Exercise 2: A Reactive Agent for the Pickup and Delivery Problem

Group №: 48
Alexandru Mocanu, Ivan Yurov

October 8, 2019

1 Problem Representation

1.1 Representation Description

When choosing a representation of the world, our goal was to find an injective mapping from the state of the world, as observed by the agent, to the state space that we would use for finding the agent's strategy. Let us denote by N the number of cities.
We consider the state as a *(city, available_delivery)* tuple, where *city* is the city that the agent is currently present in, and *available_delivery* is either the number of the city that we have a delivery for from *city*, or N in case no delivery is available. The state is mapped into an integer as *city.id*(N+1)*.
Actions are categorized into move actions and pick-up action. The move actions are numbered from 0 to *N-1* corresponding to the city number that we want to move to, while the pick-up action is numbered N.
For a (state, action) pair, where state = (c_1, d_1), we have the following rewarding scheme:

- move action to city c_2 - The reward is minus the cost of going from c_1 to c_2, if c_1 and c_2 are neighbors.
- pick-up action - If there is some available delivery in c_1 for city c_2, the reward is the value of the delivery minus the distance from c_1 to c_2.
- The rest of the cases are invalid, so there is no reward defined for these.

For a (state, action, next_state) tuple, where state = (c_1, d_1) and next_state = (c_2, c_2), we have the following transition probabilities:

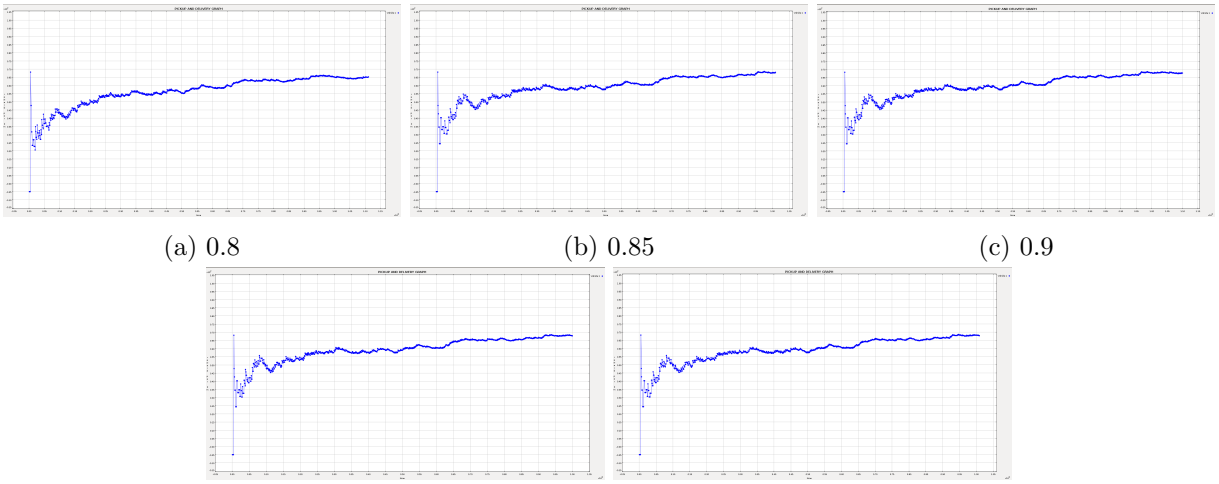
- move action to city c_2 - The probability is that of finding a delivery to d_2 from city c_2.
- pick-up action if $d_1 == c_2$ - The probability is again that of finding a delivery to d_2 from city c_2.
- For the rest of the cases, the transition probability is zero.

1.2 Implementation Details

The reward and transition tables are stored as multidimensional arrays and are indexed by states and actions, which are represented as integers like we presented above.
For the value iteration algorithm, we store the Q and V values in arrays, as well as an array of best actions for each of the possible states. This final array is then used by the agent to decide what action to take when reaching some city.

Discount factor	Number of steps
0.80	44
0.85	58
0.90	88
0.95	177
0.99	888

Table 1: Number of steps for the convergence of value iteration



(a) 0.8 (b) 0.85 (c) 0.9

(d) 0.95 (e) 0.99

Figure 1: Evolution of reward per km for different discount factors

2 Results

In all the experiments we use the map of France with the random seed provided in the skeleton.

2.1 Experiment 1: Discount factor

2.1.1 Setting

We used discount factors of 0.8, 0.85, 0.9, 0.95 and 0.99, running single agent simulations.

2.1.2 Observations

Figure 1 shows the evolution of the reward per km for different discount factors.
In all the settings, we observe an initial spike of about 68, due to the first delivery. The reward density then plummets and starts increasing gradually, approaching somewhat of a plateau after about 80k units of time. We also note that the reward density reaches higher values for discount factors above 0.8. For the other factors, the agent seems to learn similar if not identical policies, which lead to almost the same plot.
The main difference between the runs with different discount factors is the number of steps for value iteration to converge:

2.2 Experiment 2: Comparisons with dummy agents

We compare our agent to two less complex ones, a random agent and a dummy agent that delivers any package that it finds or moves to the closest neighboring city if there is no package to deliver.

(a) One random agent (b) One dummy agent (c) One smart agent

(d) Two random agents (e) Two dummy agents (f) Two smart agents

(g) Three random agents (h) Three dummy agents (i) Three smart agents

Figure 2: Evolution of reward per km for different agent types

2.2.1 Setting

We use one, two and three agents respectively from each of the types mentioned above. For the smart agent, we set the discount factor to 0.95.

2.2.2 Observations

As we can see from figure 2, the smart agent outperforms the other ones. The dummy agent still performs better than the random agent, as expected. It may seem that the dummy agents perform better than the smart one when two or three agents are run at the same time. However, as time goes by, the dummy agents tend to converge to the performance of a singly run agent. This convergence of the performance of multiple agents to that of an individual agent takes time mainly due to the agents starting in different positions.