose). This means that all the terminal states are on the same depth. We can also see from the state definition that no cycles are possible.

2 Implementation

The states are implemented by a class that is responsible of the creation of the children of a state. This class is also hashable, can be checked for equality (equal if in same city and having the tasks lists), can be ordered (by using the sum of the cost (total distance from initial state to the state) and the heuristic). It also contains a pointer toward it's parent State such that we can compute the plan from the final states.

For both BFS and A*, we can sensibly reduce the computation times by eliminating the more costly of two equivalent states (same city and list of picked up/available tasks but different path).
If it's the second equivalent state that has the worst cost, we can avoid to add it to the queue. For the

opposite case, we should remove the previous state from the (priority) queue which is a $\Theta(n)$ operation for both A* and BFS. This is too costly for what it brings (we tested with and without) and so we do not do it.
For A*, not removing it actually doesn't even has non negligible disadvantages since we have a priority queue. If we denote the two equivalent states S_1 and S_2, with $cost(S_1) > cost(S_2)$ and S_1 having been put in the queue before S_2, S_2 will be popped before S_1 (better cost), its children will be added to the queue and when S_1 will be popped, his children won't be added to the queue since there will already be equivalent states with a better cost (S_1's children).

2.1 BFS

For the BFS, we maintain a Queue (linked list) for the nodes to visit and a HashMap to hold all of the States that have been in the queue and their score. For every element of the queue, if it is non terminal, we compute it's children and add them to the queue if they should be, while maintaining the Hashmap. If it is terminal, we check if it's the best terminal state we have encountered so far and keep it in that case. Once the queue is empty, we return the Plan associated with the best final state.

2.2 A*

The main difference with the implementation of BFS is that we now maintain a Priority Queue over the States (ordered by the sum of the cost and the heuristic). Once we found a terminal state, we stop since the first terminal state is guaranteed to be optimal as long as the heuristic is admissible.

2.3 Heuristic Function

The idea of the heuristic function is to take the longest path of size 1 or 2. What we mean by longest path of size 1 is the distance needed to deliver one task without caring about the others (distance to delivery city for a picked-up task and pathLength of not picked-up task). What we mean by path of size 2 is a path of size one whose delivery city is A + the path associated with a task not yet picked up whose pickup city is A.
The heuristic needs to be computed fast so we have to use something simple with the costs of the paths, we first only considered paths of size 1 but then thought about paths of size 2 since cities that have pickup and delivery tasks are not that uncommon. We could also have increased the size of the maximal path but there is a computational tradeoff here and paths of size 3, 4, ... doesn't seem to be that common. We cannot combine paths otherwise than if $1.delivery == 2.pickup$ since we could have situations where $1.pickup, 2.pickup, 3.pickup, 3.deliver, 2.deliver, 1.deliver$ are on the same line.
This heuristic is admissible since it always underestimates the true cost, indeed the heuristic express the max cost of delivering one of the packages or two of them that are consecutive (without taking capacity into account) which is always lower or equal to the cost of delivering all of them.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We used a uniform task weight of 3, Switzerland as topology, a $Uniform[0, 1]$ probability distribution and a constant reward distribution encouraging short distances between 100 and 99999 and started from Lausanne.

3.1.2 Observations

We can conclude that A* is indeed orders of magnitude faster than BFS.
As we can see on Table 1, A* can handle up to 13 tasks while BFS cannot go further than 8 (both with a capacity of 30). A* results vary a lot with the seed used (A* cannot go further than 9 tasks with a seed of 23456 and a capacity of 30). This is not that much the case for BFS since he has to explore the whole tree anyway. The capacity of the vehicle, which determines (at fixed task weight) the number of tasks a vehicle can carry at a time also plays an important role since we can limit the number of possible actions and therefore reduce the times by modifying it.

Tasks	Seed	Vehicle's capacity	A* time	BFS time	optimal cost
7	15	30	$< 0.5s$	$< 0.5s$	$1260km$
8	15	30	$< 0.5s$	$6s$	$1340km$
8	15	10	$< 0.5s$	$< 0.5s$	$1380km$
8	23456	30	$1s$	$7s$	$1710km$
9	15	30	$< 0.5s$	$??s$	$1340km$
9	23456	30	$3s$	$??s$	$1720km$
10	15	30	$< 0.5s$	$??s$	$1520km$
10	23456	30	$??s$	$??s$	$??km$
10	23456	10	$< 0.5s$	$??s$	$2020km$
11	15	30	$1s$	$??s$	$1520km$
12	15	30	$4s$	$??s$	$1520km$
13	15	30	$29s$	$??s$	$1520km$

Table 1: A* vs BFS: perfomrances comparison

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

Same as above with capacity = 30, 8 tasks and seed = 15, we used the A* algorithm, the first vehicle starts in Lausanne, the second in Zurich, the third in Bern and the fourth in Basel.

3.2.2 Observations

We can see the path is far from being optimal since the agent only reacts when he goes to a city where the task he expected to find has disappeared. The bad results are perfectly normal since the algorithm is not suited to handle these problems at all. To get an idea of how much suboptimal the problem is, we can compare the total travelled distance (history.xml file) from every vehicle when we increase the number of vehicles for the same tasks. With the given settings and one vehicle, the total travelled distance is $1340km$, with two vehicles, $1130 + 1080 = 2210km$ and with 4 $1050 + 720 + 920 + 750 = 3440km$. So we can observe that the situation get worse (almost proportionally) the more agents there are (which is logical since the number of plans recomputations increases as well).